

Arduino Programming with .NET and Sketch



Agus Kurniawan

Apress®

Arduino Programming with .NET and Sketch



Agus Kurniawan

Apress®

Arduino Programming with .NET and Sketch

Agus Kurniawan
Depok
Indonesia

ISBN-13 (pbk): 978-1-4842-2658-2
DOI 10.1007/978-1-4842-2659-9

ISBN-13 (electronic): 978-1-4842-2659-9

Library of Congress Control Number: 2017936052

Copyright © 2017 by Agus Kurniawan

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Natalie Pao
Development Editor: Jim Markham
Technical Reviewer: Fabio Claudio Ferracchiati
Coordinating Editor: Jessica Vakili
Copy Editor: Larissa Shmailo
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover image designed by Freepik

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit <http://www.apress.com/rights-permissions>.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/us/services/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484226582. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

Contents at a Glance

About the Author	ix
About the Technical Reviewer	xi
Acknowledgements	xiii
Introduction	xv
■ Chapter 1: Introduction to Arduino Boards and Development.....	1
■ Chapter 2: Interfacing .NET and Arduino	21
■ Chapter 3: Sensing and Actuating	45
■ Chapter 4: Windows Remote Arduino	69
■ Chapter 5: Building Your Own IoT Using Arduino and .NET	109
Index.....	165

Contents

About the Author	ix
About the Technical Reviewer	xi
Acknowledgements	xiii
Introduction	xv
■ Chapter 1: Introduction to Arduino Boards and Development	1
Exploring Arduino Boards	1
Arduino Boards for Beginners.....	2
Arduino Boards for Advanced Users	3
Arduino for Internet of Things.....	4
Arduino-Compatible.....	7
Setting up Your Development Environment.....	9
Build your First Project: Blinking	11
Sketch Programming.....	15
Arduino Programming Using Visual Studio.....	15
Summary.....	19
■ Chapter 2: Interfacing .NET and Arduino	21
Arduino I/O Communication	21
Serial Communication - UART	22
How the Program Works	25
SPI Communication	26
How the Program Works	29

■ CONTENTS

TWI/I2C Communication	29
How the Program Works.....	33
Control Arduino Board from .NET	34
How the Program Works.....	38
Introducing Firmata Protocol.....	39
Summary	44
■ Chapter 3: Sensing and Actuating	45
Overview of Sensing and Actuating in Arduino	45
Exploring Sensor and Actuator Devices	46
Sensor Devices	46
How the Program Works.....	52
Actuator Devices.....	53
Creating an Arduino Sensing App Using .NET.....	58
How the Program Works.....	60
How the Program Works.....	63
Creating an Arduino Actuating App Using .NET	65
Summary	68
■ Chapter 4: Windows Remote Arduino.....	69
Setting up Arduino for Windows Remote Arduino	70
Building Your First Program for Windows Remote Arduino	71
Wiring	71
Arduino Program.....	72
.NET Application Program	73
Adding Windows Remote Arduino Library	75
Writing .NET Program	77
Testing	82

Control Arduino Analog I/O	85
Wiring	86
Creating a UWP Project.....	87
Arduino Program.....	87
Writing the UWP Program	87
Testing	91
Remote Arduino Through I2C Bus.....	92
Wiring for I2C Application	93
Creating a UWP Project.....	94
Writing UWP Program	95
Testing	99
Windows Remote Arduino Over Bluetooth.....	100
Wiring for WRA with Bluetooth	101
Pairing Arduino Bluetooth and Computer	102
Creating a UWP Project.....	104
Writing an Arduino Program	105
Writing a UWP Program	105
Testing	107
Summary	107
■ Chapter 5: Building Your Own IoT Using Arduino and .NET	109
Introduction to Internet of Things and Arduino.....	109
Connecting Arduino to Internet Network	110
Connecting to a Wired Network.....	110
Connecting to a WiFi Network	114
Accessing Arduino over a Network from .NET Application	121
Wiring	122
Building a Sketch Program.....	124

Building a UWP Application.....	129
Testing	133
Windows Remote Arduino (WRA) over WiFi	134
Configure Arduino for WRA over WiFi.....	134
Building a UWP Application.....	135
Testing	140
RF Communication for Arduino.....	141
Configuring XBee IEEE 802.15.4	143
Building an Arduino Sketch Program.....	145
Building a UWP Program.....	146
Testing	151
Building a LoRa Network for Arduino	152
Location-based Application for Arduino	156
Arduino and Cloud Server	161
Arduino Cloud	161
Summary.....	163
Index.....	165

About the Author

Agus Kurniawan is a lecturer, IT consultant, and an author. He has 16 years of experience in various software and hardware development projects, delivering materials in training and workshops, and technical writing. He has been awarded the Microsoft Most Valuable Professional (MVP) award 14 years in a row.

He is currently doing some research and also getting involved in teaching activities related to networking and security systems at the Faculty of Computer Science, Universitas Indonesia and Samsung R&D Institute, Indonesia. Currently, he is pursuing a PhD in computer science at the Freie Universität Berlin, Germany. He can be reached on his blog at <http://blog.aguskurniawan.net> and on Twitter at @agusk2010.

About the Technical Reviewer

Fabio Claudio Ferracchiati is a senior consultant and a senior analyst/developer using Microsoft technologies. He works for React Consulting (www.reactconsulting.it). He is a Microsoft Certified Solution Developer for .NET, a Microsoft Certified Application Developer for .NET, a Microsoft Certified Professional, and a prolific author and technical reviewer. Over the past ten years, he's written articles for Italian and international magazines and coauthored more than ten books on a variety of computer topics.

Acknowledgments

We would like to thank Apress for all their help in making this book possible. Specifically, we would like to thank Natalie Pao and Jessica Vakili, our coordinating editors, for helping us stay focused and overcoming many obstacles. Without them, this book would not have been possible.

Special thanks to James Markham, our development editor, and Fabio Claudio Ferracchiati, our technical reviewer, for all his suggestions during the editorial review process to help make this a great book.

We would also like to thank the Arduino and .NET communities anywhere in the world for contributing and making learning Arduino programming easy.

Last, but not least, a thank you to my wife, Ela, and my children, Thariq and Zahra, for their great support in completing this book.

Introduction

Arduino is a board development platform with which we can develop an embedded application with several sensor and actuator devices. Arduino is an open source-based hardware. There are many Arduino models that you can use to develop. This book is designed for developers (especially for .NET developers) that want to build Arduino programs for general and specific purposes.

For the Readers

This book assumes you may have previous programming experience. The book is also written for someone who may have developed programs using .NET and wants to develop an embedded program with Arduino boards.

How This Book Is Organized

This book is designed with a step-by-step approach. You will learn how to build Arduino programs using sketch and .NET, and explore Arduino capabilities such as digital and analog I/O processing, serial communication, SPI, and I2C bus.

You will find out how .NET collaborates with sketch programs on Arduino to control sensor and actuator devices remotely. The Internet of Things (IoT) topic is introduced, including its implementation. Finally, a cloud server is used to connect to the Arduino board.

Required Software, Materials, and Equipment

In general, a computer with Windows OS. Windows 10 installed is recommended. You should install Arduino software and Visual Studio on your computer.

You need several Arduino models to implement our demo. Furthermore, you should provide several sensor and actuator devices and several network modules such as WiFi, Bluetooth, GPS, and LoRa.

CHAPTER 1



Introduction to Arduino Boards and Development

Arduino is one of the most famous development boards. You can attach sensor and actuator devices easily into the board. This chapter will explore how to work with Arduino development and prepare for a development machine. To work on this chapter, you should have one of the Arduino board models for implementation.

This chapter covers the following topics:

- Exploring Arduino boards.
- Setting up development.
- Building your first project.
- Sketch programming.
- Arduino programming using Visual Studio.

Exploring Arduino Boards

Arduino is a board development platform with which we can develop an embedded application with several sensor and actuator devices. Arduino is an open source-based hardware. It means we can develop our own Arduino board, but you should not use the Arduino name because it's a trademark. Currently Arduino boards are produced by Arduino LLC (www.arduino.cc) and Arduino SRL (www.arduino.org). Some Arduino models which are produced by Arduino LLC and Arduino SRL are different.

In general, an Arduino board has several digital and analog I/O pins, which are used to sense and actuate with external devices. In addition, Arduino provides UART, SPI, and I2C protocols. Each Arduino model has unique features and forms. Make sure you don't choose the wrong board model. One of the Arduino board samples can be seen in Figure 1-1.

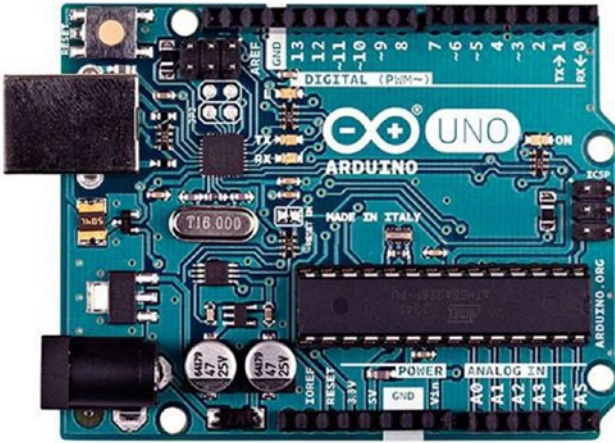


Figure 1-1. *Arduino UNO R3 board*

The advantage of an Arduino board is it's easy to use. You don't need to solder electronics components. An Arduino board is ready to use. You just attach sensor and actuator devices into the board via jumper cables.

In this section, we explore various Arduino boards from Arduino LLC and Arduino SRL. Each Arduino model has unique features. To optimize Arduino board usage, you should know and understand what kind of Arduino model it is. I will introduce various Arduino models based on complexity level and usage range.

Let's start to explore Arduino boards.

Arduino Boards for Beginners

An Arduino UNO board is a development board which I recommend to anyone who wants to learn Arduino programming. There are many Arduino shields which are attached to the board. Furthermore, most tutorials and books use Arduino UNO as an experimental board. Arduino UNO has completed I/O protocols, such as digital and analog I/O, SPI, UART, and I2C/TWI, so you can utilize these I/O pins to work with sensor and actuator devices. Arduino UNO is easier to find and buy.

You can review the Arduino UNO board on this website: <https://www.arduino.cc/en/Main/ArduinoBoardUno>. You can also review Arduino UNO from Arduino SRL on this site: <http://www.arduino.org/products/boards/arduino-uno>.



Figure 1-2. *Arduino UNO board from Arduino LLC*

Arduino Boards for Advanced Users

In some cases you want to optimize your board's I/O or want to debug your programs. The Arduino MEGA 2560 board provides more I/O pins (about 54 I/O pins) and advanced MCU to accelerate your program. This board runs on the top of MCU Atmega 2560 with an internal flash memory of 256 KB. The Arduino MEGA 2560 board also has two UARTs. You can review this board on this site: <https://www.arduino.cc/en/Main/ArduinoBoardMega2560>. For Arduino MEGA 2560 from Arduino SRL, you can review it on <http://www.arduino.org/products/boards/arduino-mega-2560>. You can see the Arduino MEGA 2560 board in Figure 1-3.



Figure 1-3. *Arduino MEGA 2560*

Most Arduino boards don't have a built-in debug chip, so if you want to debug our program, you should add an additional debug tool. Fortunately, we can use the Arduino ZERO board (<https://www.arduino.cc/en/Main/ArduinoBoardZero>) from Arduino LLC and the Arduino M0 PRO board (<http://www.arduino.org/products/boards/arduino-m0-pro>) from Arduino SRL, which are supported for debugging without additional tools. These boards have Atmel's Embedded Debugger (EDBG) to be used for debugging. I suggest you to use these Arduino models if you have concerns about debugging without additional tools. A form of Arduino Zero board is shown in Figure 1-4.



Figure 1-4. *Arduino ZERO*

Arduino for Internet of Things

Today the Internet is a common term used to describe how to access data from a remote site. We can access the data from any device and anywhere. In the context of Arduino, it's very useful if our boards can be connected to the Internet. Imagine your boards sense the physical object and then send it to our smartphone. This happens if our boards are connected to Internet.

There are many options for Arduino shields for network modules to make our Arduino boards connect to the Internet. This means you should buy additional modules to make your boards connect to the Internet. Fortunately, various Arduino board models have a built-in network module in the board. We explore some Arduino board models with Internet connectivity capability.

Arduino UNO WiFi is an Arduino UNO board with an additional chip (ESP8266). This chip can be used to connect existing WiFi networks and also can work as an access point (AP) node. Arduino UNO WiFi is manufactured by Arduino SRL. You can see the Arduino UNO WiFi form in Figure 1-5. To obtain more information about Arduino UNO WiFi, you can review it on this site: <http://www.arduino.org/products/boards/arduino-uno-wifi>.

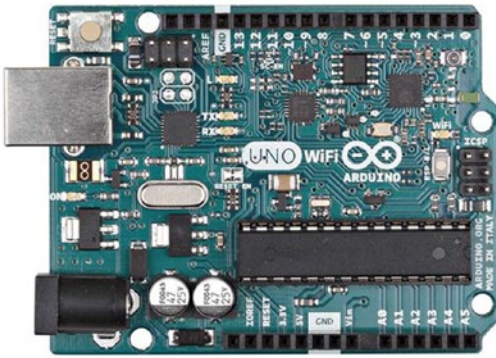


Figure 1-5. *Arduino UNO WiFi*

Arduino/Genuino MKR1000 is a development board with a ready-for-IoT scenario which is manufactured by Arduino LLC. The board runs with ATSAMW25 SoC, which consists of SAMD21 Cortex-M0+, WINC1500 WiFi module, and ECC508 CryptoAuthentication. It's designed for IoT connectivity, including accessing the cloud server. Further information about Arduino/Genuino MKR1000 can be read on <https://www.arduino.cc/en/Main/ArduinoMKR1000>. This board size, which is shown in Figure 1-6, is small.

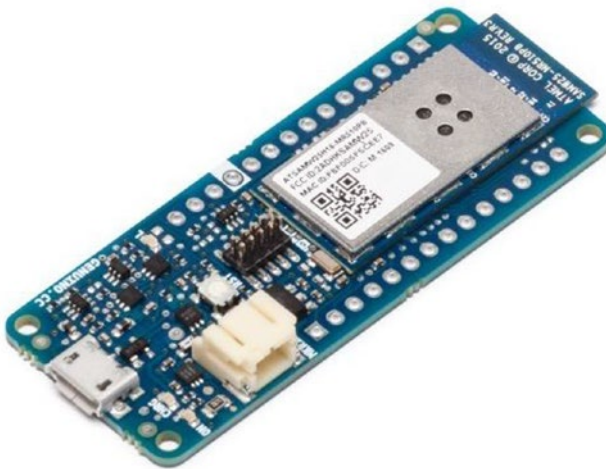


Figure 1-6. *Arduino/Genuino MKR1000*

Most Arduino boards work with an RTOS environment. Arduino YUN combines MCU and WiFi MCU, which runs OpenWrt Linux (called Linino OS). This board likes a mini-computer with Linux OS. You can control Arduino MCU ATmega32u4 from Linux. We also can access Linux API from the Arduino program. The Arduino YUN board has built-in WiFi and Ethernet modules to solve your IoT cases. Figure 1-7 is a form of Arduino YUN. You can review and buy this board on this site: <http://www.arduino.org/products/boards/arduino-yun>.

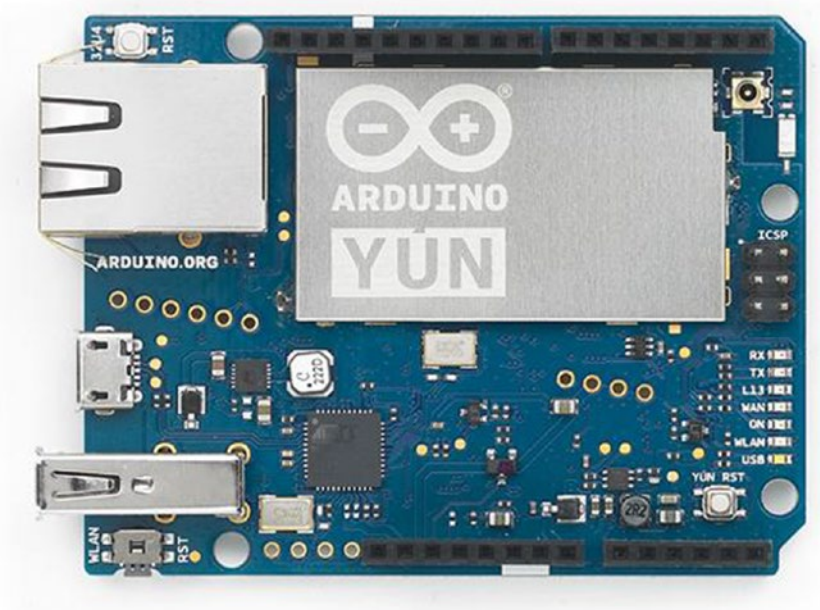


Figure 1-7. *Arduino YUN*

If you're looking for an Arduino with BLE connectivity capability, you can consider using the Arduino/Genuino 101 board, which is shown in Figure 1-8. This board uses Intel Curie as MCU, which has a built-in BLE module. You can control this board through Bluetooth on your smartphone, such as Android and iPhone smartphones. For further information about Arduino/Genuino 101, I recommend reading this website: <https://www.arduino.cc/en/Main/ArduinoBoard101>.



Figure 1-8. *Arduino/Genuino 101*

Arduino-Compatible

In general, the original Arduino board price is expensive. If you have pricing issues on your Arduino learning process, you could buy an Arduino-compatible board. As we know, Arduino shares its design and scheme under an open source project. It means we can build our own Arduino board without using the “Arduino” name for our board. Arduino-compatible boards usually are manufactured by third-party companies. This section will go through various Arduino-compatible boards in the market.

SparkFun RedBoard is a kind of Arduino-compatible board which is manufactured by SparkFun. This board has a form like the Arduino UNO board and uses ATmega328 as MCU with installed Optiboot (UNO) Bootloader. For development, we can use Arduino IDE to write a Sketch program and then flash the program to the board. If you are interested in SparkFun RedBoard, you can buy it on <https://www.sparkfun.com/products/12757>. You can see SparkFun RedBoard in Figure 1-9.

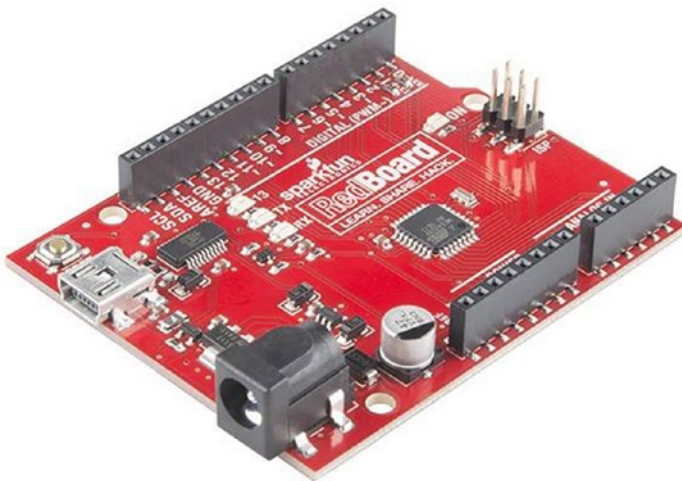


Figure 1-9. *SparkFun RedBoard*

Really Bare Bones Board (RBBB) is an Arduino-compatible board which is designed by Modern Device. This board doesn't provide a serial module so if you want to develop a Sketch program, you would need a serial tool such as FTDI cable. RBBB board is displayed in Figure 1-10. For further information about this board, you can review and buy it on <https://moderndevise.com/product/rbbb-kit/>.

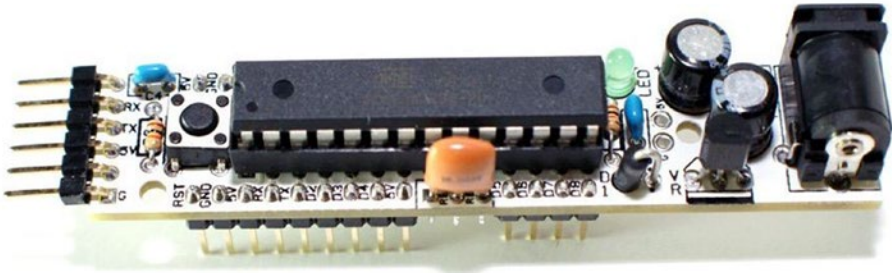


Figure 1-10. Really Bare Bones Board (RBBB)

Feather is an Arduino development brand from Adafruit. Adafruit Feather 32u4 Adalogger is one of the Feather board models. This board provides a MicroSD module for storing data. We can develop an Arduino program using Adafruit Feather 32u4 Adalogger. For further information, you can review and buy this board on <https://www.adafruit.com/products/2795>.

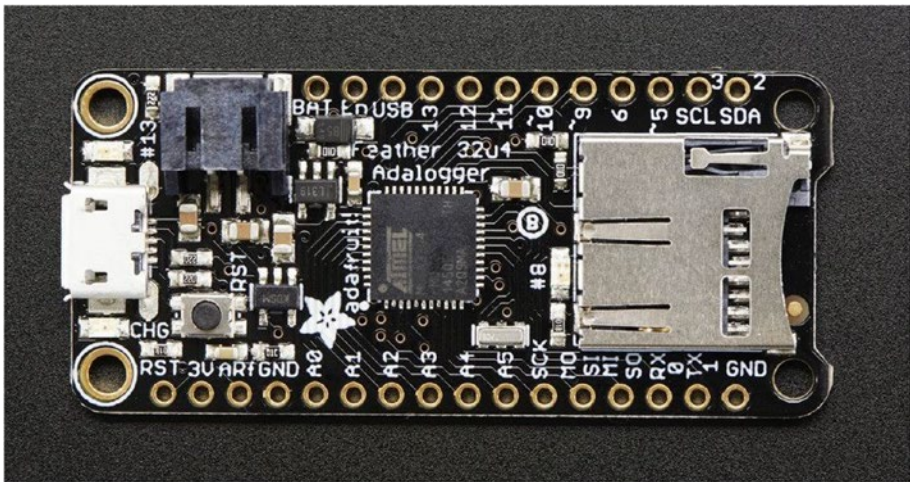


Figure 1-11. Adafruit Feather 32u4 Adalogger

In the next section, we will set up our development environment on a computer. We will use the official application for Arduino development.

Setting up Your Development Environment

The advantage of Arduino development is it's easier to set up a development environment because it supports multiple platforms such as Windows, Linux, and Mac. In this book, we focus on the Windows platform. We also explore .NET technology to access Arduino boards.

The first thing you should prepare is to install Arduino software. Depending on what a kind of Arduino module, if you have Arduino boards from Arduino LLC, you can download Arduino software on <https://www.arduino.cc/en/Main/Software>. Otherwise, Arduino boards from SRL can download Arduino software from this site: <http://www.arduino.org/downloads>.

After downloading and installing Arduino software, you can run the program. You can see the Arduino form from Arduino LLC in Figure 1-12.

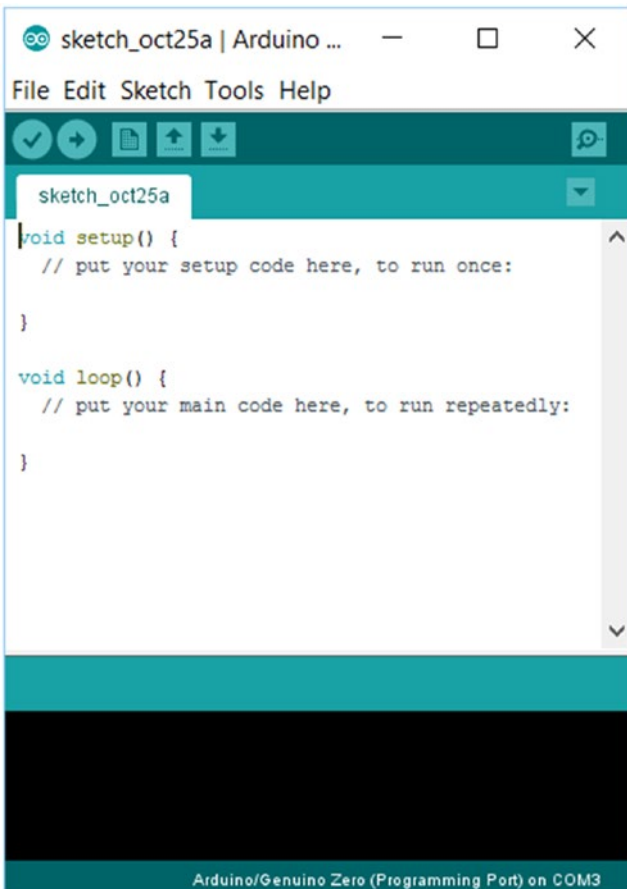


Figure 1-12. Arduino IDE

Now you can attach the Arduino board into your computer. If you have Arduino boards based on ATTiny and ATmega MCUs such as Arduino UNO, you don't need to install a driver. Arduino software has installed it for you.

For instance, I attach my Arduino UNO on Windows 10; I obtained that my Arduino UNO board is recognized as COM5, shown in Figure 1-13.

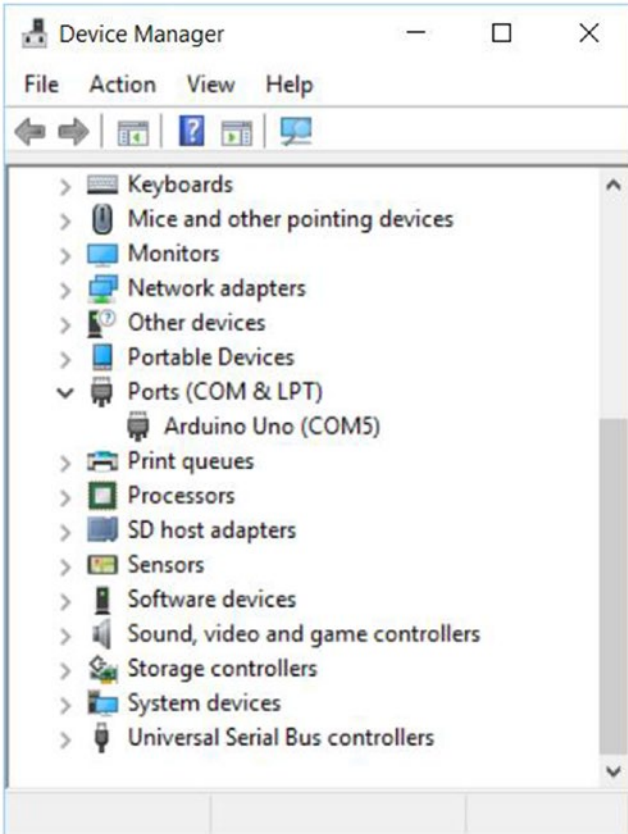


Figure 1-13. Arduino UNO is recognized in Device Manager on Windows 10

You also need Visual Studio to develop a .NET application, which accesses Arduino boards. If you don't have a Visual Studio license, you can download and install Visual Studio Express edition on this URL: <https://www.visualstudio.com/vs/visual-studio-express/>. For testing, I use Visual Studio 2015 with update 2. It is shown in Figure 1-14.

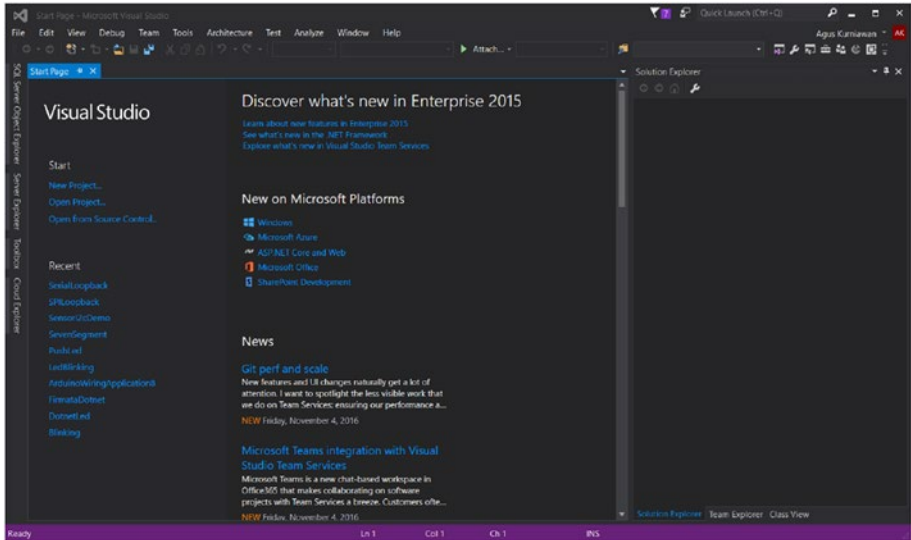


Figure 1-14. Visual Studio 2015

We learn how to develop a .NET application to control our Arduino boards in the next chapter. Make sure you have the Visual Studio tool. In the next section, we try to build a simple Arduino program using Arduino software. We use a built-in LED on an Arduino board.

Build your First Project: Blinking

When I obtain a new Arduino board, the first thing that I do is to build a simple app: blinking. This application runs to turn on/off an LED. Most Arduino boards have a built-in LED which is attached on digital pin 13.

In this section, we learn how to build an Arduino app to turn on/off LED. We can use a program sample from Arduino software. You can find it by clicking menu File ► Examples ► 01.Basic ► Blink. After clicking, you should see the program, shown in Figure 1-15.

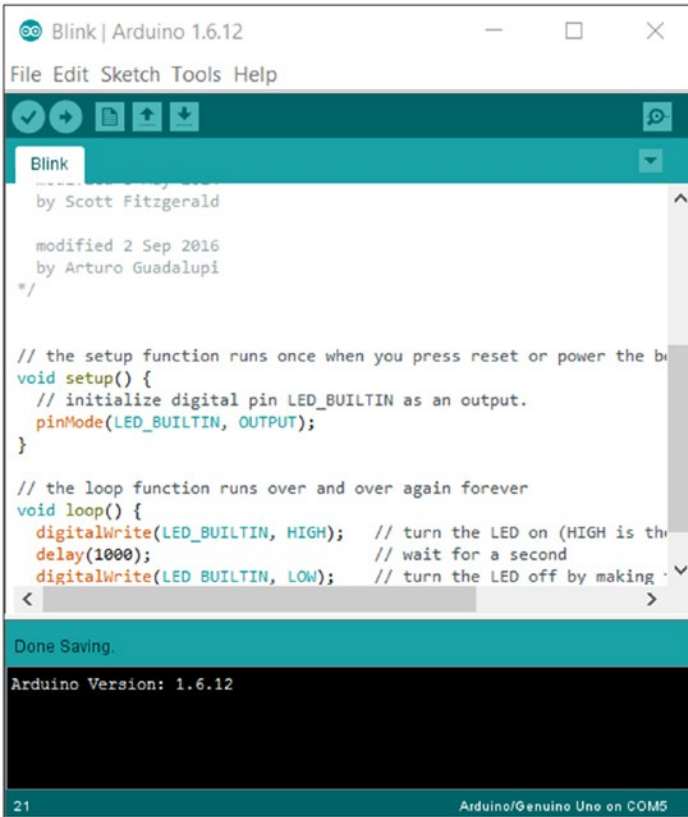


Figure 1-15. Blink program on Arduino software

In general, you should use the following code:

```
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);           // wait for a second
  digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
  delay(1000);           // wait for a second
}
```


You may see value 13 is replaced by LED_BUILTIN, which is shown in the following code:

```
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}
```

Save this program. To compile the program, you can click the Verify icon. Before uploading the program into an Arduino board, you should configure your IDE. Change the Arduino board target by clicking menu Tools ► Board (Figure 1-16). Then, you also need to set a serial port of the attached Arduino board by clicking menu Tools ► Port.

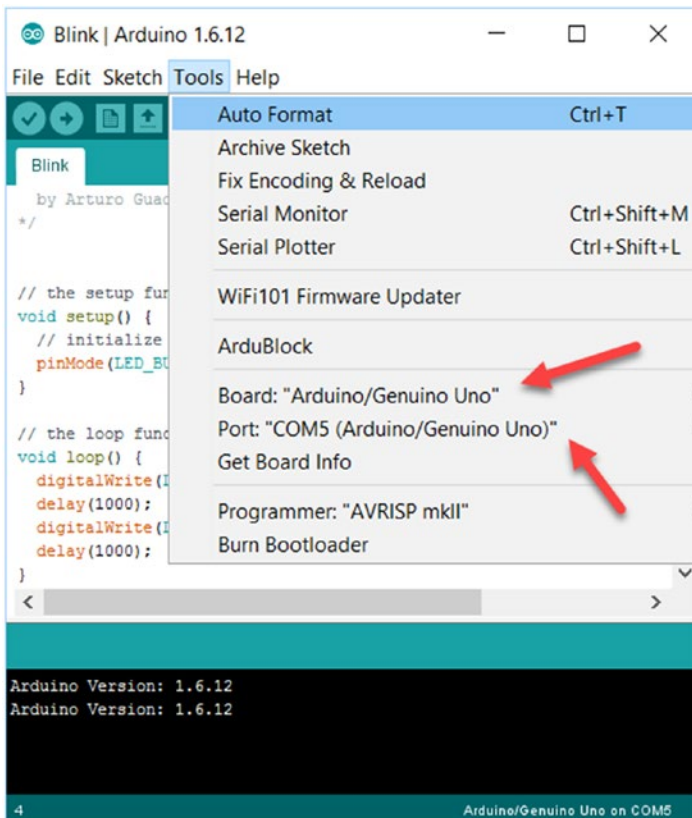


Figure 1-16. Configure board target and port

After it's configured, you can compile and upload the program. Click the Upload icon. You can see Verify and Upload icons in Figure 1-17.

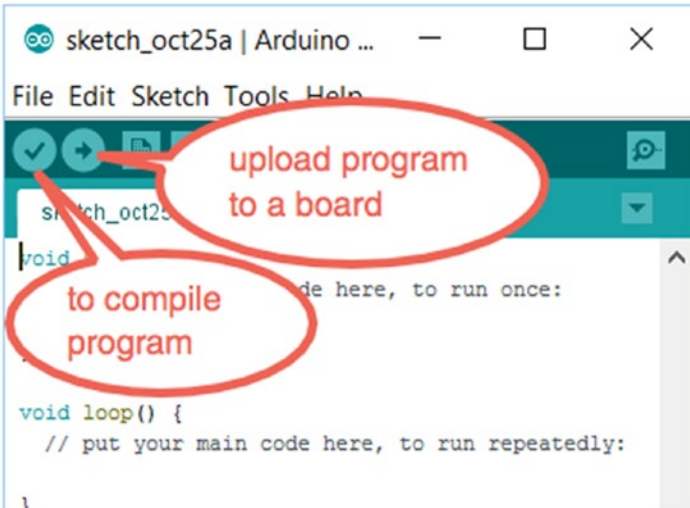


Figure 1-17. Compile and upload program on Arduino software

If you succeed, you should see a built-in LED is blinking. For instance, Arduino UNO has a built-in LED which is shown in Figure 1-18.

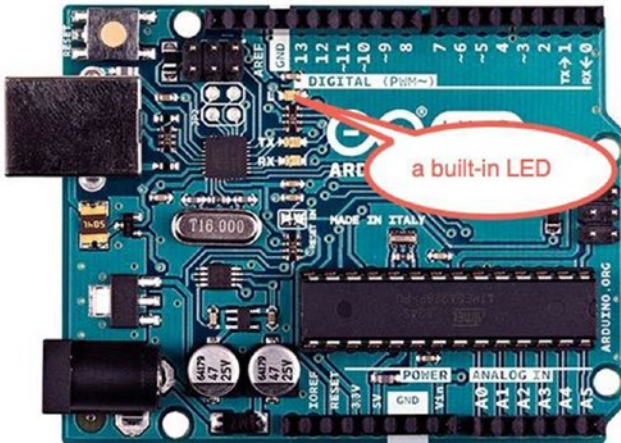


Figure 1-18. Blinking LED on Arduino UNO

Sketch Programming

To develop an Arduino app, you should know Sketch programming. In general, Sketch program uses C/C++ syntax. If you have experience in C/C++, you should be able to write an Arduino program easily.

The thing that you should know is to understand a sketch programming model. A sketch program uses two functions to run an Arduino program. The following is a sketch program in general.

```
setup() {
  // do something
}

loop(){
  // do something
}
```

On `setup()` function, the sketch program will execute once. However, `loop()` function is a function which is executed continually.

If you can build your own functions, then put them in either `setup()` function or `loop()` function. For instance, I created `foo()` and `perform()` functions and executed them on my Sketch program as follows.

```
setup() { // do something
  foo();}loop(){ // do something
  perform();}foo() {
}

perform() {
}
```

We also need to know some APIs in Sketch program. I recommend you read these on the official website. You can read it on <https://www.arduino.cc/en/Reference/HomePage> from Arduino LLC. Arduino SRL provides the reference website in <http://www.arduino.org/learning/reference>.

Arduino Programming Using Visual Studio

If you love Visual Studio as a development tool, you can use this tool to develop an Arduino program. We can use Visual Micro as an add-on tool for Visual Studio. You can download it on <http://www.visualmicro.com>. After it's installed, you should see new project templates, which are shown in Figure 1-19. To work with Visual Micro, you still need to install Arduino software.

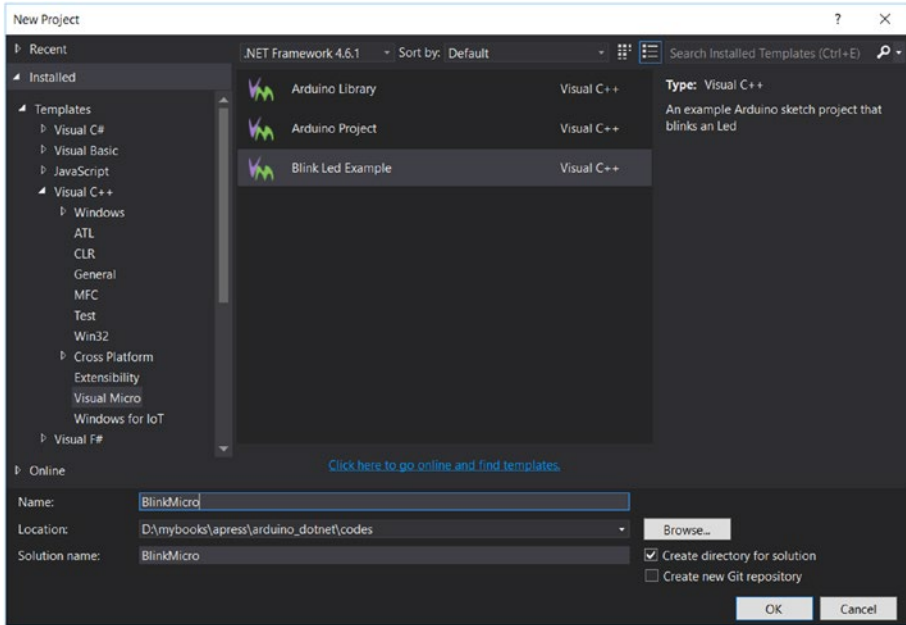


Figure 1-19. Arduino project template on Visual Studio 2015

Now you can bring your Sketch program into Visual Studio. You can write Sketch program into Visual Studio. For instance, we can write Sketch program by selecting Blink Led Example from Visual C++ > Visual Micro.

After it's selected, we can see a blinking Sketch program in Visual Studio editor (see Figure 1-20). You should find xxx.ino file where xxx is a Sketch file.

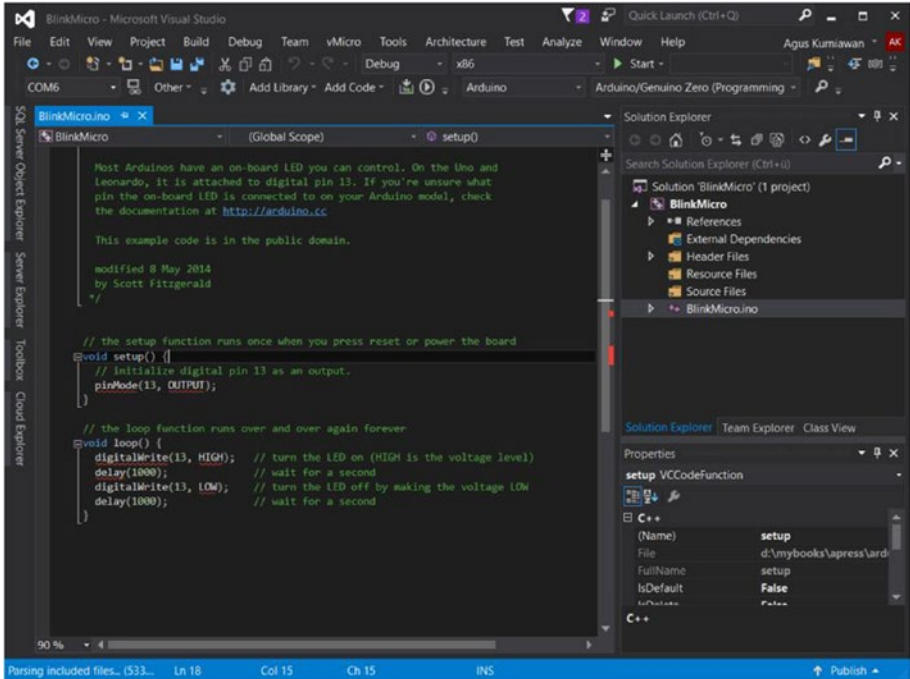


Figure 1-20. Sketch program in Visual Studio 2015 editor

Before you build and upload sketch, you should configure an Arduino board target. You should see a list of Arduino board model. You can see it in Figure 1-21. Select your Arduino board model from the list.

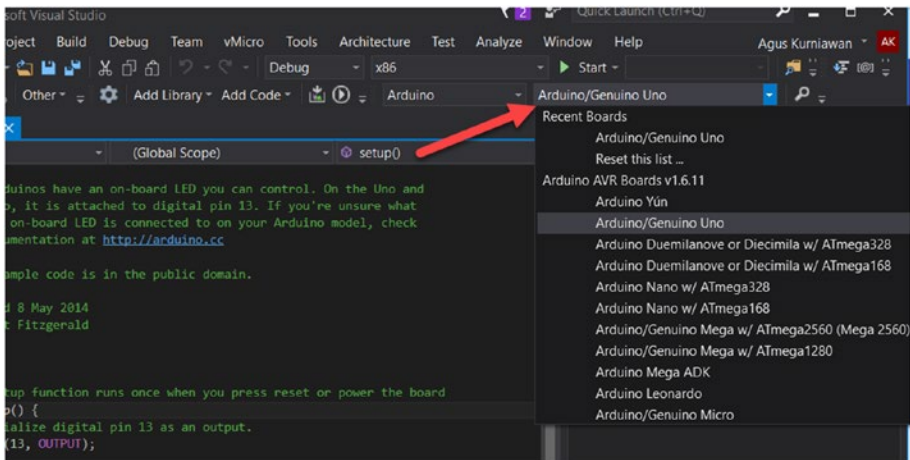


Figure 1-21. Select Arduino board model for board target

You also need to configure a serial port for Arduino. You can see its configuration on a menu which is shown in Figure 1-22. Build and upload menus also can be found on that toolbar.

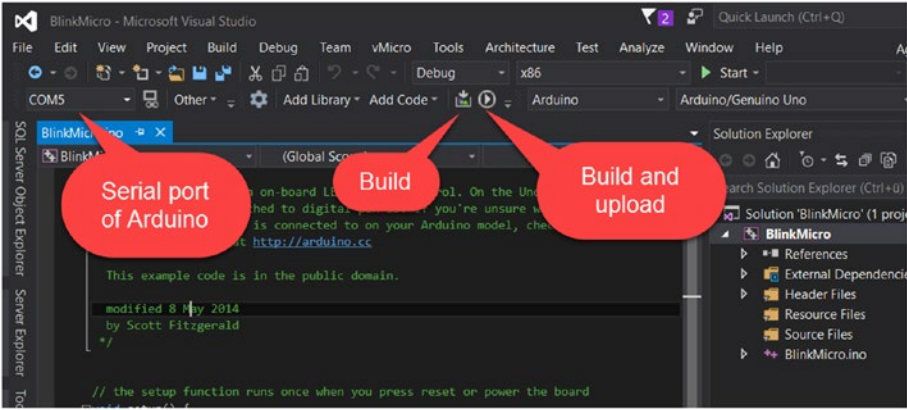


Figure 1-22. Menu for selecting serial port, building, and uploading

Save your sketch. Then, try to build and upload the program. The program will run on an Arduino board. Visual Studio debugger also runs. You can see it in Figure 1-23.

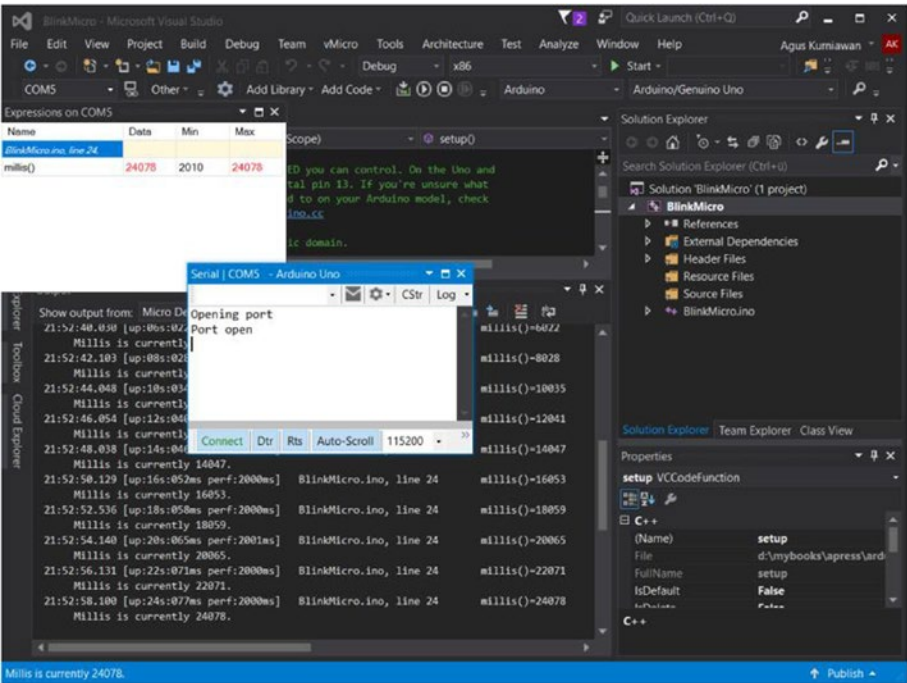


Figure 1-23. Running and debugging Arduino sketch on Visual Studio

Summary

We have explored various Arduino models and tried to build a simple Arduino program using Arduino software. We also built an Arduino program using Visual Studio and then deployed it into an Arduino board.

You can now practice developing Arduino programs with sketch. In the next chapter, we build an interface program to communicate between a .NET application and Arduino boards.

CHAPTER 2



Interfacing .NET and Arduino

.NET is a complete technology to solve your business cases, including hardware programming. In this chapter, we will learn how to control Arduino boards from a .NET application. Several Arduino I/O communication models are introduced, so our .NET application can control Arduino boards. The last topic is to learn Firmata protocol, one of the protocols to communicate between an Arduino board and a computer.

This chapter covers the following topics:

- Arduino I/O Communication
- Controlling Arduino from .NET
- Introduction to Firmata protocol

Arduino I/O Communication

Arduino boards usually provide I/O pins so we attach our sensor or actuator devices into the board. Each Arduino board has an I/O layout so you should check it. For instance, Arduino UNO can be described for I/O layout, as shown in Figure 2-1. This layout is taken from the Pighixx website, <http://www.pighixx.com/test/portfolio-items/uno/>.

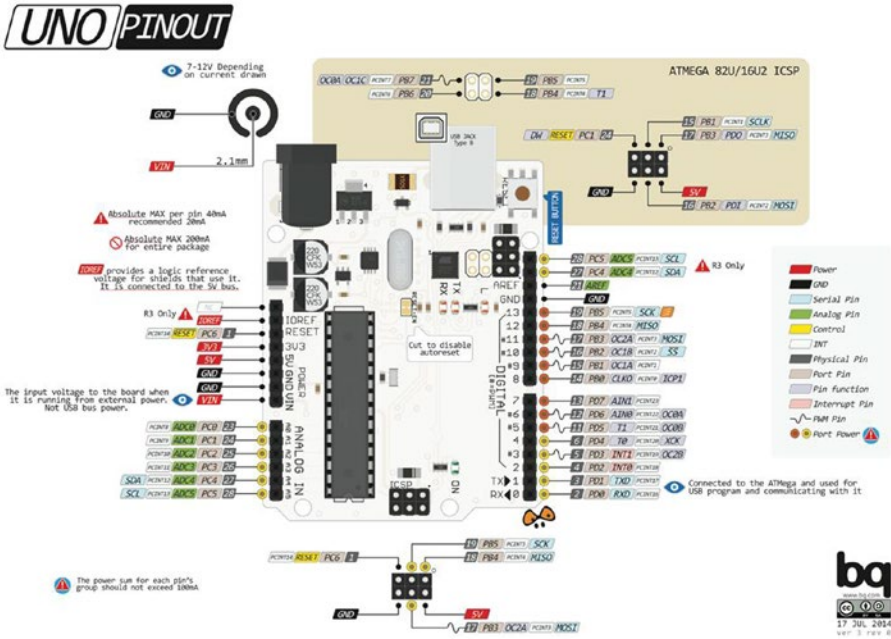


Figure 2-1. Arduino UNO I/O layout

To access an external resource from Arduino I/O, we should deal with protocol format. In general, Arduino boards have the following three communication models:

- serial communication
- SPI communication
- TWI/I2C communication

We use these communication models to communicate with a .NET application. Each communication will be explained on the next section. Sketch will be used to describe how these I/O communication work.

Let's explore!

Serial Communication - UART

Serial communication—sometimes it's called UART (Universal Asynchronous Receiver Transmitter)—is an old communication model. In the serial communication method, we send one bit at time, sequentially, over a communication channel. If you have 32 bits of data, the data will send one by one bit. A speed of data sending is depending on a serial speed, called baudrate.

Most Arduino boards have serial communication capability via UART pins. For instance, Arduino UNO has UART pins on digital pin 0 and 1.

In the Sketch program, we can use the Serial library (<https://www.arduino.cc/en/Reference/Serial>) to communicate UART pins on Arduino boards. `Serial.print()` and `Serial.read()` functions are used to write and read data on Arduino UART.

For testing, we build a Sketch to write data on Arduino UART. I used Arduino UNO. This program will write data to serial port on Arduino. For testing, we write sequential numbers into a serial port. Then, we listen to incoming messages from serial port using Serial Monitor, a tool from Arduino software.

Now you can open Arduino IDE and write the following code.

```
int led = 13;
int val = 10;
void setup() {
  pinMode(led, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  digitalWrite(led,HIGH);
  Serial.print("val=");
  Serial.println(val);
  delay(1000);

  digitalWrite(led,LOW);
  val++;
  if(val>50)
    val = 10;
}
```

I'll explain how this sketch works after this next step. Save this program as "SerialDemo."

Compile and upload sketch into an Arduino board. To see program output on Serial I/O, we can use a Serial Monitor. This tool is available from Arduino software. Just click menu Tools ► Serial Monitor. See Figure 2-2.

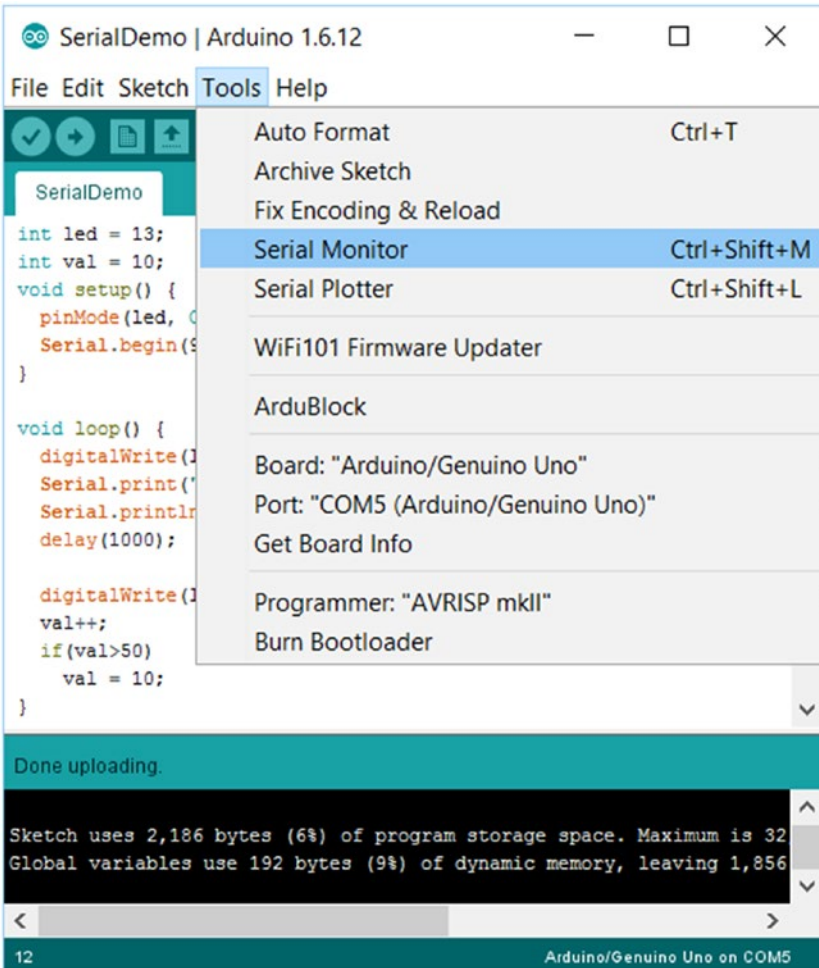


Figure 2-2. Open Serial Monitor tool on Arduino IDE

After opening the Serial Monitor tool, you should see a dialog, shown in Figure 2-3. Change baudrate to 9600 because our program uses baudrate 9600 to communicate with Arduino serial.

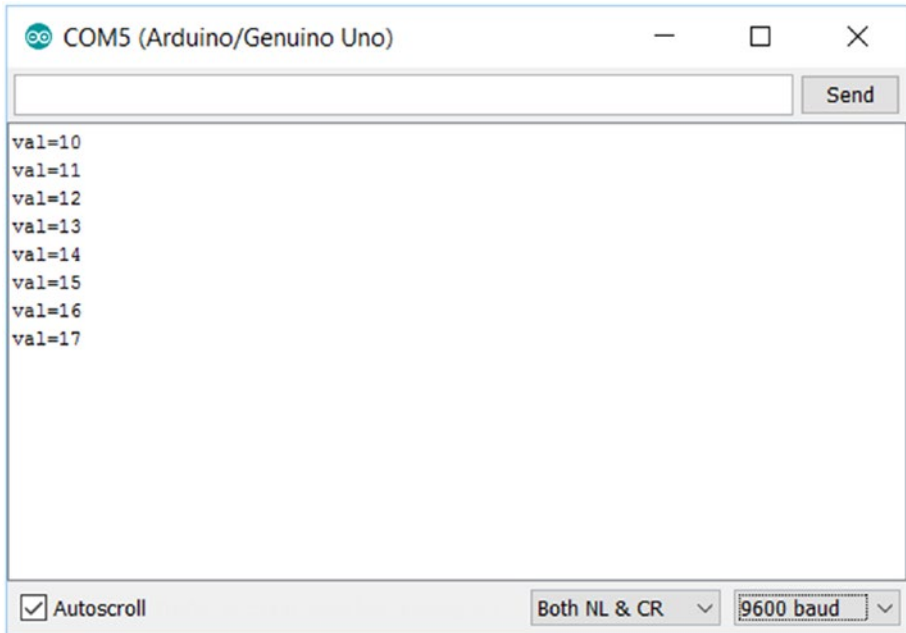


Figure 2-3. Program output on Serial Monitor tool

You should see “value=xx.” xx is a sequential number which is generated from our Sketch program.

How the Program Works

Now I’ll explain how the previous program works. First, we initialize our serial port with baudrate 9600. We also use a built-in LED on pin 13. This happens on `setup()` function.

```
int led = 13;
int val = 10;
void setup() {
  pinMode(led, OUTPUT);
  Serial.begin(9600);
}
```

Then, we send the message “value = xx” to serial output. xx is a sequential number starting from 10 to 50. We perform it on `loop()` function. To write data to serial output, we call `Serial.print()` and `Serial.println()` functions.

```
void loop() {
  digitalWrite(led,HIGH);
  Serial.print("val=");
  Serial.println(val);
  delay(1000);
}
```

```
digitalWrite(led,LOW);
val++;
if(val>50)
    val = 10;
}
```

This is a simple program. However, you can do more practice using Serial library from Sketch program. For instance, you send sensor data such as temperature and humidity to serial port.

SPI Communication

Since UART does not use a clock to control transmitter and receiver, messages may get problems in communication. We get a problem in UART when we want to attach more devices into UART bus. SPI protocol is designed to work with several devices in serial bus.

Serial Peripheral Interface (SPI) uses master and slave clock to determine which device will communicate. SPI usually is implemented in three pins: SCK for clock, MOSI for Master Out Slave In, and MISO for Master In Slave Out. SS (Slave Select) signals are used for the master to control many slave devices. The Arduino UNO board, for instance, has SPI pins on 10 (SS), 11 (MOSI), 12 (MISO), and 13 (SCK).

There are many options for sensor and actuator devices which use SPI protocol. For instance, SparkFun Triple Axis Accelerometer Breakout from SparkFun (<https://www.sparkfun.com/products/11446>). This breakout uses IC ADXL362 to manage the accelerometer sensor. You can see this breakout in Figure 2-4.

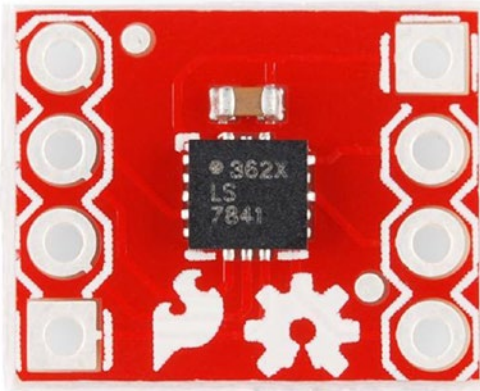


Figure 2-4. SparkFun Triple Axis Accelerometer Breakout from SparkFun

To test our Arduino with SPI protocol, we can connect MOSI and MISO pins using a jumper cable. We connect digital pin 11 to digital pin 12, as shown in Figure 2-5.

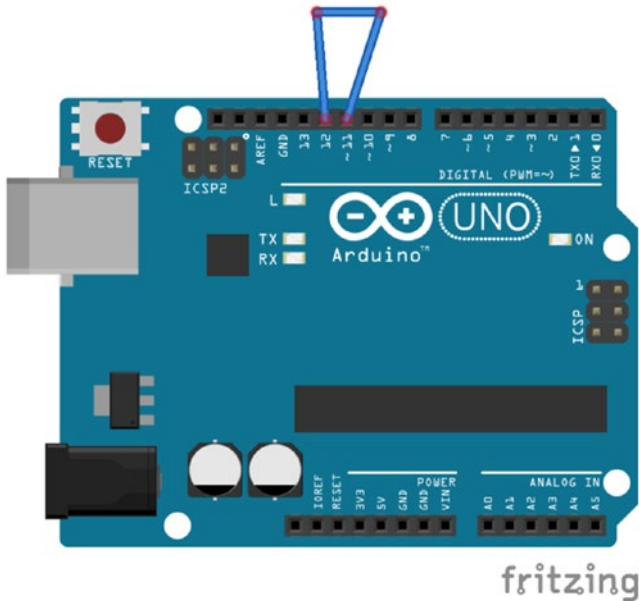


Figure 2-5. Connecting MISO and MOSI pins

Now we start to build the Arduino program. In this case, we send data over SPI and wait for incoming data from SPI. This program is an SPI loopback in which input and output SPI pins are connected. We can use the SPI library (<https://www.arduino.cc/en/Reference/SPI>) to access the SPI protocol. In the SPI library, we can use `SPI.transfer()` function to send and receive data. The program will send random numbers to SPI. Open Arduino IDE and write the following code.

```
#include <SPI.h>

byte sendData,recvData;
void setup() {
  SPI.begin();
  Serial.begin(9600);
}

// source:
// http://forum.arduino.cc/index.php?topic=197633.0
byte randomDigit() {
  unsigned long t = micros();
  byte r = (t % 10) + 1;
  for (byte i = 1; i <= 4; i++) {
    t /= 10;
    r *= ((t % 10) + 1);
  }
}
```

```

    r %= 11;
  }
  return (r - 1);
}

void loop() {
  sendData = randomDigit();
  recvData = SPI.transfer(sendData);

  Serial.print("Send=");
  Serial.println(sendData,DEC);
  Serial.print("Recv=");
  Serial.println(recvData,DEC);
  delay(800);
}

```

Save this sketch as “SPIDemo.”

Now you can compile and upload the Sketch program into Arduino UNO. Open a Serial Monitor tool to see the program output. Don't forget to change baudrate to 9600. A sample of the program can be seen in Figure 2-6.

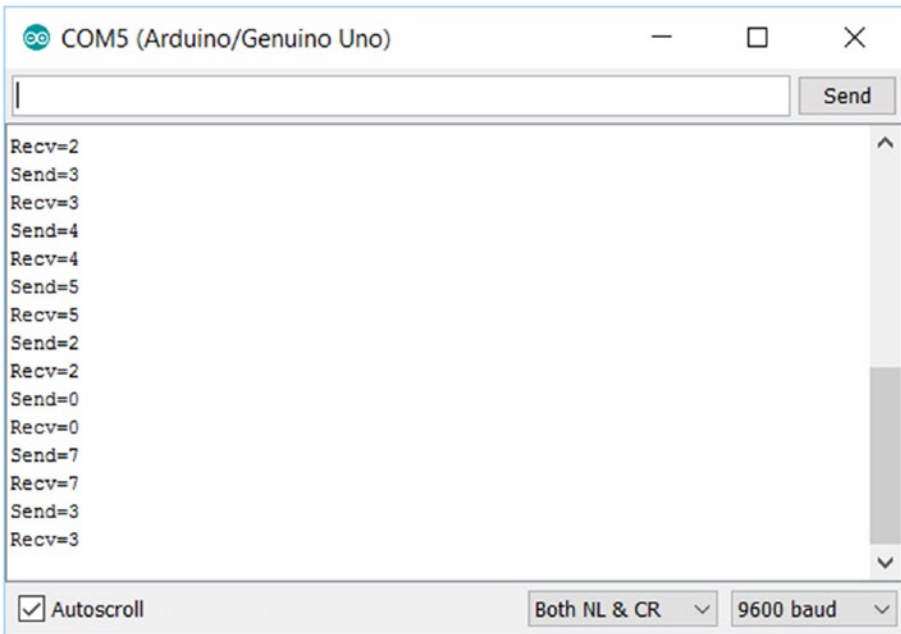


Figure 2-6. Program output for SPI Sketch program

How the Program Works

To access the SPI library, we call `SPI.begin()` function into `setup()` function. To work with the SPI library, we need to include the `SPI.h` header file in your program. In general, this header file is already installed in Arduino software.

```
#include <SPI.h>

byte sendData,recvData;
void setup() {
    SPI.begin();
    Serial.begin(9600);
}

```

In `loop()` function, we perform to send a random number via SPI. This random number is generated by calling `randomDigit()` function. We can use the `SPI.transfer()` function to send and retrieve data on the SPI bus. Finally, we display data which is sent and received on the serial port so we can monitor it via Serial Monitor tool.

```
void loop() {
    sendData = randomDigit();
    recvData = SPI.transfer(sendData);

    Serial.print("Send=");
    Serial.println(sendData,DEC);
    Serial.print("Recv=");
    Serial.println(recvData,DEC);
    delay(800);
}

```

You have learned how to access SPI by implementing an SPI loopback scenario. If you want to get more advanced practice, I recommend you use sensor or actuator devices based on SPI protocol. Please remember that each sensor/actuator device has specific commands over SPI to interact with Arduino boards. You should check them on their datasheet documents.

TWI/I2C Communication

As we know, SPI protocol needs four pins to communicate among devices. This is one drawback in SPI protocol. SPI only allows one master on the bus with multiple slave devices. On the other hand, I2C (Inter-Integrated Circuit)/TWI (Two-Wire Interface) is also a synchronous protocol which needs two pins to communicate among devices. I2C protocol has multi-master devices which talk to multi-slave devices.

I2C protocols have two wires: SDA and SCL. Arduino boards usually have an I2C bus. For instance, Arduino UNO has I2C on pins A4 (SDA) and A5 (SCL). Please check your Arduino model to see the I2C pinout. We can access I2C using the Wire library (<https://www.arduino.cc/en/Reference/Wire>).

There are many devices which use I2C to be attached to Arduino boards. One I2C sensor device is SparkFun Humidity and Temperature Sensor Breakout from SparkFun (<https://www.sparkfun.com/products/13763>). This breakout uses IC Si7021 to sense humidity and temperature. The sensor data can be accessed using I2C protocol. You can see this breakout in Figure 2-7.

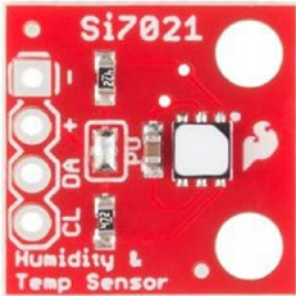


Figure 2-7. SparkFun Humidity and Temperature Sensor Breakout from SparkFun

Another module is the PCF8591 AD/DA module. It's not an expensive module. You can find it on <http://www.electrodragon.com/product/pcf8591-adc-dac-adda-analog-digital-converter-module/>. Ebay and Chinese websites also sell this module. The advantage of this module is that the module provides three sample inputs: Thermistor, Photocell, and Potentiometer. Figure 2-8 is a form of the PCF8591 AD/DA module. In this case, I use it for I2C testing on an Arduino UNO board.

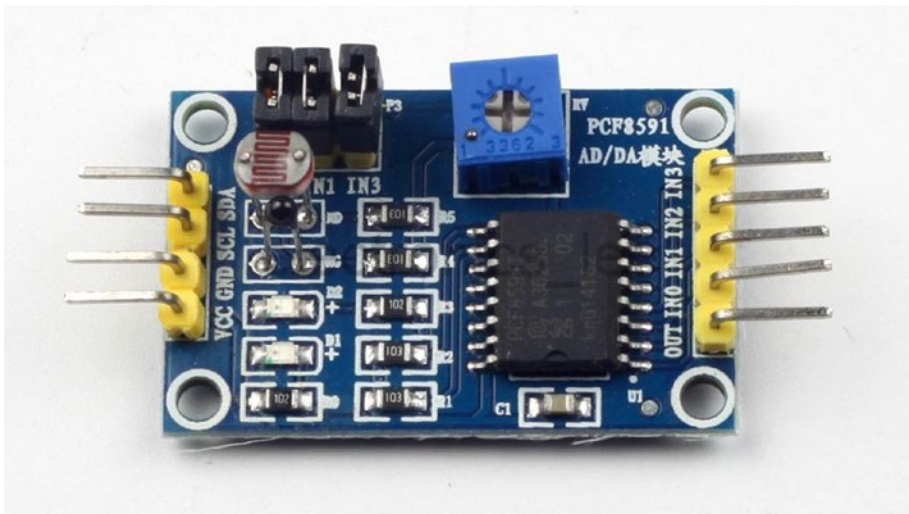


Figure 2-8. PCF8591 AD/DA module

For an I2C demo on an Arduino board, we use a PCF8591 AD/DA module. We will access Thermistor, Photocell, and Potentiometer through I2C protocol using Sketch program.

For wiring implementation, you can connect the PCF8591 AD/DA module to an Arduino UNO board as follows:

- Connect SDA pin to Arduino SDA pin (A4).
- Connect SCL pin to Arduino SCL pin (A5).
- Connect GND pin to Arduino GND.
- Connect VCC pin to Arduino VCC 3.3V.

You can see my wiring in Figure 2-9.

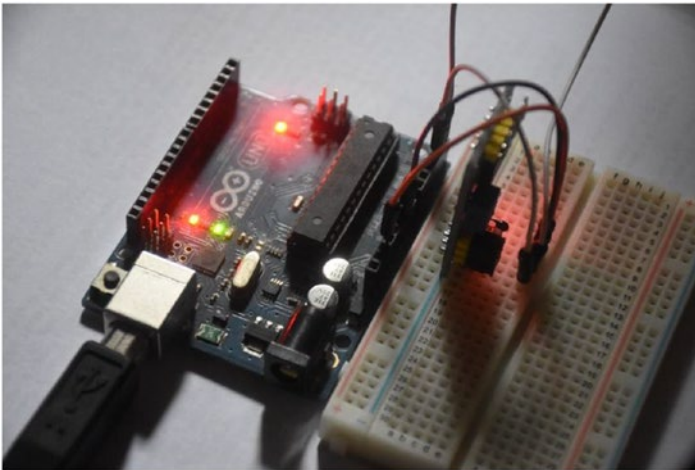


Figure 2-9. Wiring PCF8591 AD/DA module and Arduino UNO

We can use the Wire library to access the I2C bus. The PCF8591 AD/DA module runs on address (0x90 >> 1) or 0x48. We pass this address to the Wire library. Based on a document from the PCF8591 AD/DA module, Thermistor runs on channel 0x00, Photocell on 0x01, and Potentiometer on 0x03.

Now open Arduino IDE and write this code:

```
#include "Wire.h"
#define PCF8591 (0x90 >> 1) // I2C bus address
#define PCF8591_ADC_CH0 0x00 // thermistor
#define PCF8591_ADC_CH1 0x01 // photo-voltaic cell
#define PCF8591_ADC_CH2 0x02
#define PCF8591_ADC_CH3 0x03 // potentiometer
byte ADC1, ADC2, ADC3;
```

```

void setup()
{
  Wire.begin();
  Serial.begin(9600);
}
void loop()
{
  // read thermistor
  Wire.beginTransmission(PCF8591);
  Wire.write(PCF8591_ADC_CH0);
  Wire.endTransmission();
  Wire.requestFrom(PCF8591, 2);
  ADC1=Wire.read();
  ADC1=Wire.read();

  Serial.print("Thermistor=");
  Serial.println(ADC1);

  // read photo-voltaic cell
  Wire.beginTransmission(PCF8591);
  Wire.write(PCF8591_ADC_CH1);
  Wire.endTransmission();
  Wire.requestFrom(PCF8591, 2);
  ADC2=Wire.read();
  ADC2=Wire.read();

  Serial.print("Photo-voltaic cell=");
  Serial.println(ADC2);

  // potentiometer
  Wire.beginTransmission(PCF8591);
  Wire.write(PCF8591_ADC_CH3);
  Wire.endTransmission();
  Wire.requestFrom(PCF8591, 2);
  ADC3=Wire.read();
  ADC3=Wire.read();

  Serial.print("potentiometer=");
  Serial.println(ADC3);

  delay(500);
}

```

Save this sketch as “I2CSensor.”

Compile and upload the program into the Arduino board. You can see the output using the Serial Monitor tool. A sample output can be seen in [Figure 2-10](#).

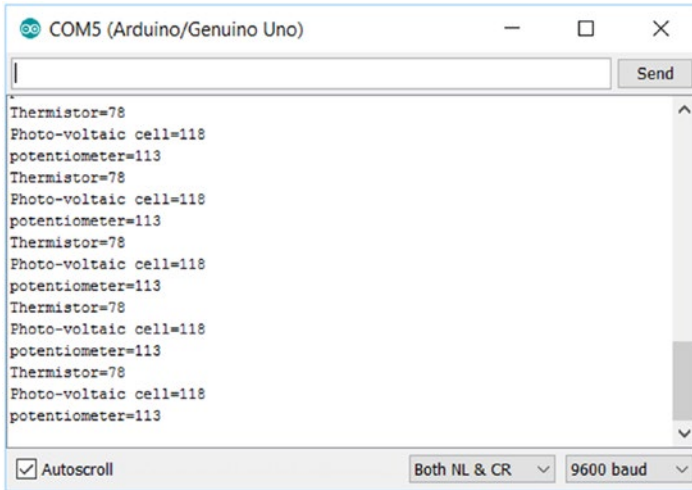


Figure 2-10. Program output for I2C Sketch program

How the Program Works

The program requires a Wire library, so you should include a Wire.h header file in our Sketch program. We also declare I2C addresses and commands for the PCF8591 AD/DA module.

```
#include "Wire.h"
#define PCF8591 (0x90 >> 1) // I2C bus address
#define PCF8591_ADC_CH0 0x00 // thermistor
#define PCF8591_ADC_CH1 0x01 // photo-voltaic cell
#define PCF8591_ADC_CH2 0x02
#define PCF8591_ADC_CH3 0x03 // potentiometer
byte ADC1, ADC2, ADC3;
```

We activate our Wire library on `setup()` function by calling `Wire.begin()` function. We also need to activate the Serial library in order to send a result of I2C reading by calling `Serial.begin()` function.

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

To read Thermistor data from the PCF8591 AD/DA module, you send channel mode, 0x00, into PCF8591 AD/DA I2C address. Then, you should read two bytes to obtain Thermistor data.

```

Wire.beginTransmission(PCF8591);
Wire.write(PCF8591_ADC_CH0);
Wire.endTransmission();
Wire.requestFrom(PCF8591, 2);
ADC1=Wire.read();
ADC1=Wire.read();

Serial.print("Thermistor=");
Serial.println(ADC1);

```

We perform the same task to read Photocell, and Potentiometer from PCF8591 AD/DA I2C. You just pass PCF8591_ADC_CH1 and PCF8591_ADC_CH3 values to the module. Then, wait for incoming two bytes from PCF8591 AD/DA.

Now that we have learned how to work with Arduino I/O communication, let's continue to control the Arduino board from our .NET application.

Control Arduino Board from .NET

We have learned several Arduino I/O communications using Sketch programming. Now we continue to access Arduino I/O from a .NET application.

In this section, I show you a simple communication between a .NET application and Arduino. I use serial communication. As we know, an Arduino board can be connected to a computer via Serial USB. We utilize this feature in our program. The idea is to send a command to Arduino to do something through serial communication protocol.

For testing, we build a .NET application to control three LEDs. We send commands by sending the message "1" to turn on LED 1, "2" to turn on LED 2, and "3" to turn on LED 3 via serial communication. In .NET, we can use a SerialPort object to access Serial I/O or UART. First, attach three LEDs into the Arduino board. The following is our wiring:

- Connect LED 1 to digital pin 12.
- Connect LED 2 to digital pin 11.
- Connect LED 3 to digital pin 10.
- Connect all other LED pins (GND pin) to GND pin.

You can see this wiring in [Figure 2-11](#).

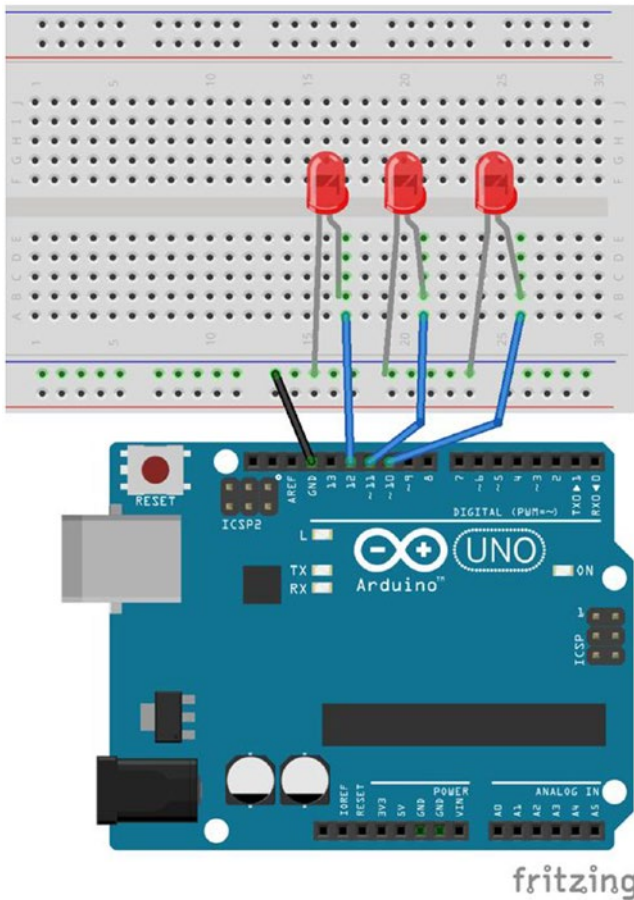


Figure 2-11. Wiring for LED control program

In order for our Arduino to understand our commands from a .NET application, we should use a Sketch program to receive commands from a .NET application. Write the program into Arduino IDE as follows:

```
int led1 = 12;
int led2 = 11;
int led3 = 10;

void setup() {
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  Serial.begin(9600);
}
```

```

void loop() {

    if (Serial.available() > 0) {

        char inputData = Serial.read();
        if(inputData=='1'){
            digitalWrite(led1, HIGH);
            digitalWrite(led2, LOW);
            digitalWrite(led3, LOW);
        }
        if(inputData=='2'){
            digitalWrite(led1, LOW);
            digitalWrite(led2, HIGH);
            digitalWrite(led3, LOW);
        }
        if(inputData=='3'){
            digitalWrite(led1, LOW);
            digitalWrite(led2, LOW);
            digitalWrite(led3, HIGH);
        }

    }

}

```

This program will listen incoming messages from serial port on `loop()` function. If we receive data “1”, we turn on LED 1. Otherwise, if we obtain data “2” or “3” on the serial port of Arduino, we turn on LED 2 or LED 3.

Save this program as “ledcontrol”. After that, you can compile and upload the Sketch program into the Arduino board.

The next step is to write a .NET application using Visual Studio. Create a Console Application project from Visual Studio. On the `Program.cs` file, we write a program to access the Serial port and send the message “1”, “2” and “3”. These inputs come from the user. Using `Console.ReadLine()`, we can get input data from the Terminal. Then, send it to the Serial port. Write this program:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO.Ports;

namespace DotnetBlinking
{
    class Program
    {
        static bool running = true;

```

```

static void Main(string[] args)
{
    try
    {
        SerialPort serialPort = new SerialPort("COM5");
        serialPort.BaudRate = 9600;

        Console.CancelKeyPress += delegate (object sender,
        ConsoleCancelEventArgs e)
        {
            e.Cancel = true;
            running = false;
        };

        serialPort.Open();
        Console.WriteLine("Press 1, 2, 3 to turn on LED 1, 2, 3 and
        then press ENTER");
        Console.WriteLine("Press CTRL+C to exit");
        while (running)
        {
            Console.Write(">> ");
            string data = Console.ReadLine();
            if(!string.IsNullOrEmpty(data))
            {
                serialPort.Write(data);
                Console.WriteLine("Sent to Arduino: {0}", data);
            }
        }
        serialPort.Close();
    }
    catch (Exception err)
    {
        Console.WriteLine(err);
    }
    Console.WriteLine("Program exit. Press ENTER to close.");
    Console.ReadLine();
}
}
}

```

Save this program. Run this program from Visual Studio. Please change your Arduino port before running the program. You should get a Terminal. Try to fill "1" to turn on LED 1 or "2" to turn on LED 2. A sample of program output can be seen in Figure 2-12. If you get a problem due to security, you can probably run this program at Administrator level.


```

file:///D:/mybooks/apress/arduino_dotnet/codes/DotnetArduino/DotnetBlinki...
Press 1, 2, 3 to turn on LED 1, 2, 3 and then press ENTER
Press CTRL+C to exit
>> 2
Sent to Arduino: 2
>> 3
Sent to Arduino: 3
>> 1
Sent to Arduino: 1
>> 3
Sent to Arduino: 3
>> 1
Sent to Arduino: 1
>> 4
Sent to Arduino: 4
>>

```

Figure 2-12. Program output for LED control app

How the Program Works

How does this .NET application work? The idea of the application is a simple approach. We open a serial port of Arduino. Then, the application is waiting for input data from the user.

```

Console.CancelKeyPress += delegate (object sender, ConsoleCancelEventArgs e)
{
    e.Cancel = true;
    running = false;
};
serialPort.Open();

```

Once the data is received, the application passes it to a serial port using `serialPort.Write()` method. We implement the infinity loop using `while()` syntax. This looping will be stopped if the user presses CTRL+C keys, which our program is listening to via a `Console.CancelKeyPress` event.

```

while (running)
{
    Console.Write(">> ");
    string data = Console.ReadLine();
    if(!string.IsNullOrEmpty(data))
    {
        serialPort.Write(data);
        Console.WriteLine("Sent to Arduino: {0}", data);
    }
}

```

That is how our application works. You can probably make experiments by applying Windows Forms. Please remember locked access, since our .NET application tries to perform blocking on waiting for input from the user.

Introducing Firmata Protocol

Firmata is a protocol for communicating with microcontrollers from software on a computer. Basically, we have implemented the same approach on the previous section. However, Firmata protocol provides a more general model to communicate between the Arduino board and the computer. For further information about Firmata protocol, I suggest you to read this website: <https://github.com/firmata/protocol>.

In order for Arduino to understand our commands from the computer, we should run the Firmata program in the Arduino boards. You can find this program in Arduino IDE, by clicking menu File ► Examples ► Firmata ► StandardFirmata.

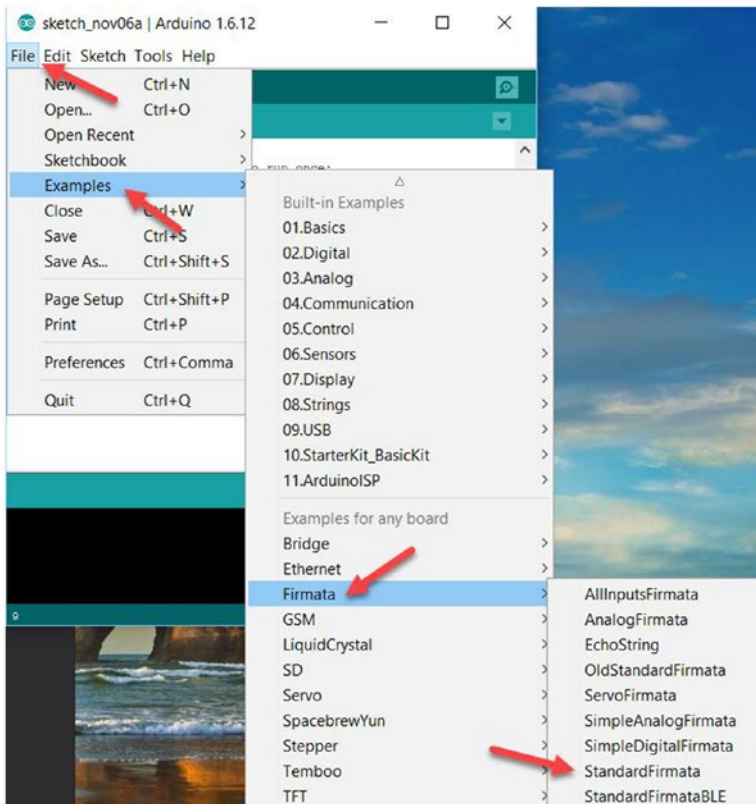


Figure 2-13. Adding Firmata Sketch program

After it's clicked, you can see the Sketch program for Firmata in Arduino IDE. You can see it in Figure 2-14. You can configure the Arduino target and its port before flashing to the Arduino board. If flashing sketch is done, your Arduino board is ready for receiving commands via Firmata protocol.

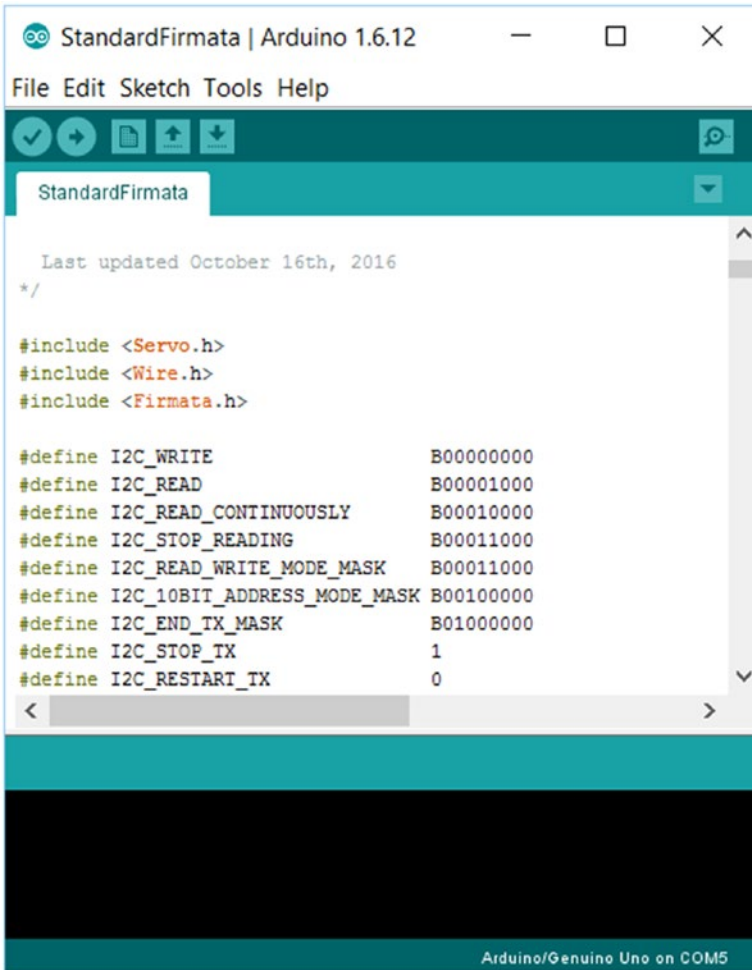


Figure 2-14. Sketch program for standard Firmata

To access Arduino via Firmata protocol, we should send commands based on Firmata protocol. However, we can use several Firmata client libraries, which community people created. You can check them on <https://github.com/firmata/arduino>. Since we use .NET to communicate with an Arduino board, we use a Firmata client for a .NET platform.

I use SolidSoils4Arduino for Firmata client in .NET. This library is supported for Standard Firmata 2.4. You download this library on <https://github.com/SolidSoils/Arduino>. Then, open it using Visual Studio and compile the project with your current .NET framework version. A project list of SolidSoils4Arduino can be seen in Figure 2-15.

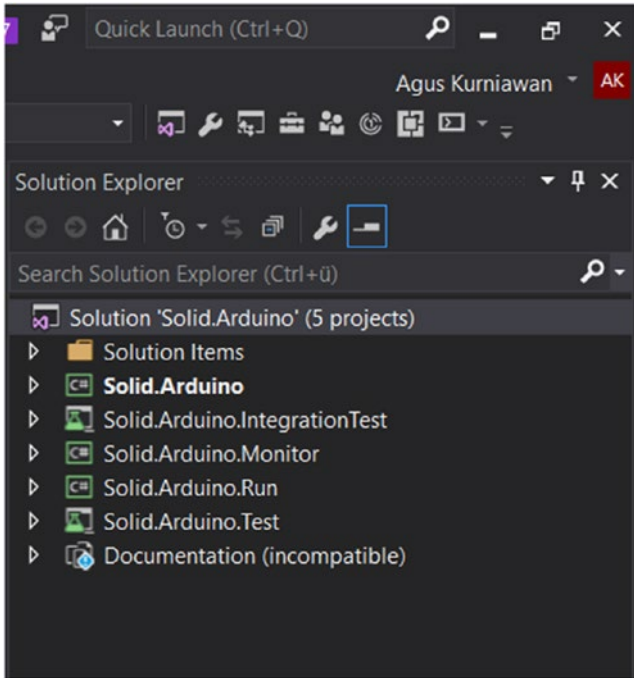


Figure 2-15. Project solution from SolidArduino

After it's compiled, you can get an *.DLL file on a Solid.Arduino project. It will be used in our project. This scenario can be implemented if you separate the application or probably change the .NET version with a specific target. Otherwise, you can add this project into your .NET application project by adding a project reference.

In this scenario, we build a Console application to reduce a complexity GUI of the .NET application. Now create a new project in Visual Studio. Select "Console Application" for your project type. For instance, my project name is "FirmataApp", which is shown in Figure 2-16.

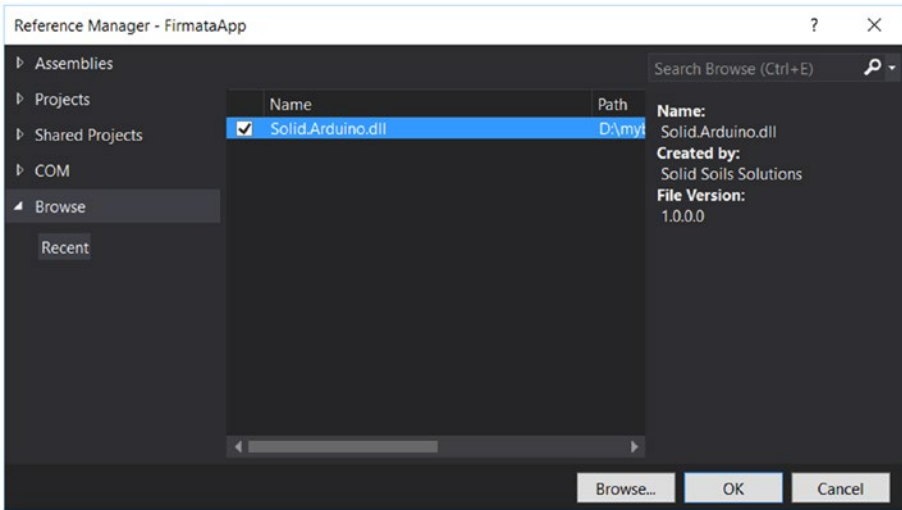


Figure 2-16. Add SolidSoils4Arduino (SolidArduino.dll) into our project

Furthermore, we add a Solid.Arduino library project into your project. You can add it via Reference Manager.

In this scenario, we use the same wiring from the previous section. We use three LEDs and try to control these LEDs from .NET.

After you've created a project in Visual Studio, you can modify Program.cs. First, we instantiate our SerialConnection and ArduinoSession object. Change the Arduino port based on your configuration.

To access digital output in Arduino, we set digital the pin mode using the SetDigitalPinMode() method. We can turn on/off the LED using the SetDigitalPin() method.

For a demo, we build an application to turn on/off three LEDs sequentially. Now write the following codes in a Program.cs file:

```
using System;
using System.Threading;
using Solid.Arduino;
using Solid.Arduino.Firmata;

namespace FirmataApp
{
    class Program
    {
        static bool running = true;
        static void Main(string[] args)
        {

            try
            {
```

```

var connection = new SerialConnection("COM5",
SerialBaudRate.Bps_57600);
var session = new ArduinoSession(connection, timeOut: 250);

Console.CancelKeyPress += delegate (object sender,
ConsoleCancelEventArgs e)
{
    e.Cancel = true;
    running = false;
};

IFirmataProtocol firmata = (IFirmataProtocol)session;

int led1 = 12;
int led2 = 11;
int led3 = 10;

firmata.SetDigitalPinMode(led1, PinMode.DigitalOutput);
firmata.SetDigitalPinMode(led2, PinMode.DigitalOutput);
firmata.SetDigitalPinMode(led3, PinMode.DigitalOutput);

while (running)
{
    // led 1
    Console.WriteLine("Turn on LED 1");
    firmata.SetDigitalPin(led1, true);
    firmata.SetDigitalPin(led2, false);
    firmata.SetDigitalPin(led3, false);
    Thread.Sleep(1000); // sleep

    // led 2
    Console.WriteLine("Turn on LED 2");
    firmata.SetDigitalPin(led1, false);
    firmata.SetDigitalPin(led2, true);
    firmata.SetDigitalPin(led3, false);
    Thread.Sleep(1000);

    // led 3
    Console.WriteLine("Turn on LED 3");
    firmata.SetDigitalPin(led1, false);
    firmata.SetDigitalPin(led2, false);
    firmata.SetDigitalPin(led3, true);
    Thread.Sleep(1000);
}
// turn off LEDs
firmata.SetDigitalPin(led1, false);
firmata.SetDigitalPin(led2, false);
firmata.SetDigitalPin(led3, false);

connection.Close();
}

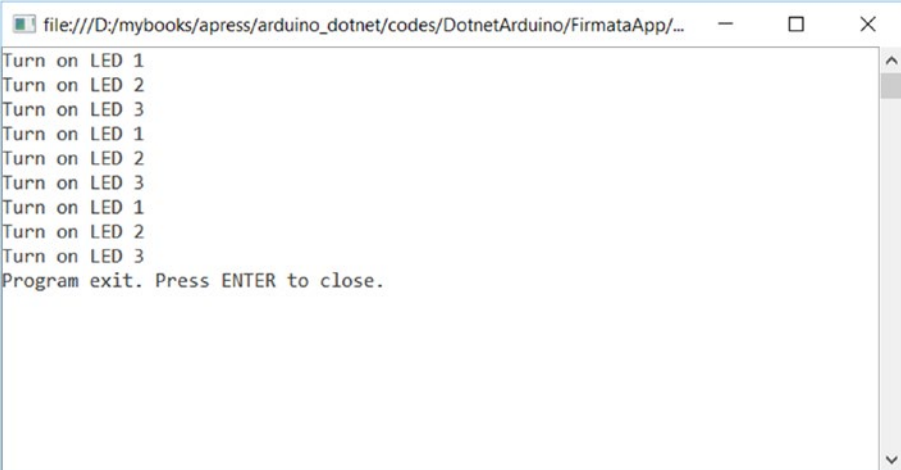
```

```

        catch (Exception err)
        {
            Console.WriteLine(err);
        }
        Console.WriteLine("Program exit. Press ENTER to close.");
        Console.ReadLine();
    }
}
}

```

Save the program. Then, run the program. You should see three blinking LEDs. In Terminal, you also see messages, shown in Figure 2-17 as a sample.



```

file:///D:/mybooks/apress/arduino_dotnet/codes/DotnetArduino/FirmataApp/...
Turn on LED 1
Turn on LED 2
Turn on LED 3
Turn on LED 1
Turn on LED 2
Turn on LED 3
Turn on LED 1
Turn on LED 2
Turn on LED 3
Program exit. Press ENTER to close.

```

Figure 2-17. Program output for controlling LEDs from .NET app

You can build your own .NET application with a specific problem to control the Arduino board over the Firmata protocol. You can utilize the SolidSoils4Arduino library to control the Arduino board over the Firmata protocol. For instance, you send a command to the Arduino board to open a door after a .NET application receives a message order. Moreover, you can build a Windows Forms and a WPF application to communicate with Arduino boards.

Summary

We learned various Arduino I/O communications and tried to build Sketch programs to access those. Furthermore, we learned how to control Arduino from a .NET Application. This application sends a specific command to Arduino through Arduino I/O communication. In the last section, we continued to use a Firmata protocol, one of the protocols to communicate to Arduino from the application, to control our Arduino from the .NET application. In the next chapter, we will learn how to sense and actuate on Arduino boards.

CHAPTER 3



Sensing and Actuating

Sensing and actuating are common activities in an embedded control system. These involve external devices such as sensors and actuators. In this chapter, we learn several sensor and actuator devices on Arduino boards using a Sketch program and then they will be used by a .NET application.

This chapter covers the following topics:

- Overview of sensing and actuating in Arduino
- Exploring sensor and actuator devices
- Creating an Arduino sensing app using .NET
- Creating an Arduino actuating app using .NET

Overview of Sensing and Actuating in Arduino

Sensing is a process of converting from physical object to digital data. For instance, sensing temperature. A sensor device can sense its environment temperature and then it convert to digital form. Actuating is a process in which an MCU sends digital data to an actuator device to perform something, such as an LED and motor. Figure 3-1 shows the process of both. To capture and convert physical object to digital form, Arduino uses sensor devices such as a camera to measure temperature and humidity. After obtaining the digital form of a physical object, we can perform a computation based on your scenario. On the other hand, if you want to make Arduino do something, you can perform an actuating process using actuator devices such as a servo motor and a relay module.



Figure 3-1. Sensing and actuating on Arduino board

Arduino is a development board which is designed to be attached to sensor and actuator devices through its I/O pins or Arduino shield module. These external devices, which are connected to an Arduino board, can be accessed via I/O protocols which we already learned in the previous chapter.

Arduino can sense and actuate simultaneously. For instance, Arduino senses temperature and humidity. If the measurement result is achieved in a certain threshold, Arduino can perform something via actuator devices, such as turning on a relay module or a servo motor.

In the next section, we explore sensor and actuator devices as an example, so we use those devices in our demo.

Exploring Sensor and Actuator Devices

In this section, we'll review various sensors and actuators and then see how to use them from an Arduino board using Sketch. These devices can be attached to an Arduino board, but some of those devices need an additional treatment such as command codes, data length, and maximum voltage, in order for Arduino to communicate with these devices. You should check their datasheet from manufacturer.

Sensor Devices

Sensor devices usually convert from physical objects to digital form. There are many sensor devices whose sensing features you can integrate with Arduino boards. A sample of a sensor device list is found on the SparkFun website, in their electronics online store: <https://www.sparkfun.com/categories/23>.

In general, each sensor device has one feature to sense the physical object, for instance, acceleration, temperature, and humidity. However, some sensor devices have more than one sensing feature, for example, the DHT22 module (<https://www.adafruit.com/products/385> from Adafruit, and <https://www.sparkfun.com/products/10167> from SparkFun). This sensor module can sense temperature and humidity in a single module. This module is easy to use with Arduino boards. You can see a form of the DHT22 module in Figure 3-2.

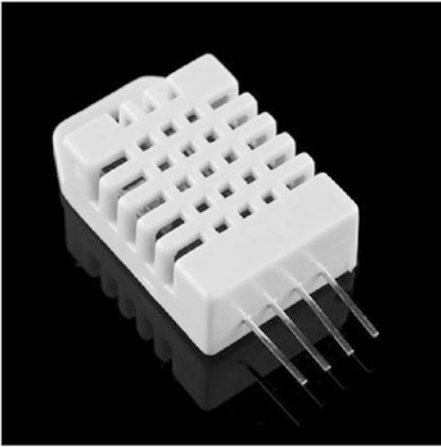


Figure 3-2. DHT22 module

Another sensor device is SparkFun Electret Microphone Breakout from SparkFun (<https://www.sparkfun.com/products/12758>). This module can convert sound to analog form. We can attach this sensor to Arduino analog pins. You can see a SparkFun Electret Microphone Breakout in Figure 3-3.



Figure 3-3. SparkFun Electret Microphone Breakout from SparkFun

In the next section, we will build an Arduino program so you can understand how Arduino senses through sensor devices.

Accessing and Testing the Sensor

To understand how to work with sensing in Arduino, we'll build a simple Sketch program to access a sensor device. In this case, we use a DHT22 module to sense temperature and humidity. The DHT22 module layout can be described in Figure 3-4.

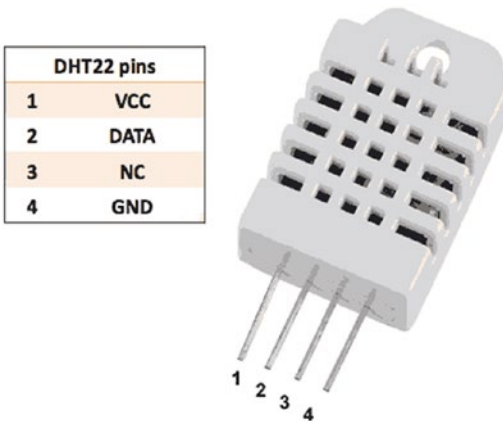


Figure 3-4. DHT22 module layout

You can see a DHT22 layout in Figure 3-4. We can connect DHT22 to our Arduino board. The following is our wiring:

- DHT22 pin 1 to Arduino VCC 3.3V
- DHT22 pin 2 to Arduino digital pin 8
- DHT22 pin GND to Arduino GND

You can see the wiring implementation in Figure 3-5. You can connect this sensor to an Arduino board through jumper cables. For another approach, you can use a breadboard as a medium to which an Arduino board and sensor devices are connected.

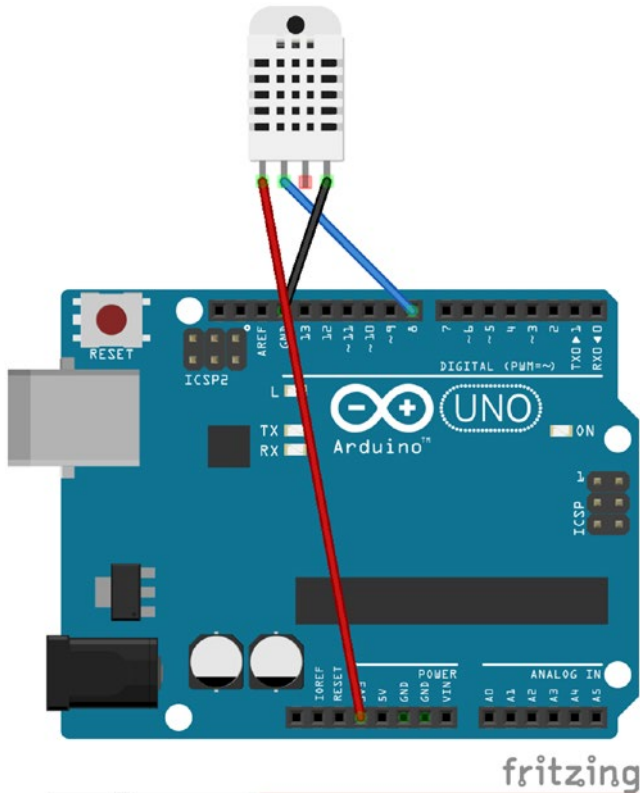


Figure 3-5. Wiring for DHT22 module and Arduino board

The next step is to build a Sketch program. To access DHT22, we can use the DHT sensor library from Adafruit. It can be installed from Arduino IDE. Click menu Sketch ► Include Library ► Manage Libraries. After it's clicked, you should see a dialog which shows a list of Arduino libraries. Find and install the **DHT sensor library**. You can see it in Figure 3-6. Make sure your computer is connected to the Internet to install an additional library.

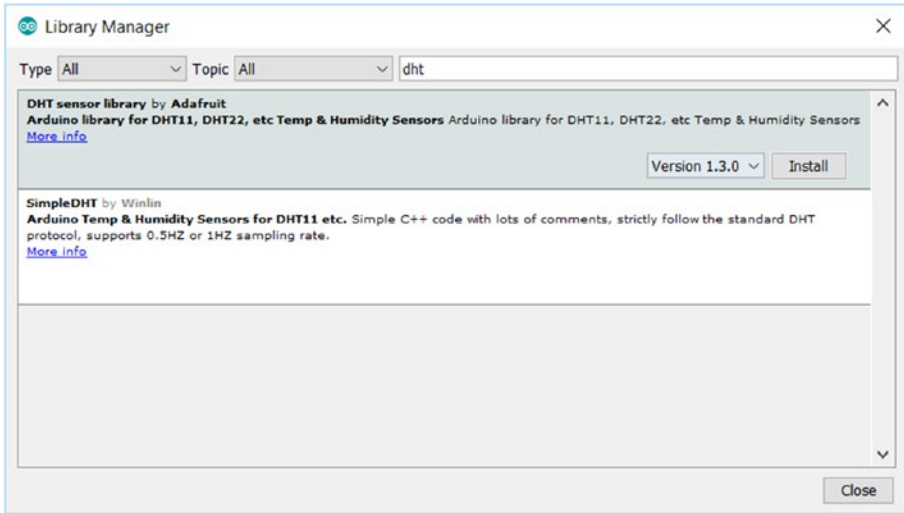


Figure 3-6. Install DHT sensor library from Adafruit

For testing, we write a program to read temperature and humidity from the DHT22 module. After it's sensed, we print sensor data to a UART port. Now open Arduino IDE and configure it for your Arduino board model and port. Write the following code:

```
#include "DHT.h"

// define DHT22
#define DHTTYPE DHT22
// define pin on DHT22
#define DHTPIN 8

DHT dht(DHTPIN, DHTTYPE);

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

void loop() {
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds (its a very slow sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
```

```

// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t)) {
  Serial.println("Failed to read from DHT sensor!");
  return;
}

// Compute heat index in Celsius (isFahreheit = false)
float hic = dht.computeHeatIndex(t, h, false);

Serial.print("Humidity: ");
Serial.print(h);
Serial.print(" %\t");
Serial.print("Temperature: ");
Serial.print(t);
Serial.print(" *C\t");
Serial.print("Heat index: ");
Serial.print(hic);
Serial.println(" *C ");
}

```

Save this program as “ArduinoDHT”.

Then, you can compile and upload the program into the Arduino board. To see our program output, you can run the Serial Monitor tool. A sample of the program output can be seen in Figure 3-7.

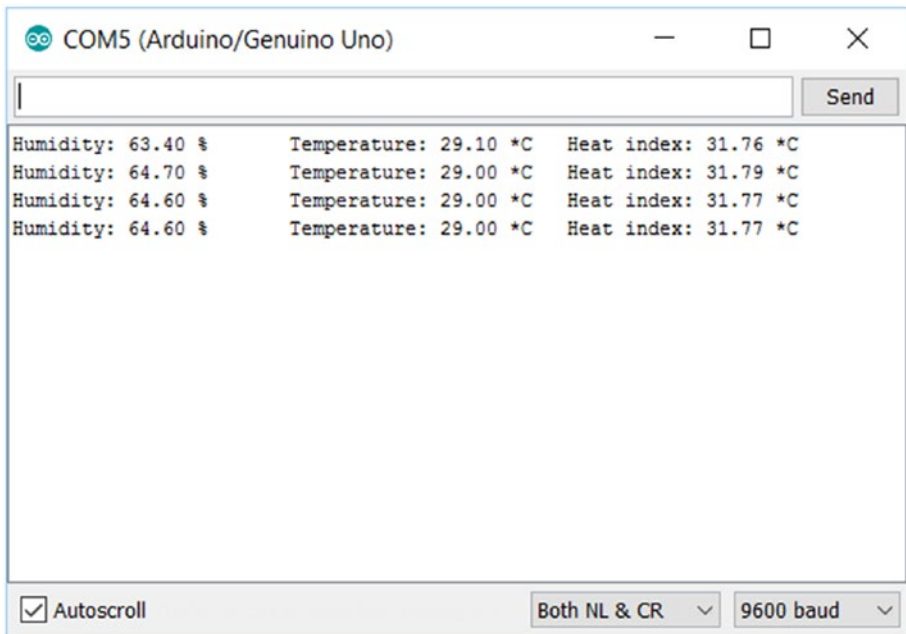


Figure 3-7. Program output for DHT22

How the Program Works

We declare our DHT object and passing DHT module type and data pin via digital pin 8. This needs a DHT.h header file to work with a DHT object.

```
#include "DHT.h"

// define DHT22
#define DHTTYPE DHT22
// define pin on DHT22
#define DHTPIN 8

DHT dht(DHTPIN, DHTTYPE);
```

We activate our DHT and Serial libraries on `setup()` function by calling `begin()` function from these libraries.

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

In `loop()` function from the Sketch program, we read temperature, humidity, and a heat index using a DHT library. All DHT functions are already available for you.

```
float h = dht.readHumidity();
// Read temperature as Celsius (the default)
float t = dht.readTemperature();

// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t)) {
  Serial.println("Failed to read from DHT sensor!");
  return;
}

// Compute heat index in Celsius (isFahreheit = false)
float hic = dht.computeHeatIndex(t, h, false);
```

After reading sensor data from a DHT module, we display these data on a serial port so we can see that data using a Serial Monitor tool.

```
Serial.print("Humidity: ");
Serial.print(h);
Serial.print(" %\t");
Serial.print("Temperature: ");
Serial.print(t);
Serial.print(" *C\t");
Serial.print("Heat index: ");
Serial.print(hic);
Serial.println(" *C ");
```

That's it; now you can build your own demo with several sensor devices. In the next section, we will explore actuator devices and interact with them from Arduino boards.

Actuator Devices

Arduino can actuate to perform something such as blinking LEDs, generating sounds, and moving motors. This task usually has the objective of informing and to giving notification to users or systems so they can take a response.

One of the actuator devices is an RGB LED. It can display a certain color by combining the three colors: Red, Green and Blue. The RGB LED could have two forms: common anode (<https://www.sparkfun.com/products/10820>) and common cathode (<https://www.sparkfun.com/products/9264>). You can see a sample form of an RGB LED in Figure 3-8.



Figure 3-8. RGB LED with common anode form

Another actuator device is servo. There are many models for servo which can be used in our Arduino boards. You can find a servo device easily in your local or online shop, for instance, SparkFun (<https://www.sparkfun.com/products/11965>). Figure 3-9 shows a servo motor from SparkFun which can be applied in Arduino boards.



Figure 3-9. Servo motor

Accessing and Testing the Actuator

As I explained in the previous section, the actuator is a device on which Arduino does perform something. To illustrate how to work with an actuator device, I show you how to use one of the cheapest actuators, RGB LED. RGB LED is a useful actuator which is used to display an indicator, for instance, while Arduino is computing something. Arduino can send an indicator of this state by displaying a specific color on an RGB LED.

In general, RGB LED pin layout can be described, as shown in Figure 3-10, as follows:

- Pin 1 is a red pin.
- Pin 2 is VCC or GND. It depends on the RGB mode: RGB common anode or RGB common cathode.
- Pin 3 is a green pin.
- Pin 4 is a blue pin.



Figure 3-10. RGB LED pin layout

An RGB LED can work if we attach this device to PWM (Pulse Width Modulation) pins. In general, PWM pins on an Arduino board is identified by a (~) sign on digital pins. You can check these pins on Arduino's datasheet; for instance, Arduino UNO (<https://www.arduino.cc/en/Main/ArduinoBoardUno>) has PWM pins on 3, 5, 6, 9, 10, and 11. For testing, I use an Arduino UNO board. I use PWM pins on 9, 10, and 11. An RGB LED is attached to Arduino with the following wiring:

- Pin 1 (red pin) is connected to Arduino digital pin 9.
- Pin 2 (common pin) is connected to Arduino VCC 3.3V or GND.
- Pin 1 (green pin) is connected to Arduino digital pin 10.
- Pin 1 (blue pin) is connected to Arduino digital pin 11.

You can see this wiring in [Figure 3-11](#).

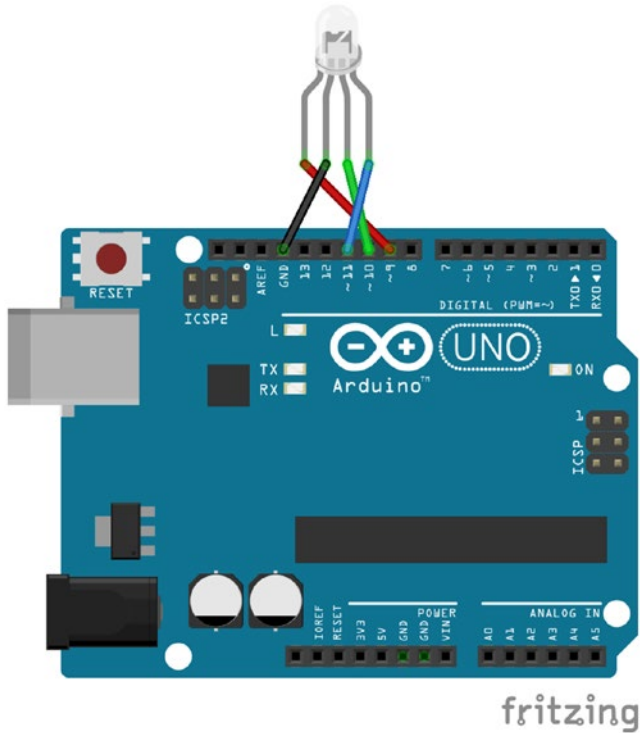


Figure 3-11. Wiring RGB LED and Arduino UNO

We continue to write a Sketch program. We will build a program to display several colors on an RGB LED. The idea of the program to display several colors is combining red, green, and blue values to an RGB LED. You can make your own experiment to display a specific color. In this program, we construct several colors such as Red, Green, Blue, Yellow, Purple, and Aqua. Now you can open an Arduino IDE and write the following code:

```
int redPin = 9;
int greenPin = 10;
int bluePin = 11;

void setup()
{
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  Serial.begin(9600);
}
```

```

void loop()
{
  setColor(0, 255, 255); // red
  Serial.println("red");
  delay(1000);
  setColor(255, 0, 255); // green
  Serial.println("green");
  delay(1000);
  setColor(255, 255, 0); // blue
  Serial.println("blue");
  delay(1000);
  setColor(0, 0, 255); // yellow
  Serial.println("yellow");
  delay(1000);
  setColor(80, 255, 80); // purple
  Serial.println("purple");
  delay(1000);
  setColor(255, 0, 0); // aqua
  Serial.println("aqua");
  delay(1000);
}

void setColor(int red, int green, int blue)
{
  analogWrite(redPin, red);
  analogWrite(greenPin, green);
  analogWrite(bluePin, blue);
}

```

As you see from the code above, we set a specific color by calling the `setColor()` function, which sets color values on red, green, and blue parameters. We use the `analogWrite()` function to write a PWM value on Arduino.

Save this program as “RGBLed”. Then, compile and upload the program into your Arduino board. You should see the RGB LED displaying several colors. You can also see the program output using a Serial Monitor, as shown in Figure 3-12.

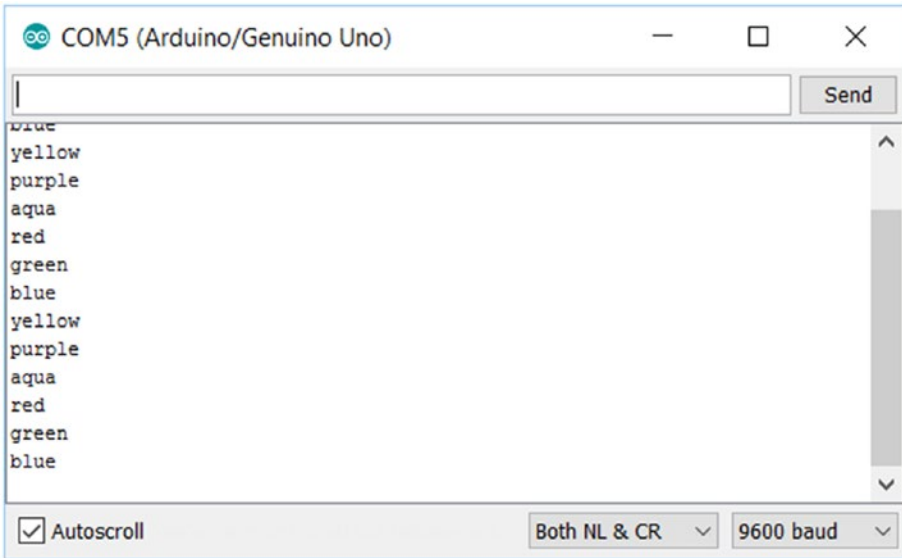


Figure 3-12. Program output for RGB app

Creating an Arduino Sensing App Using .NET

In the previous section, we learned how to access sensor and actuator devices from Arduino boards using Sketch. In this section, we explore how to access sensor devices from a .NET application through Arduino. We have learned communication models between an Arduino board and a computer. It is important because we can use these communication models to retrieve sensor data or perform tasks via actuator devices.

Sensing data from Arduino is done by creating a Sketch program and then publishing to Serial communication, SPI, or I2C protocol. However, you also can implement a Firmata protocol to communicate with an Arduino board.

The new approach which I want to share about sensing is to use JSON as a message format between Arduino and a computer. The idea is that a computer sends data to Arduino via a specific protocol, for instance, UART. Then, Arduino will send a response in JSON, too. This approach is easy to implement. It's also useful if you want to communicate with an external server such as RESTful server.

Fortunately, we can use an ArduinoJson library (<https://github.com/bblanchon/ArduinoJson>) to generate and parse a JSON message. You can download and install it manually. Otherwise, you also can install it via Library Manager, which is shown in Figure 3-13.

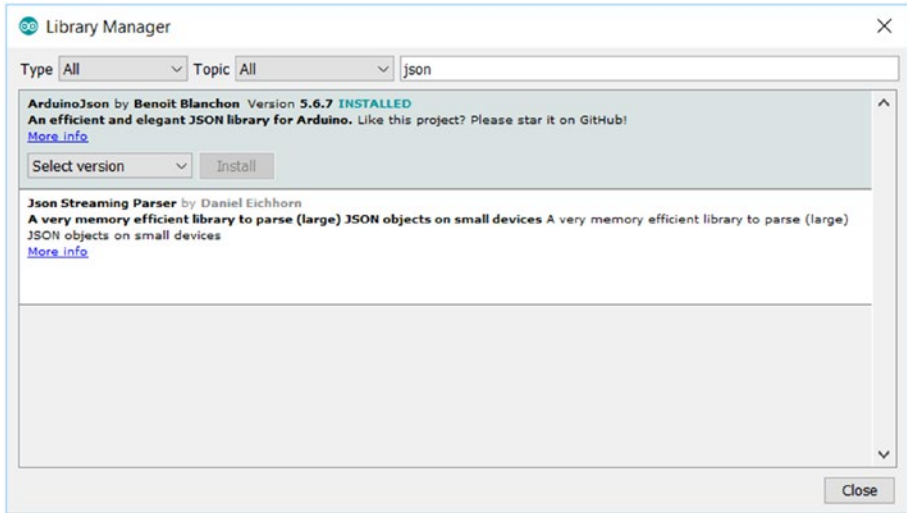


Figure 3-13. Install ArduinoJson

After installing ArduinoJson, we can build a Sketch program. For a scenario case, I use a DHT22 module to sense temperature and humidity. Our Sketch program will sense and then push data to a serial port in JSON format.

In the previous section, we already learned how to access DHT22 using Sketch. In this program, we will use it again. After obtaining the DHT22 data, we wrap the data into JSON format using a JsonObject object.

Now open your Arduino IDE and write the following code:

```
#include "DHT.h"
#include <ArduinoJson.h>

// define DHT22
#define DHTTYPE DHT22
// define pin on DHT22
#define DHTPIN 8

DHT dht(DHTPIN, DHTTYPE);

StaticJsonBuffer<200> jsonBuffer;
JsonObject& root = jsonBuffer.createObject();

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

```

void loop() {
    delay(2000);

    // Reading temperature or humidity takes about 250 milliseconds!
    // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
    float h = dht.readHumidity();
    // Read temperature as Celsius (the default)
    float t = dht.readTemperature();

    // Check if any reads failed and exit early (to try again).
    if (isnan(h) || isnan(t)) {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }

    // Compute heat index in Celsius (isFahreheit = false)
    float hic = dht.computeHeatIndex(t, h, false);

    root["Humidity"] = double_with_n_digits(h, 2);
    root["Temperature"] = double_with_n_digits(t, 2);
    root.printTo(Serial);
    Serial.println("");
}

```

How the Program Works

We instantiate our JSON using a `StaticJsonBuffer` object with 200 characters. Then, we create a JSON object by calling `createObject()` function.

```

StaticJsonBuffer<200> jsonBuffer;
JsonObject& root = jsonBuffer.createObject();

```

In `loop()` function, we read sensor data through DHT22. We obtain humidity `h` and temperature `t`. Then, we pass these values into JSON data. After that, we pass JSON data into a serial port so our program, such as a .NET application, reads data.

```

root["Humidity"] = double_with_n_digits(h, 2);
root["Temperature"] = double_with_n_digits(t, 2);
root.printTo(Serial);
Serial.println("");

```

Save this program as “DHTJson”. Compile and upload this program into the Arduino board.

The next step is to build a .NET application. In this scenario, I use `Json.NET` (<http://www.newtonsoft.com/json>) to parse JSON data which come from Arduino. You can install it via Package Manager Console by typing this command:

```
PM> Install-Package Newtonsoft.Json
```

You can see the installation process in Figure 3-14.

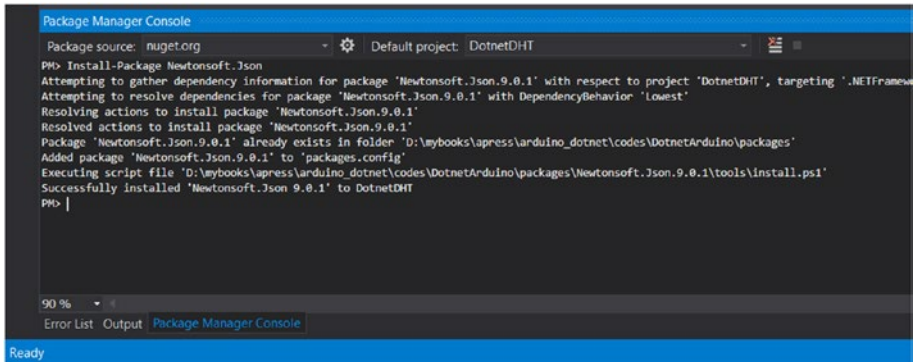


Figure 3-14. Install Json.NET via Package Manager Console

Using Visual Studio, we build a .NET application. We build a program to read data from a serial port. Create a new project with Console Application called “DotnetDHT”. After it’s created, install Json.NET via Package Manager Console. Make sure your computer is connected to the Internet.

Now create a sensor object by adding a class called “Sensor”. Then, write the following codes:

```
using System;

namespace DotnetDHT
{
    class Sensor
    {
        public string Humidity { set; get; }
        public string Temperature { set; get; }
    }
}
```

A Sensor object will be used to hold sensor data from Arduino. We parse JSON data and put it into a Sensor object. We can use `JsonConvert.DeserializeObject<Sensor>()` to parse a string to an object. Now you can modify the Program.cs file to read incoming messages from serial communication and then parse them. The following are codes in the Program.cs file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO.Ports;

using Newtonsoft.Json;
namespace DotnetDHT
```



```

{
class Program
{
    static bool running = true;
    static void Main(string[] args)
    {
        try
        {
            SerialPort serialPort = new SerialPort("COM5");
            serialPort.BaudRate = 9600;

            Console.CancelKeyPress += delegate (object sender,
            ConsoleCancelEventArgs e)
            {
                e.Cancel = true;
                running = false;
            };

            serialPort.Open();
            while (running)
            {
                string data = serialPort.ReadLine();
                if (!string.IsNullOrEmpty(data))
                {
                    Console.WriteLine(data);
                    if(data.Contains("Humidity") && data.
                    Contains("Temperature"))
                    {
                        Sensor s = JsonConvert.DeserializeObject<Sensor
                        >(data);

                        Console.WriteLine("Temperature: {0} ^C",
                        s.Temperature);
                        Console.WriteLine("Humidity: {0}", s.Humidity);
                    }
                }
            }
            serialPort.Close();
        }
        catch (Exception err)
        {
            Console.WriteLine(err);
        }
        Console.WriteLine("Program exit. Press ENTER to close.");
        Console.ReadLine();
    }
}
}

```

Save the program. Please change the serial port for your Arduino. This program uses COM5 since my Arduino was detected as COM5 on Windows. You should change it by your Arduino serial port. Please read Chapter 1 to check a serial port of an Arduino board.

Now you can compile and run the program from Visual Studio directly. You should see incoming messages from Arduino. These messages will be parsed and shown in Terminal for temperature and humidity values.

You can see the sample of program output in Figure 3-15.

```

file:///D:/mybooks/apress/arduino_dotnet/codes/DotnetArduino/DotnetDHT/...
Temperature: 28.20 ^C
Humidity: 65.90
{"Humidity":65.80,"Temperature":28.20}
Temperature: 28.20 ^C
Humidity: 65.80
28.20}
{"Humidity":66.00,"Temperature":28.20}
Temperature: 28.20 ^C
Humidity: 66.00
{"Humidity":65.90,"Temperature":28.20}
Temperature: 28.20 ^C
Humidity: 65.90
{"Humidity":65.80,"Temperature":28.20}
Temperature: 28.20 ^C
Humidity: 65.80
{"Humidity":66.00,"Temperature":28.20}
Temperature: 28.20 ^C
Humidity: 66.00
{"Humidity":66.00,"Temperature":28.20}
Temperature: 28.20 ^C
Humidity: 66.00
{"Humidity":66.10,"Temperature":28.20}
Temperature: 28.20 ^C
Humidity: 66.10
  
```

Figure 3-15. Program output for reading sensor data from Arduino

How the Program Works

The program configures a serial port of Arduino. In this case, I use COM5 for my Arduino. You should change it with a serial port of Arduino in Windows.

```

SerialPort serialPort = new SerialPort("COM5");
serialPort.BaudRate = 9600;
  
```

Since our program runs continuously, we need to break our program by pressing CTRL+C. We utilize `CancelKeyPress` delete from `Console` object. If the user presses CTRL+C, we change `running` value to `false` so our looping program will break.

```
Console.CancelKeyPress += delegate (object sender, ConsoleCancelEventArgs e)
{
    e.Cancel = true;
    running = false;
};
```

Next, our program opens a serial port and reads incoming data from a serial port using `ReadLine()` method. Then, we display our sensor data to the Terminal. Since sensor data is JSON format, we should parse JSON data using `JsonConvert.DeserializeObject<>()`. We will obtain `Sensor` data as result of the JSON parser. Finally, we close our serial port connection by calling `Close()` method.

```
serialPort.Open();
while (running)
{
    string data = serialPort.ReadLine();
    if (!string.IsNullOrEmpty(data))
    {
        Console.WriteLine(data);
        if(data.Contains("Humidity") && data.Contains("Temperature"))
        {
            Sensor s = JsonConvert.DeserializeObject<Sensor>(data);

            Console.WriteLine("Temperature: {0} ^C", s.Temperature);
            Console.WriteLine("Humidity: {0}", s.Humidity);
        }
    }
}
serialPort.Close();
```

This is the last of our .NET programs to sense from Arduino. You can do more practice using another sensor device. Then, a .NET program will read the sensor data.

Creating an Arduino Actuating App Using .NET

We have learned to access sensor data from Arduino using .NET. You can get more practice by trying to work some sensor devices. In this section, we build a .NET application to actuate via actuator devices.

For testing, I use an RGB LED as an actuator device. We already learned to work with RGB LED using Sketch. Now we will work with this actuator using .NET. We change the color in RGB LEDs from our .NET application. Communication between Arduino and a computer uses Firmata protocol so you should install a Standard Firmata Sketch in your Arduino board.

Open Visual Studio and create a Windows Form application called “RGBControlApp”. We build a UI form, which is shown in Figure 3-16.

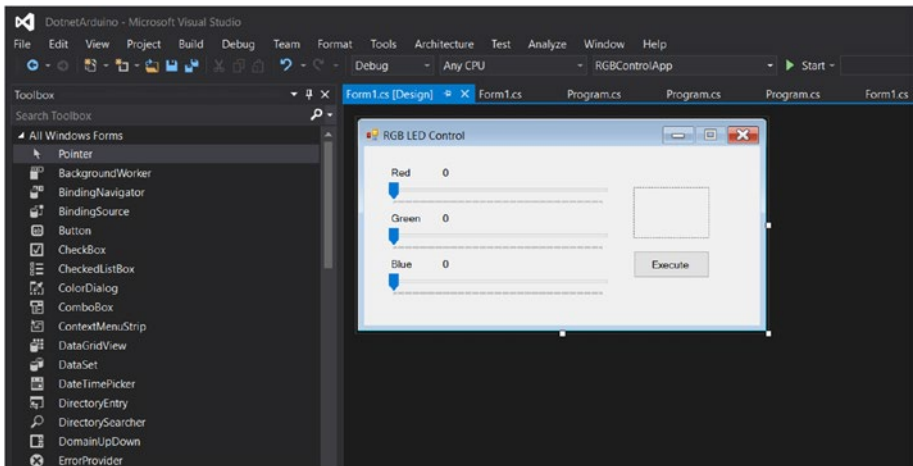


Figure 3-16. Build UI application

After creating the project, we should add a SolidSoils4Arduino library (<https://github.com/SolidSoils/Arduino>) into our project. It is a Firmata client for .NET. We have learned it in the previous chapter.

In our Windows Form (Form1.cs), we add a SolidSoils4Arduino library by writing these codes:

```
using Solid.Arduino;
using Solid.Arduino.Firmata;
```

We declare several variables and set the initialization values from a `TrackBar` object on a constructor of `Form`.

```
public partial class Form1 : Form
{
    private int redVal;
    private int greenVal;
    private int blueVal;
    public Form1()
    {
        InitializeComponent();

        redVal = trackRed.Value;
        greenVal = trackGreen.Value;
        blueVal = trackBlue.Value;
    }
}
```

While users change color values on `TrackBar` objects, we should catch this event. We can use a `ValueChanged` event. You should catch it for all `TrackBar` objects. Then, we read a color value from `TrackBar` and update our panel color.

```
private void trackRed_ValueChanged(object sender, EventArgs e)
{
    redVal = trackRed.Value;
    lbRed.Text = redVal.ToString();
    UpdateColorPanel();
}

private void trackGreen_ValueChanged(object sender, EventArgs e)
{
    greenVal = trackGreen.Value;
    lbGreen.Text = greenVal.ToString();
    UpdateColorPanel();
}

private void trackBlue_ValueChanged(object sender, EventArgs e)
{
    blueVal = trackBlue.Value;
    lbBlue.Text = blueVal.ToString();
    UpdateColorPanel();
}
```

`UpdateColorPanel()` function updates the panel color by changing `BackColor` from panel properties.

```
private void UpdateColorPanel()
{
    panelColor.BackColor = Color.FromArgb(redVal, greenVal, blueVal);
}
}
```

When users click the Execute button, our program will connect to the Arduino board and change the color in the RGB LED. Please change the Arduino port. In this case, my Arduino was detected as COM5. You should change it for your Arduino serial port.

```
private void btnExecute_Click(object sender, EventArgs e)
{
    try
    {
        var connection = new SerialConnection("COM5", SerialBaudRate.
            Bps_57600);
        var session = new ArduinoSession(connection, timeOut: 250);

        IFirmataProtocol firmata = (IFirmataProtocol)session;

        int redPin = 9;
        int greenPin = 10;
        int bluePin = 11;

        firmata.SetDigitalPinMode(redPin, PinMode.PwmOutput);
        firmata.SetDigitalPinMode(greenPin, PinMode.PwmOutput);
        firmata.SetDigitalPinMode(bluePin, PinMode.PwmOutput);

        firmata.SetDigitalPin(redPin, redVal);
        firmata.SetDigitalPin(greenPin, greenVal);
        firmata.SetDigitalPin(bluePin, blueVal);

        connection.Close();
    }
    catch(Exception err)
    {
        MessageBox.Show(err.Message);
    }
}
}
```

You can see the program above. When users click the Execute button, the program will send red, green, and blue values to Arduino via Firmata by calling a `SetDigitalPin()` method. Before we call this method, you should declare your pins as `PwmOutput` by calling a `SetDigitalPinMode()` method.

Save the program. Now you can run the program. After running, you can change red, green and blue values by changing `TrackBar`. After it's selected, click the Execute button. Then, you should see that the RGB LED will be lighting with your chosen color. You can see my application form in Figure 3-17.

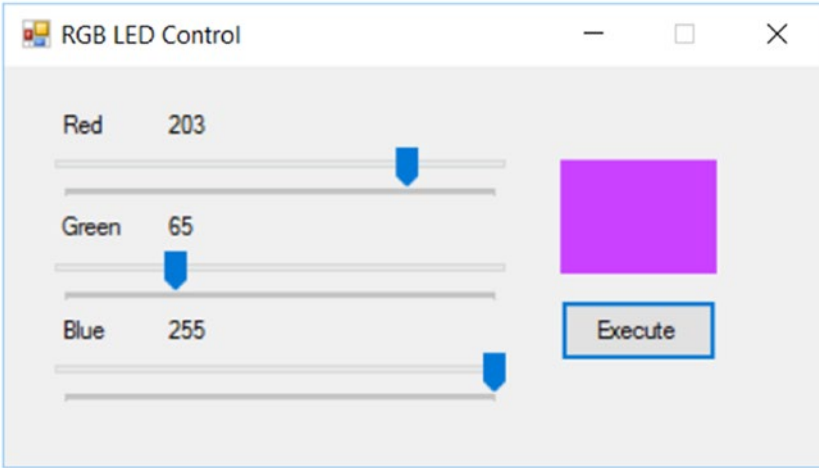


Figure 3-17. Running application for RGBControlApp

Summary

We have learned how to access sensor and actuator devices on Arduino using a sketch program. We use an RGB LED and a DHT22 module as a sample of actuator and sensor. Furthermore, we built a .NET program to access sensor and actuator devices on Arduino. In the next chapter, we will learn how to work Windows Remote Arduino (WRA) for a Universal Windows Application (UWP) platform. This platform can run on Raspberry Pi, Windows tablet, and Windows desktop.

CHAPTER 4



Windows Remote Arduino

Microsoft is working on IoT development intensively. The main technology from Microsoft, .NET Microsoft, is being pushed in order to perform with current IoT platforms such as Arduino and Raspberry Pi. Microsoft released Windows Remote Arduino library to enable Microsoft Windows Platform (called Universal Windows Platform, or UWP) to work on an Arduino platform. UWP technology makes our program run on any Windows platform, such as computer, tablet, smartphone, and IoT boards.

This chapter covers the following topics:

- Setting up Arduino for Windows Remote Arduino
- Building your first program for Windows Remote Arduino
- Controlling Arduino analog I/O
- Remoting Arduino through I2C bus
- Windows Remote Arduino over Bluetooth

In the previous chapter, we learned how to sense and actuate on Arduino from a .NET application. We also tried to communicate with Arduino via Firmata protocol. In this chapter, we will learn how to access an Arduino board from .NET using Windows Remote Arduino.

Windows Remote Arduino is a library from Microsoft in which a Windows application-based .NET framework can control an Arduino board through USB and Bluetooth communication. What I mean in this case by “.NET framework” is .NET framework for Universal Windows Platform. Microsoft has introduced a Universal Windows Platform (UWP) on Windows 8 or later.

Through Universal Windows Platform (UWP), we can develop a .NET application which runs on any Windows platform, such as desktop, smartphone, tablet and IoT boards. Technically, some .NET APIs are cut to generalize application architecture. For further information about Universal Windows Platform, I suggest you read this website: <https://msdn.microsoft.com/en-us/windows/uwp/get-started/whats-a-uwp>.

Figure 4-1 shows how UWP interacts among devices.

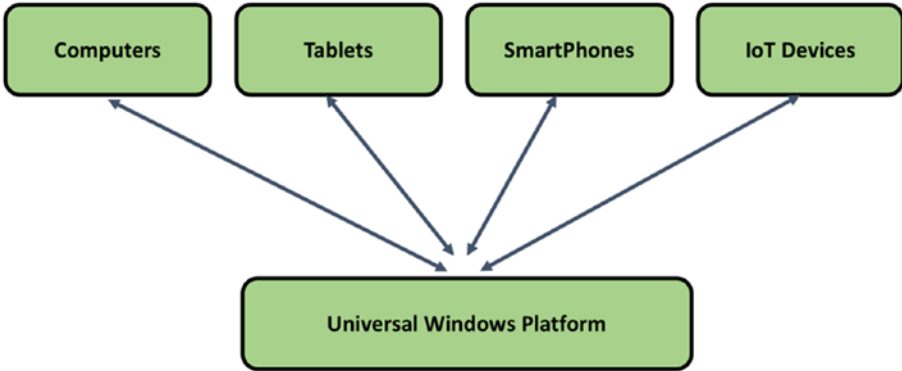


Figure 4-1. *Universal Windows Platform targets*

Windows Remote Arduino is designed for Universal Windows Platform, so if you have Windows tablets such as Microsoft Surface, we can control our Arduino to retrieve sensor information or to actuate some physical objects surrounding an Arduino board.

UWP technology uses a .NET framework for the base of the program. It means UWP framework is a subset of .NET framework. Graphical Unit Interface (GUI) on UWP uses XAML technology. You should be familiar XAML on Windows Presentation Foundation (WPF), so you don't need to invest more in learning UWP if you have experience in WPF.

Setting up Arduino for Windows Remote Arduino

There is no special requirement to work with Windows Remote Arduino. Each Arduino board should be deployed in a Firmata Sketch program in order for the Arduino board to communicate with another device.

The thing that you should understand is that Windows Remote Arduino works on the top of Universal Windows Platform. To develop this app, you should install Windows 8 or later. Currently, I recommend using Windows 10 for development.

To develop Universal Windows Platform (UWP), we can use Visual Studio 2015 or later. It has installed UWP project templates. You can see it in [Figure 4-2](#).

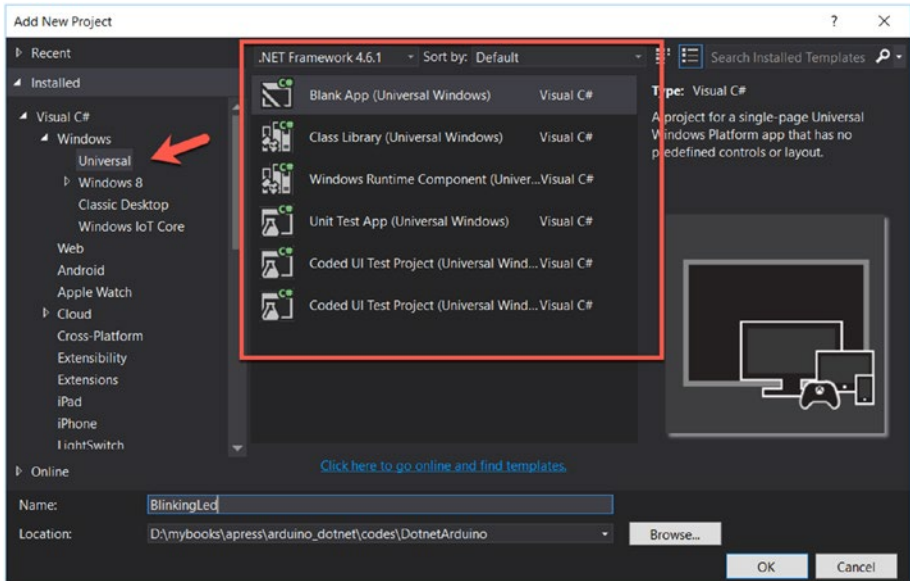


Figure 4-2. Project templates for Universal Windows Platform (UWP)

Building Your First Program for Windows Remote Arduino

In this section, we build the first program for Windows Remote Arduino. We develop a blinking LED application for Universal Windows Platform. In this case, we need three LEDs. We also need a computer with Windows 10 OS installed. These three LEDs will be turned on/off manually from the application.

Let's start!

Wiring

To implement our demo, we should wire our components. Our three LEDs are attached to Arduino with the following scheme:

- LED 1 is connected to Arduino digital pin 12.
- LED 1 is connected to Arduino digital pin 11.
- LED 1 is connected to Arduino digital pin 10.

You may use a resistor on each LED to protect the LED. Finally, you can see my wiring in Figure 4-3.

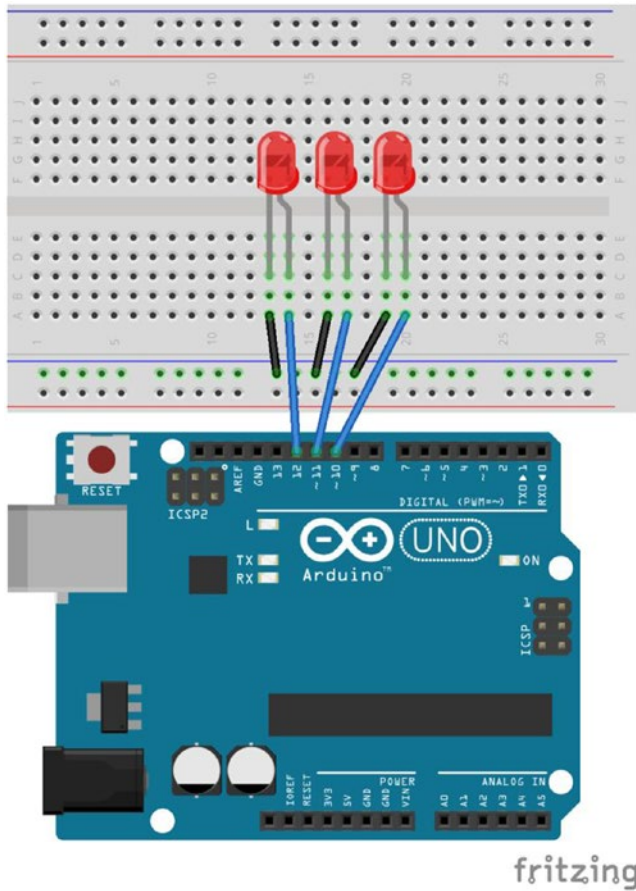


Figure 4-3. Wiring for blinking LEDs

Arduino Program

In order for our application to communicate with Arduino boards, we need to configure our boards. Windows Remote Arduino requires an installed Firmata Sketch program in the Arduino board.

To install Firmata Sketch, you can open Arduino IDE. Then, click menu File ► Examples ► Firmata ► StandardFirmata. You should see the Firmata Sketch program on Arduino IDE. After that, you should change baudrate to 57600. You can see it in Figure 4-4.

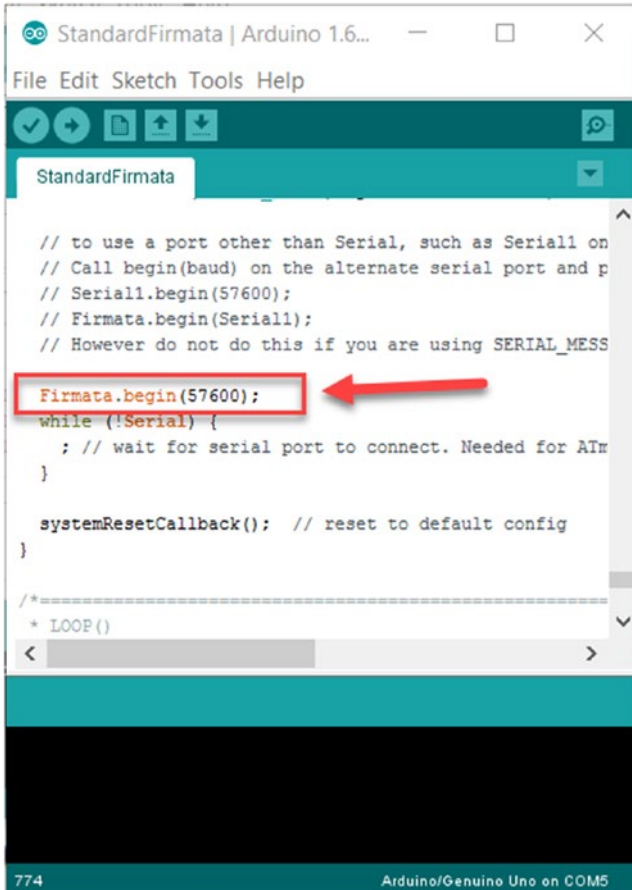


Figure 4-4. Change baudrate on Firmata Sketch

Now you can compile and upload the Firmata Sketch program into your Arduino board. If it's done, your board is ready for receiving commands from a .NET application.

.NET Application Program

The next step is to create a project for a Universal Windows Platform target. On Visual Studio 2015, you can create a new project by selecting “project template” under Windows ► Universal ► Blank App (Universal Windows), as shown in Figure 4-5. Give the project a name: “BlinkingLed”. If you're done, click the OK button.

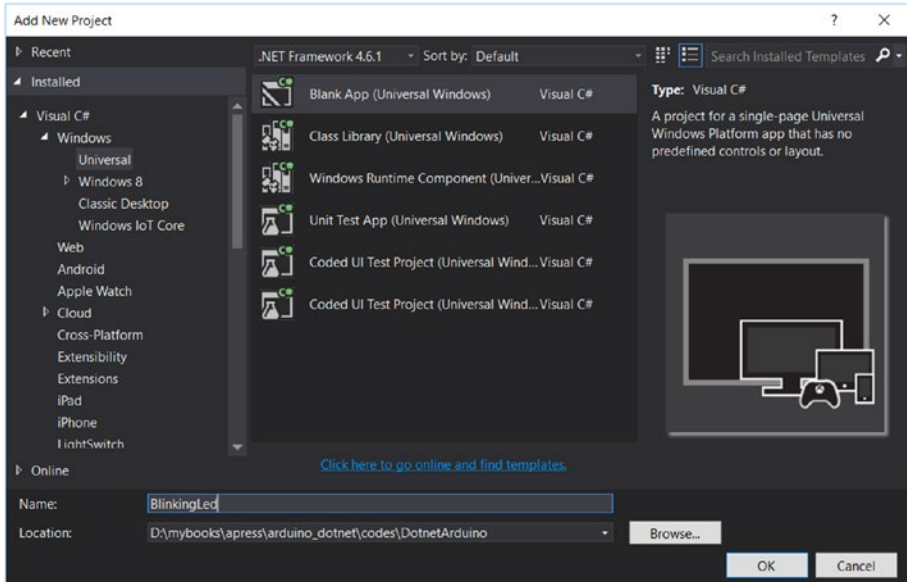


Figure 4-5. Add a new Universal Windows project

After creating a project, you will be asked for a target version. Select the latest version on Target Version. You can see it in Figure 4-6.

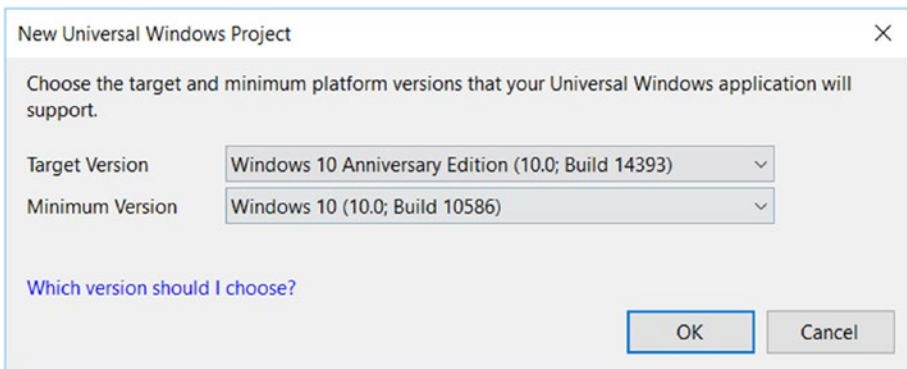


Figure 4-6. Select target version for Universal Windows Platform

If you succeed, you should see your project with XAML GUI. You can see the project file list in Figure 4-7.

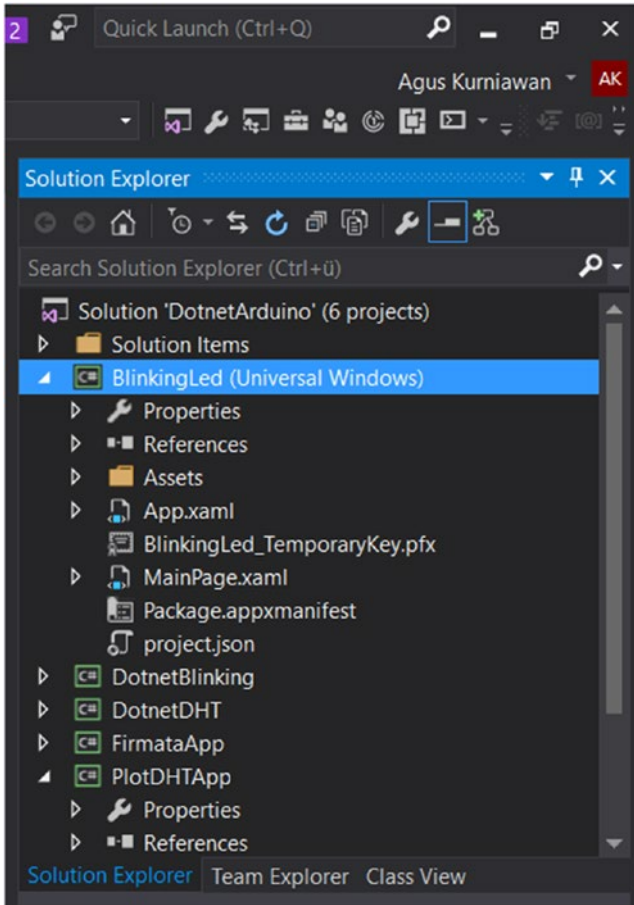


Figure 4-7. Project scheme file for Universal Windows app

Adding Windows Remote Arduino Library

To enable our Universal Windows (UWP) application to communicate with Arduino, we should add Windows Remote Arduino Library. You can click the menu on Visual Studio 2015: Tools ► NuGet Package Manager ► Package Manager Console. Then, you should see Package Manager Console on Visual Studio. Make sure you select our project. Then, type this command:

```
Install-Package Windows-Remote-Arduino
```

This task needs Internet communication. You can see the program output in Figure 4-8.

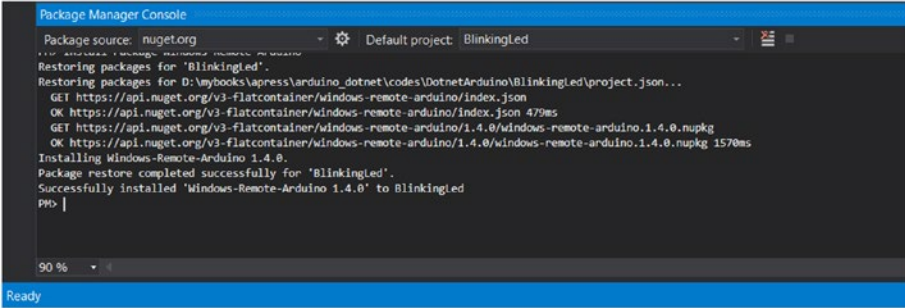


Figure 4-8. Install Windows Remote Arduino through Package Manager Console

After it's added, you should see this library on project scheme, shown in Figure 4-9. You should see Windows-Remote-Arduino on the References menu.

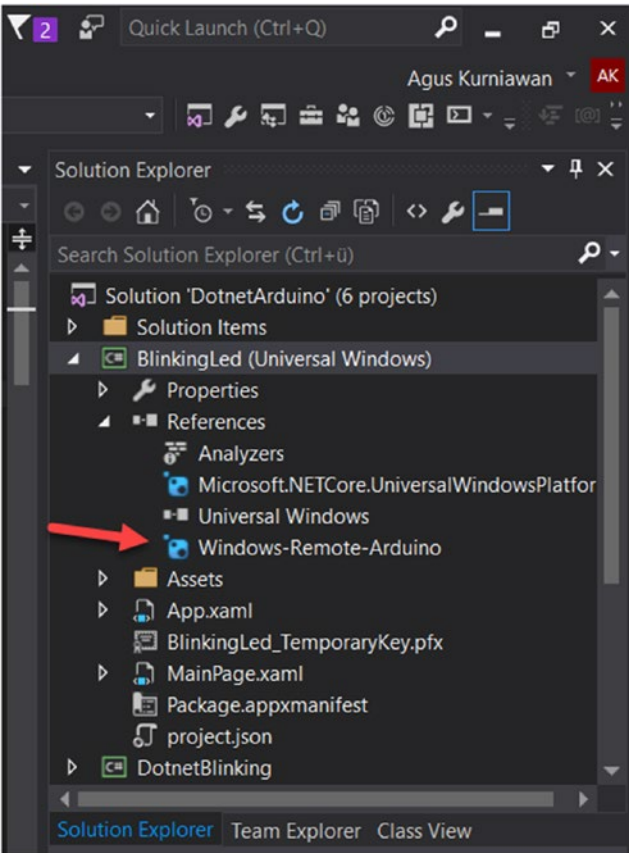


Figure 4-9. Windows Remote Arduino library on project

Writing .NET Program

We use USB as a communication media to control the Arduino board from the application. Since our application uses USB, we should enable this capability. The information about application capabilities can be seen in the Package.appxmanifest file. If you open this file, you should obtain these configurations:

```
<Capabilities>
  <Capability Name="internetClient" />

</Capabilities>
```

Now we should have a serial communication capability. You can add the following bold scripts:

```
<Capabilities>
  <Capability Name="internetClient" />

  <DeviceCapability Name="serialcommunication">
    <Device Id="any">
      <Function Type="name:serialPort"/>
    </Device>
  </DeviceCapability>
</Capabilities>
```

If you're done, save the file.

The next step is to build the application GUI. We add three CheckBoxes, one Textblock, and one button component. We design our GUI on the MainPage.xaml file, which is shown in Figure 4-10.

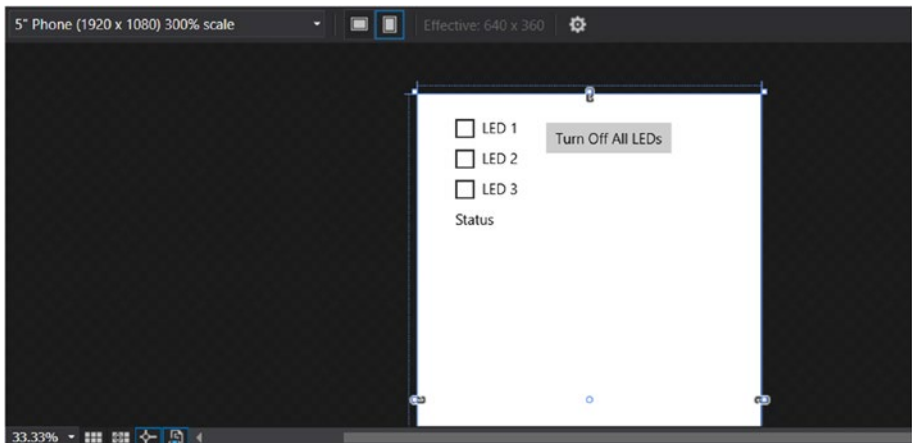


Figure 4-10. Design GUI for BlinkingLed application

The following is a summary of the MainPage.xaml file:

```
<Page
  x:Class="BlinkingLed.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:BlinkingLed"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d">

  <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <CheckBox x:Name="chkLed1" Content="LED 1" Checked="HandleCheck"
      Unchecked="HandleUnchecked" HorizontalAlignment="Left"
      Margin="40,20,0,0" VerticalAlignment="Top" Width="85"/>
    <CheckBox x:Name="chkLed2" Content="LED 2" Checked="HandleCheck"
      Unchecked="HandleUnchecked" HorizontalAlignment="Left"
      Margin="40,52,0,0" VerticalAlignment="Top" Width="85"/>
    <CheckBox x:Name="chkLed3" Content="LED 3" Checked="HandleCheck"
      Unchecked="HandleUnchecked" HorizontalAlignment="Left"
      Margin="40,84,0,0" VerticalAlignment="Top" Width="85"/>
    <TextBlock x:Name="txtStatus" HorizontalAlignment="Left"
      Margin="40,121,0,0" TextWrapping="Wrap" Text="Status"
      VerticalAlignment="Top" Width="256"/>
    <Button x:Name="btnClose" Content="Turn Off All
      LEDs" HorizontalAlignment="Left" Margin="135,30,0,0"
      VerticalAlignment="Top" Click="TurnOffLeds"/>
  </Grid>
</Page>
```

You can see on Button XAML which Click event is passed by TurnOffLeds method. This will be implemented on MainPage.xaml.cs.

Inside MainPage.xaml.cs, we should modify our codes to work with our scenario. Firstly, we add our required libraries.

```
using Microsoft.Maker.Serial;
using Microsoft.Maker.RemoteWiring;
```

We add additional variables in MainPage class.

```
private UsbSerial connection;
private RemoteDevice arduino;
private const byte LED1 = 12;
private const byte LED2 = 11;
private const byte LED3 = 10;
```

We initialize Windows Remote Arduino in MainPage constructor. We create InitWRA() method with the following codes.

```
public MainPage()
{
    this.InitializeComponent();
    this.Unloaded += MainPage_Unloaded;

    InitWRA();
}
private void MainPage_Unloaded(object sender, RoutedEventArgs e)
{
    arduino.Dispose();
}

private void InitWRA()
{
    connection = new UsbSerial("VID_2341", "PID_0043");
    arduino = new RemoteDevice(connection);

    connection.ConnectionEstablished += Connection_ConnectionEstablished;
    connection.begin(57600, SerialConfig.SERIAL_8N1);
}
```

In this code, my Arduino USB is recognized as VID_2341 and PID_0043 parameters for UsbSerial object. To obtain this data, you should open the property of Arduino USB which is detected by your computer. Open Device Manager and expand your Arduino USB. You can see it in Figure 4-11.

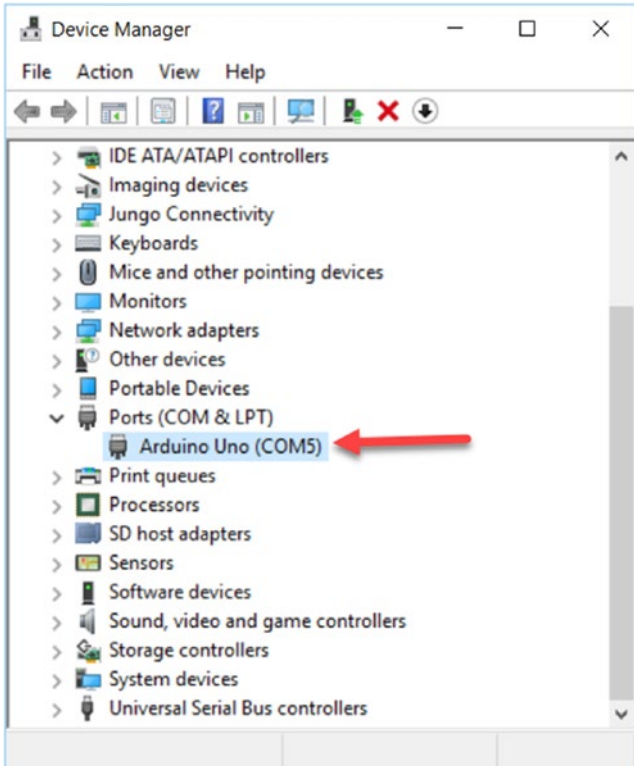


Figure 4-11. *Arduino UNO is detected as COM5 on Device Manager tool*

Open the property of Arduino USB so you obtain a dialog. Click Details tab. Then, select Hardware Ids on Property. You should see VID and PID values for your Arduino USB. You can see them on my Arduino USB, which is shown in Figure 4-12. These values should be inserted into our program.

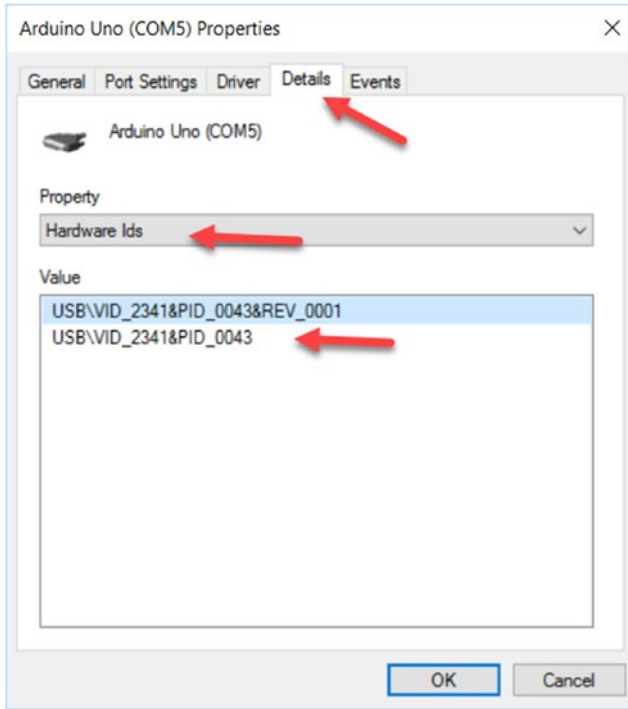


Figure 4-12. VID and PID values for Arduino USB

Now we come back to codes on `MainPage.xaml.cs`. In our method, `InitWRA()`, we implement our `Connection_ConnectionEstablished` event. We also apply our event `Checked` and `Unchecked` on the `CheckBox` component by implementing `HandleCheck()` and `HandleUnchecked()` methods. Finally, we apply `TurnOffLeds()` on `Click` event from a button. The following is the implementation.

```
private void Connection_ConnectionEstablished()
{
    arduino.pinMode(LED1, PinMode.OUTPUT);
    arduino.pinMode(LED2, PinMode.OUTPUT);
    arduino.pinMode(LED3, PinMode.OUTPUT);

    txtStatus.Text = "Connected";
}

private void HandleCheck(object sender, RoutedEventArgs e)
{
    CheckBox cb = sender as CheckBox;
    if (cb.Name == "chkLed1")
    {
        arduino.digitalWrite(LED1, PinState.HIGH);
    }
}
```

```

    if (cb.Name == "chkLed2")
    {
        arduino.digitalWrite(LED2, PinState.HIGH);
    }
    if (cb.Name == "chkLed3")
    {
        arduino.digitalWrite(LED3, PinState.HIGH);
    }
}

private void HandleUnchecked(object sender, RoutedEventArgs e)
{
    CheckBox cb = sender as CheckBox;
    if (cb.Name == "chkLed1")
    {
        arduino.digitalWrite(LED1, PinState.LOW);
    }
    if (cb.Name == "chkLed2")
    {
        arduino.digitalWrite(LED2, PinState.LOW);
    }
    if (cb.Name == "chkLed3")
    {
        arduino.digitalWrite(LED3, PinState.LOW);
    }
}

private void TurnOffLeds(object sender, RoutedEventArgs e)
{
    chkLed1.IsChecked = false;
    chkLed2.IsChecked = false;
    chkLed3.IsChecked = false;
}

```

Save all changes. Our program is ready for testing.

Testing

You can compile our program. Before you run the program, you should deploy the program to a UWP environment. Right-click on the project and then select Deploy on a context menu. It is shown in Figure 4-13.

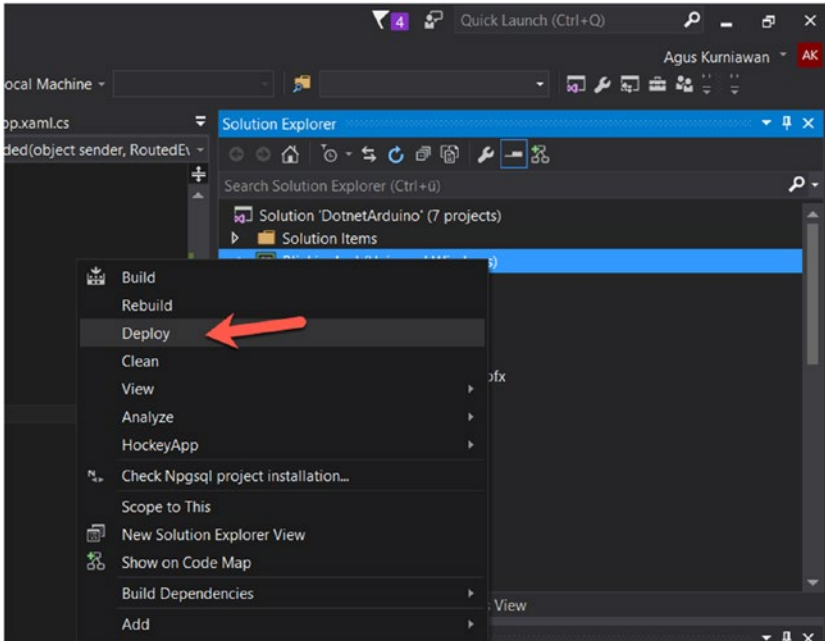


Figure 4-13. Deploy the program

Now you can run the program by clicking a green arrow with Local Machine target. After it's clicked, you should see our program dialog, as shown in Figure 4-14.

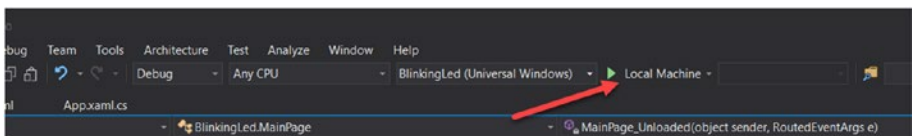


Figure 4-14. Run the program

For testing, you can turn on LED 1, LED 2, or LED 3 by clicking the checkbox.

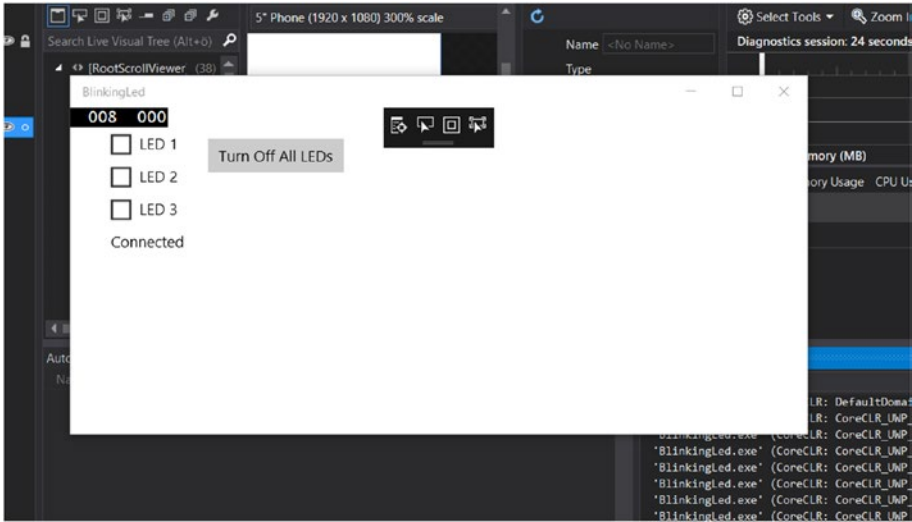


Figure 4-15. The program is running

If you checked the checkbox on LED 1 and LED 2, you should see the lighting LED on LED 1 and LED 2. To turn off all LEDs, you can click the **Turn Off All LEDs** button.

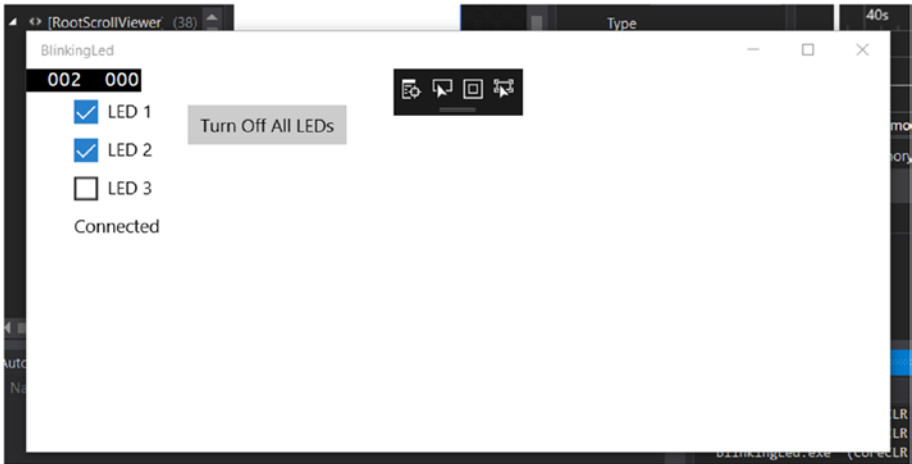


Figure 4-16. Turn on LED 1 and LED 2

We have now seen how to access digital I/O on Arduino through Windows Remote Arduino. In the next section, we continue to access analog I/O on Arduino.

Control Arduino Analog I/O

There are Analog Input and Analog Output, called PWM (Pulse Width Modulation) schemes on Arduino boards. Learning analog I/O is important because several sensor and actuator devices work using analog I/O.

In this section, we learn how to work with analog I/O on Arduino through Windows Remote Arduino library. For testing, we use RGB LED. We have already worked with it in Chapter 3. Now we try to access it using Windows Remote Arduino library.

In general, RGB LED has four pins: red, GND/VCC, green, and blue pins. On pin 2, it could be GND or VCC, depending on what type of RGB LED (anode or cathode LED). You can see this pin layout in Figure 4-17.

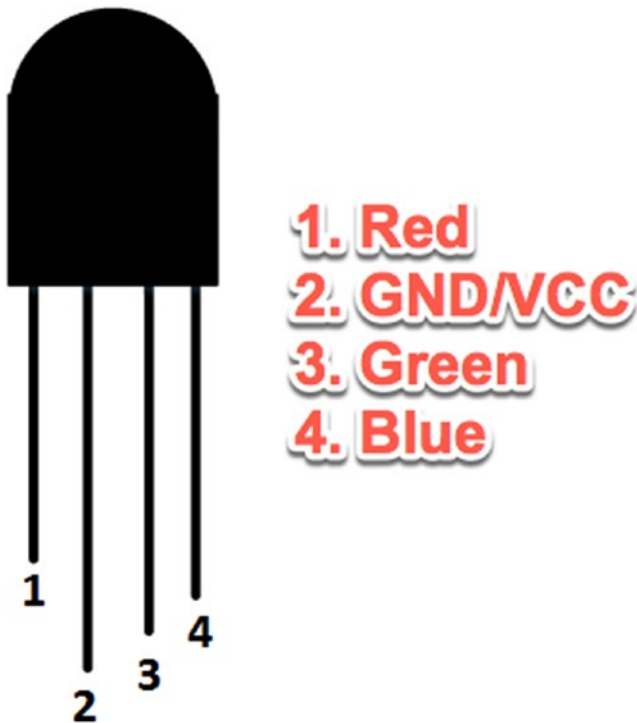


Figure 4-17. RGB LED pinout

Wiring

To implement our demo wiring, you do the following wiring:

- Red pin from RGB LED is connected to digital pin 9 Arduino.
- GND/VCC pin from RGB LED is connected to GND/VCC Arduino.
- Green pin from RGB LED is connected to digital pin 10 Arduino.
- Blue pin from RGB LED is connected to digital pin 11 Arduino.

You can see this wiring in Figure 4-18.

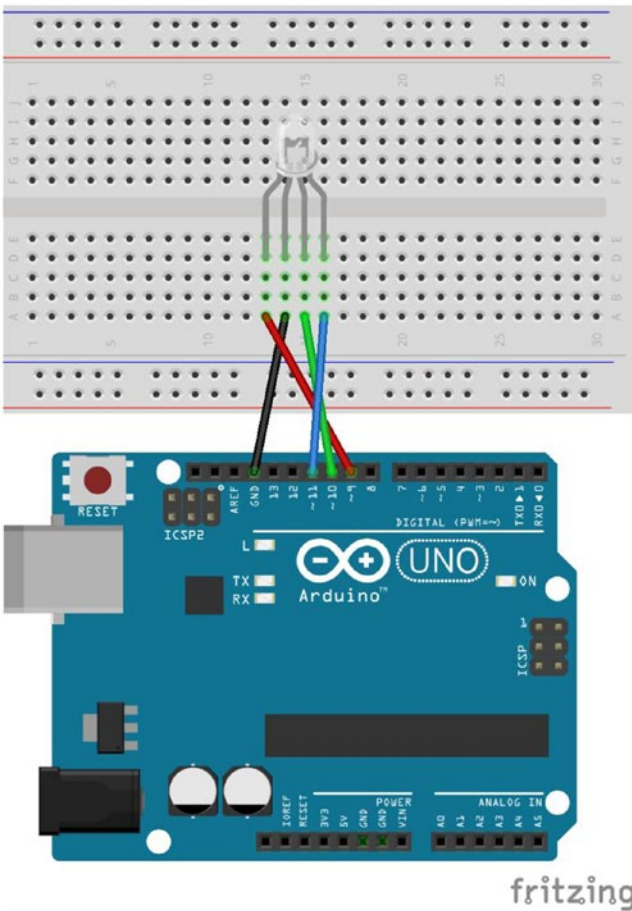


Figure 4-18. Wiring for RGB LED project

Creating a UWP Project

You can create a UWP project from Visual Studio. You have learned it in the previous section. Name the project “RGBDemo”. Then, add Windows Remote Arduino library through Package Manager Console. After it’s created, you should add additional device capability on Package.appxmanifest. The bold script is our additional script.

```
<Capabilities>
  <Capability Name="internetClient" />
  <DeviceCapability Name="serialcommunication">
    <Device Id="any">
      <Function Type="name:serialPort"/>
    </Device>
  </DeviceCapability>
</Capabilities>
```

Arduino Program

To work with a UWP program, you should install a Firmata Skeetch program in Arduino as in the section. Select StandardFirmata. You will perform this task to build a UWP project in this chapter. After StandardFirmata is loaded on Arduino IDE, flash the program into the Arduino board.

■ **Note** All Arduino programs related to a UWP application project should be installed and deployed as StandardFirmata programs into Arduino boards.

Writing the UWP Program

The idea of our program is to control RGB LEDs from a UWP Application. We show a canvas whose color can be changed by sliding red, green, and blue color controls. We will use our app GUI, which is shown in Figure 4-19.

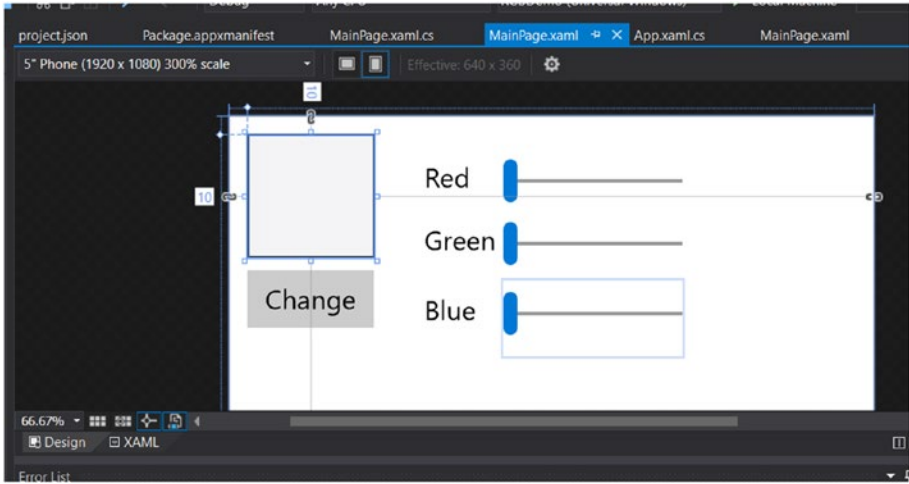


Figure 4-19. Designing UI for RGB LED app

We put a canvas for displaying the current color, three slider controls, and one button to apply changing color. You can see the complete codes for XAML as follows:

<Page

```
x:Class="RGBDemo.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="using:RGBDemo"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d">
```

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Slider x:Name="redSlider" HorizontalAlignment="Left"
        Margin="153,17,0,0" VerticalAlignment="Top" Width="100" Height="43"
        ValueChanged="ChangeRedColor" Maximum="255"/>
    <TextBlock x:Name="textBlock" HorizontalAlignment="Left"
        Margin="109,24,0,0" TextWrapping="Wrap" Text="Red"
        VerticalAlignment="Top"/>
    <Slider x:Name="greenSlider" HorizontalAlignment="Left"
        Margin="153,52,0,0" VerticalAlignment="Top" Width="100" Height="43"
        ValueChanged="ChangeGreenColor" Maximum="255"/>
    <TextBlock x:Name="textBlock_Copy" HorizontalAlignment="Left"
        Margin="109,59,0,0" TextWrapping="Wrap" Text="Green"
        VerticalAlignment="Top"/>
    <Slider x:Name="blueSlider" HorizontalAlignment="Left"
        Margin="153,91,0,0" VerticalAlignment="Top" Width="100" Height="43"
        ValueChanged="ChangeBlueColor" Maximum="255"/>
```

```

<TextBlock x:Name="textBlock_Copy1" HorizontalAlignment="Left"
Margin="109,98,0,0" TextWrapping="Wrap" Text="Blue"
VerticalAlignment="Top"/>
<Button x:Name="button" Content="Change" HorizontalAlignment="Left"
Margin="10,86,0,0" VerticalAlignment="Top" Click="ChangeColor"/>
<Canvas x:Name="cvColorView" HorizontalAlignment="Left"
Height="65" Margin="10,14,0,0" VerticalAlignment="Top" Width="71"
Background="#FFC81010"/>

</Grid>
</Page>

```

Inside Slider controls, we catch `ValueChanged` event by calling `ChangeRedColor()`, `ChangeGreenColor()`, and `ChangeBlueColor()` methods. We also catch `Click` event on button by calling `ChangeColor()` method. We will implement them in `MainPage.xaml.cs` code.

Now we should modify `MainPage.xaml.cs`. First, we add the required libraries by typing these libraries:

```

using Microsoft.Maker.Serial;
using Microsoft.Maker.RemoteWiring;

```

Then, we add several variables to manage our pins and colors.

```

private UsbSerial connection;
private RemoteDevice arduino;
private const byte RED = 9;
private const byte GREEN = 10;
private const byte BLUE = 11;
private byte red;
private byte green;
private byte blue;

```

In constructor, we initialize our WRA library. We set all pins which are used for RGB LEDs in PWM mode. Change “VID_2341” and “PID_0043” values for your Arduino USB, which we have learned in the previous section.

```

public MainPage()
{
    this.InitializeComponent();
    red = 0;
    green = 0;
    blue = 0;

    this.Unloaded += MainPage_Unloaded;
    InitWRA();
}

```

```

private void MainPage_Unloaded(object sender, RoutedEventArgs e)
{
    arduino.Dispose();
}
private void InitWRA()
{
    connection = new UsbSerial("VID_2341", "PID_0043");
    arduino = new RemoteDevice(connection);

    connection.ConnectionEstablished += Connection_ConnectionEstablished;
    connection.Begin(57600, SerialConfig.SERIAL_8N1);
}
private void Connection_ConnectionEstablished()
{
    System.Diagnostics.Debug.WriteLine("Connected");

    arduino.PinMode(RED, PinMode.PWM);
    arduino.PinMode(GREEN, PinMode.PWM);
    arduino.PinMode(BLUE, PinMode.PWM);
}

```

In the MainPage.xaml file, we set our events for ValueChanged and Click event for Button. We can implement as follows:

```

private void ChangeGreenColor(object sender, RangeBaseValueChangedEventArgs e)
{
    red = (byte)redSlider.Value;
    UpdatePanel();
}

private void ChangeRedColor(object sender, RangeBaseValueChangedEventArgs e)
{
    green = (byte)greenSlider.Value ;
    UpdatePanel();
}

private void ChangeBlueColor(object sender, RangeBaseValueChangedEventArgs e)
{
    blue = (byte)greenSlider.Value;
    UpdatePanel();
}

```

```
private void ChangeColor(object sender, RoutedEventArgs e)
{
    // change color on RGB LED
    arduino.analogWrite(RED, red);
    arduino.analogWrite(GREEN, green);
    arduino.analogWrite(BLUE, blue);
}

private void UpdatePanel()
{
    cvColorView.Background = new SolidColorBrush(Color.FromArgb(255, (byte)
red, (byte)green, (byte)blue));
}

```

You can see the codes that we set RGB LED from three color inputs. If done, save all files.

Testing

You can now compile and run the program. If you get an error while running the program, you should deploy the program and then run it. You can see the program, as shown in Figure 4-20.

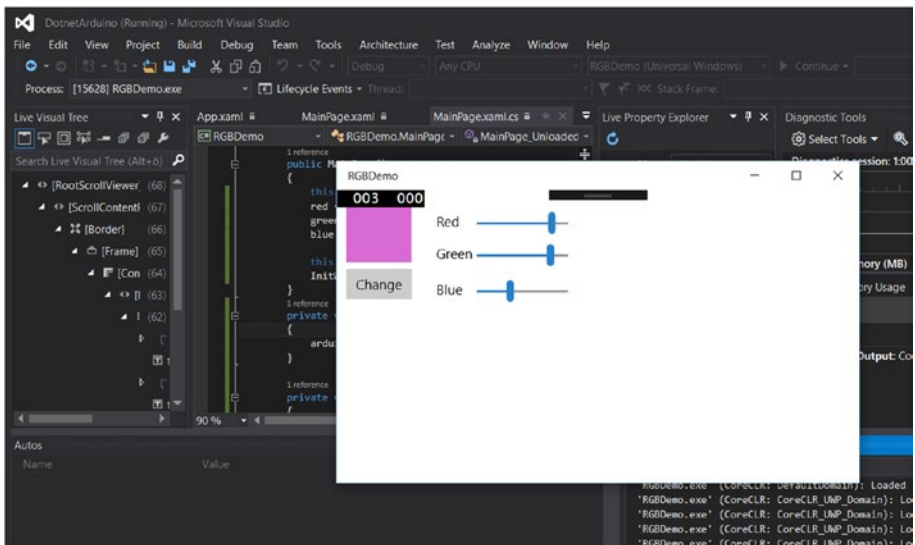


Figure 4-20. RGB LED app is running

Change the current color by sliding three colors—red, green, and blue—on slider controls. If it's done, click Change button to apply the selected color to RGB LED. You can see the sample of RGB LED output in Figure 4-21.

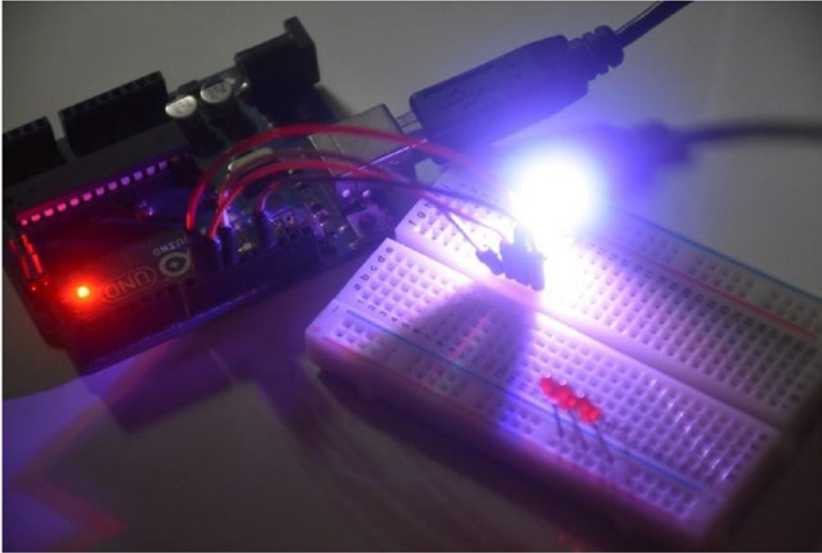


Figure 4-21. RGB LED is lighting based on selecting red, green, and blue colors

Remote Arduino Through I2C Bus

The I2C (Inter-Integrated Circuit) bus or TWI (Two Wire) Interface is used to communicate between MCU and several external devices. The I2C bus consists of two wires: SDA (Serial Data Line) and SCL (Serial Clock Line). Each Arduino board has I2C on specific pins. You can check I2C pins on the Arduino layout; for instance, Arduino UNO (based on document <https://www.arduino.cc/en/Main/ArduinoBoardUno>) has I2C pins on SDA pin on A4 pin and SCL pin on A5 pin.

In this section, we learn how to access I2C from a .NET application through Windows Remote Arduino library. For testing, I use a PCF8591 AD/DA module which consists of thermistor, photoresistor, and potentiometer. Since this module uses IC PCF8591, this module can be accessed on I2C. You can find this module easily in an online store, for example, eBay and Aliexpress.

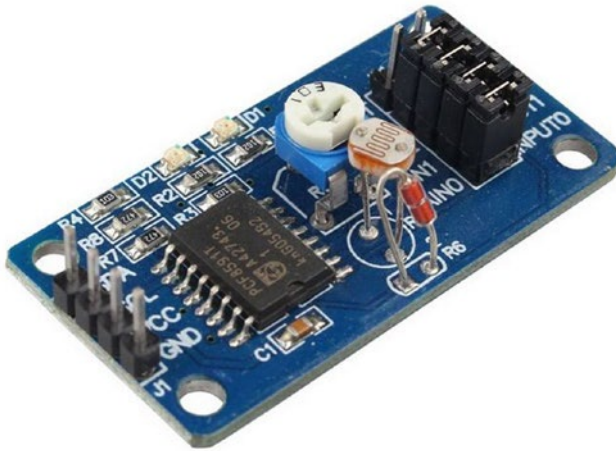


Figure 4-22. PCF8591 AD/DA module

Wiring for I2C Application

We can connect a PCF8591 AD/DA module to an Arduino UNO board directly with the following wiring:

- PCF8591 module VCC is connected to Arduino 3.3V.
- PCF8591 module GND is connected to Arduino GND.
- PCF8591 module SDA is connected to Arduino SDA (A4 pin).
- PCF8591 module SCL is connected to Arduino SCL (A5 pin).

A sample of my hardware wiring can be seen in Figure 4-23. A PCF8591 AD/DA module is placed into a breadboard. To connect to Arduino, I use jumper cables.

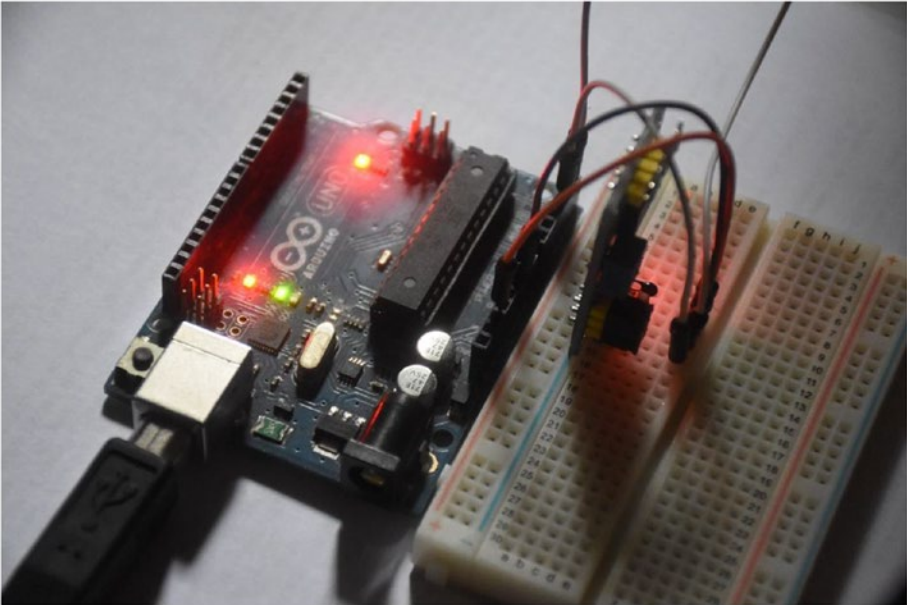


Figure 4-23. Wiring for I2C demo with WRA

Creating a UWP Project

Using Visual Studio 2015, you can create a new project with a Blank App (Universal Windows) template. After that, install Windows Remote Arduino library. Then, you also should configure the Package.appxmanifest file to additional capability to access USB. You can write these bold codes as follows:

```
<Capabilities>
  <Capability Name="internetClient" />
  <DeviceCapability Name="serialcommunication">
    <Device Id="any">
      <Function Type="name:serialPort" />
    </Device>
  </DeviceCapability>
</Capabilities>
```

Save this program. The next step is to write a .NET program.

Please install a StandardFirmata sketch program into the Arduino board before we continue our project.

Writing UWP Program

The first step to build a UWP program is to build the application UI. We just put Textblock to show information about Thermistor, Photo-voltaic and Potentiometer values. You can see the application UI in Figure 4-24.

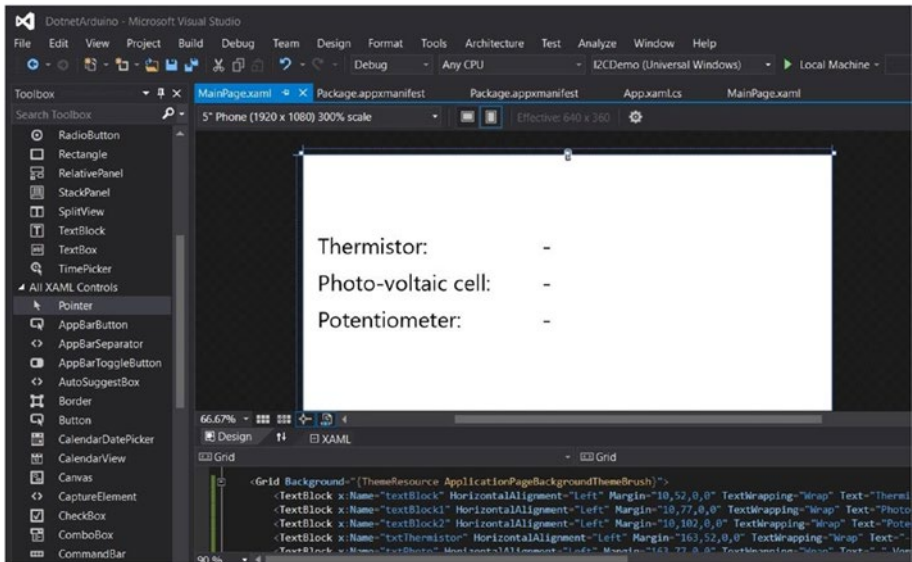


Figure 4-24. UI for I2C application

The following is an XAML script for our application UI on the MainPage.xaml file:

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <TextBlock x:Name="textBlock" HorizontalAlignment="Left"
    Margin="10,52,0,0" TextWrapping="Wrap" Text="Thermistor:"
    VerticalAlignment="Top"/>
  <TextBlock x:Name="textBlock1" HorizontalAlignment="Left"
    Margin="10,77,0,0" TextWrapping="Wrap" Text="Photo-voltaic cell:"
    VerticalAlignment="Top"/>
  <TextBlock x:Name="textBlock2" HorizontalAlignment="Left"
    Margin="10,102,0,0" TextWrapping="Wrap" Text="Potentiometer:"
    VerticalAlignment="Top" RenderTransformOrigin="-1.593,-0.469"/>
  <TextBlock x:Name="txtThermistor" HorizontalAlignment="Left"
    Margin="163,52,0,0" TextWrapping="Wrap" Text="-"
    VerticalAlignment="Top"/>
  <TextBlock x:Name="txtPhoto" HorizontalAlignment="Left"
    Margin="163,77,0,0" TextWrapping="Wrap" Text="-"
    VerticalAlignment="Top"/>
</Grid>
```

```
<TextBlock x:Name="txtPot" HorizontalAlignment="Left"
Margin="163,102,0,0" TextWrapping="Wrap" Text="-"
VerticalAlignment="Top"/>
```

```
</Grid>
```

The next step is to modify our codes in MainPage.xaml.cs. First, we add the required libraries which are used in the application.

```
using System.Text;
using Windows.ApplicationModel.Core;
using Microsoft.Maker.Serial;
using Microsoft.Maker.RemoteWiring;
```

Then, we add variables for Arduino processing and the PCF8591 AD/DA module. We also define the variable, called `i2cReading`, as Queue collection, which is used to store reading data from the module. In the reading process, we implement an asynchronous method, so we use a timer via `DispatcherTimer` object to read sensor data in a continuous time.

```
private UsbSerial connection;
private RemoteDevice arduino;
private const byte NUM_DIGITAL_PINS = 14; // Arduino Uno
private const byte SDA = 4;
private const byte SCL = 5;
private const byte PCF8591 = (0x90 >> 1); // Device address
private const byte PCF8591_ADC_CHO = 0x40; // thermistor
private const byte PCF8591_ADC_CH1 = 0x41; // photo-voltaic cell
private const byte PCF8591_ADC_CH2 = 0x42;
private const byte PCF8591_ADC_CH3 = 0x43; // potentiometer
private int index = 0;
private Queue<int> i2cReading = new Queue<int>();
private bool isReading;

private DispatcherTimer timer;
```

In constructor, we initialize our WRA library. We set all pins which are used for I2C application. Change “VID_2341” and “PID_0043” values for your Arduino USB, which we have learned.

```
public MainPage()
{
    this.InitializeComponent();
    this.Unloaded += MainPage_Unloaded;

    InitWRA();
}
```

```

private void MainPage_Unloaded(object sender, RoutedEventArgs e)
{
    connection.end();
    arduino.Dispose();
}
private void InitWRA()
{
    connection = new UsbSerial("VID_2341", "PID_0043");
    arduino = new RemoteDevice(connection);

    connection.ConnectionEstablished += Connection_ConnectionEstablished;
    connection.begin(57600, SerialConfig.SERIAL_8N1);
}

```

Once the application is connected to USB, we activate our timer to start to read data from a PCF8591 AD/DA module. I2C data is coming from I2cReplyEvent event after we call ReadADC() method.

```

private void Connection_ConnectionEstablished()
{
    System.Diagnostics.Debug.WriteLine("Connected");
    arduino.pinMode(NUM_DIGITAL_PINS + SDA, PinMode.I2C);
    arduino.pinMode(NUM_DIGITAL_PINS + SCL, PinMode.I2C);

    index = 0;

    timer = new DispatcherTimer();
    timer.Interval = TimeSpan.FromMilliseconds(2000);
    timer.Tick += Timer_Tick;
    timer.Start();
}

private void I2c_I2cReplyEvent(byte address_, byte reg_, Windows.Storage.
Streams.DataReader response)
{
    byte[] data = new byte[2];
    response.ReadBytes(data);
    int curr = i2cReading.Dequeue();
    UpdateData(curr, data[1]);
    System.Diagnostics.Debug.WriteLine("" + Convert.ToString(address_) + "-"
+ curr.ToString() + ": " + BitConverter.ToString(data));
    isReading = false;
}

```

```

private void Timer_Tick(object sender, object e)
{
    if (isReading)
        return;

    isReading = true;
    switch (index)
    {
        case 0:
            i2cReading.Enqueue(index);
            System.Diagnostics.Debug.WriteLine("PCF8591_ADC_CH0");
            ReadADC(PCF8591_ADC_CH0);
            break;
        case 1:
            i2cReading.Enqueue(index);
            System.Diagnostics.Debug.WriteLine("PCF8591_ADC_CH1");
            ReadADC(PCF8591_ADC_CH1);
            break;
        case 2:
            i2cReading.Enqueue(index);
            System.Diagnostics.Debug.WriteLine("PCF8591_ADC_CH2");
            ReadADC(PCF8591_ADC_CH2);
            break;
    }
    index++;
    if (index > 2)
        index = 0;
}

```

The last step is to define `ReadADC()` methods. In this method, we enable I2C mode and send data to I2C bus to retrieve data from the module. Each reading result is stored in Queue object. To update data into UI, we define `UpdateData()` method. We implement async method because we work in asynchronous mode so we should perform it for crossing among threadings.

```

void ReadADC(byte config)
{
    arduino.I2c.enable();
    arduino.I2c.I2cReplyEvent += I2c_I2cReplyEvent;

    arduino.I2c.beginTransaction(PCF8591);
    arduino.I2c.write(config);
    arduino.I2c.endTransmission();

    arduino.I2c.requestFrom(PCF8591, 2);
}

```

```
private async void UpdateData(int index, byte value)
{
    await CoreApplication.MainView.CoreWindow.Dispatcher.RunAsync(Windows.
    UI.Core.CoreDispatcherPriority.Normal,
        () =>
        {
            switch (index)
            {
                case 0:
                    txtThermistor.Text = Convert.ToString(value);
                    break;
                case 1:
                    txtPhoto.Text = Convert.ToString(value);
                    break;
                case 2:
                    txtPot.Text = Convert.ToString(value);
                    break;
            }
        });
}
```

Save all codes.

Testing

Now you can compile the program. First, deploy the program and then run it. You should see information for Thermistor, Photo-voltaic and Potentiometer. You can see it in Figure 4-25.

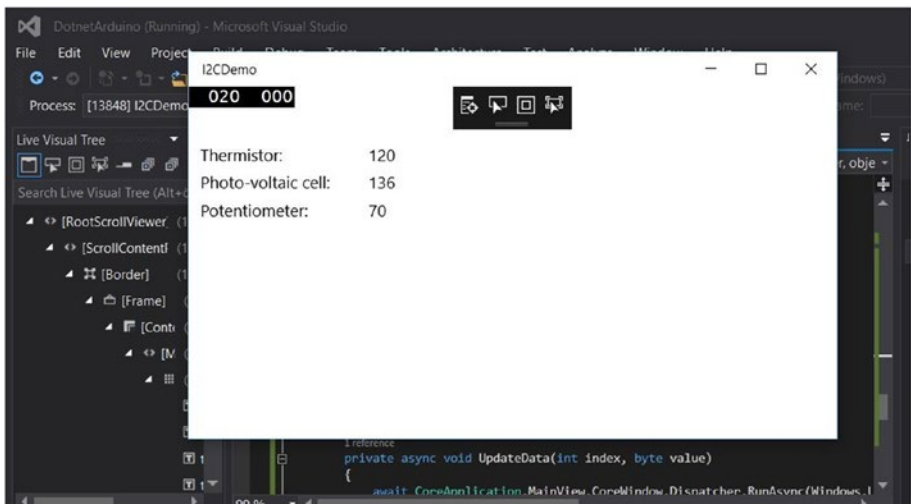


Figure 4-25. I2C application is running

Windows Remote Arduino Over Bluetooth

In the previous section, we access Arduino through Windows Remote Arduino over USB cable. In this section, we learn to access Arduino over Bluetooth. In order to work with Windows Remote Arduino over Bluetooth, we need two Bluetooth devices for Arduino and the computer. For the Arduino board, you can use any serial Bluetooth module, such as SparkFun Bluetooth Mate Silver. You can review and buy this Bluetooth module on <https://www.sparkfun.com/products/12576>. The form of SparkFun Bluetooth Mate Silver can be seen in Figure 4-26.

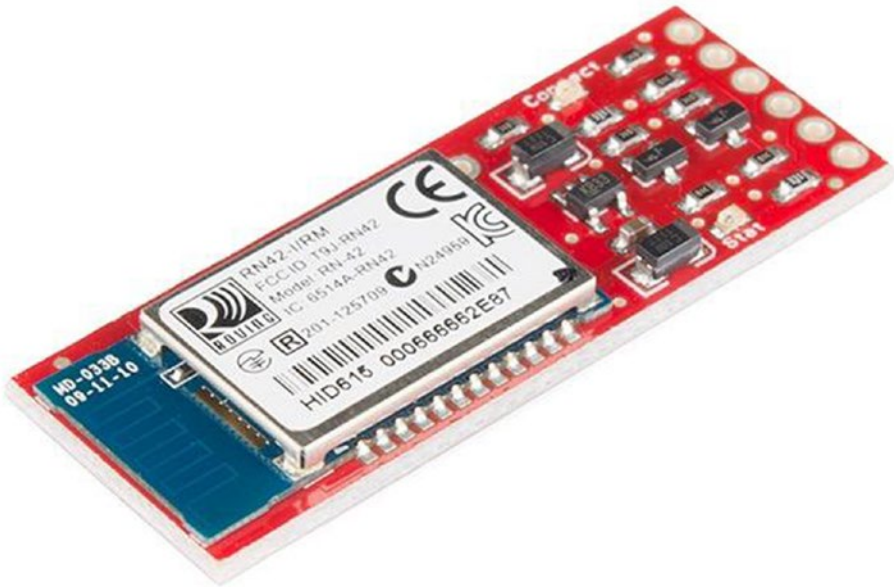


Figure 4-26. SparkFun Bluetooth Mate Silver

For a demo, I use a cheap serial Bluetooth: HC-06. This module is available in online stores such as eBay, Aliexpress, and Banggood. You can see it in Figure 4-27. Bluetooth HC-06 module has four pins: Tx, Rx, VCC, and GND.

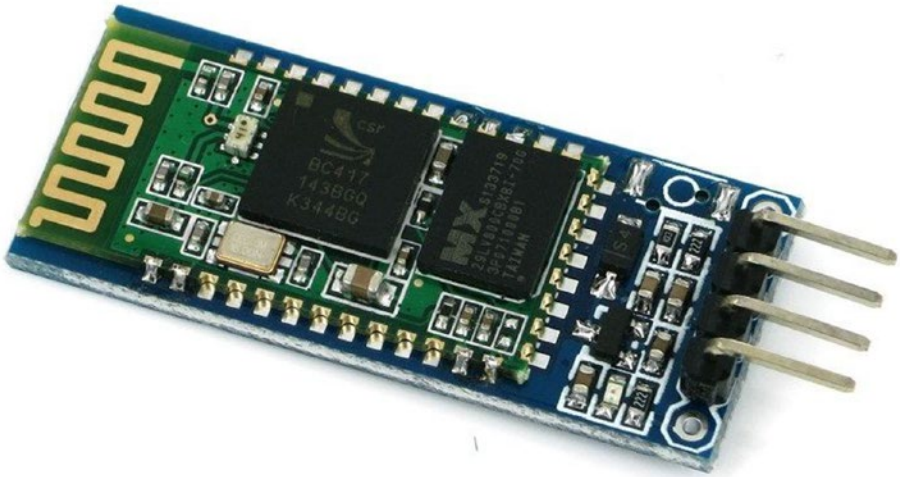


Figure 4-27. HC-06 Bluetooth module

On the computer side, we need a Bluetooth device which is compatible with the Bluetooth module from Arduino. Several computers usually have built-in Bluetooth 4.0, so you can use that. For testing, I use an external Bluetooth 4.0 USB module. You can check it on <https://www.adafruit.com/product/1327>. Just plug it into your computer. Then, the computer should recognize it.

In this section, we will use the same demo like the first demo in this chapter: lighting LEDs through UWP application. This demo will use a Bluetooth module to communicate between an Arduino board and a computer.

For testing, I use HC-06 Bluetooth for Arduino and Bluetooth 4.0 USB (CSR) for my computer. We will control LEDs to be turned on/off. Bluetooth communication between Arduino and the computer is done using Windows Remote Arduino.

Wiring for WRA with Bluetooth

Now you can connect Bluetooth and LEDs to an Arduino board. In general, Bluetooth HC-06 has four pins: Tx, Rx, VCC, and GND. So, in our wiring, it is implemented as follows:

- HC-06 Tx pin is connected to Arduino Rx pin (digital pin 0).
- HC-06 Rx pin is connected to Arduino Tx pin (digital pin 1).
- HC-06 VCC pin is connected to Arduino 3.3V pin.
- HC-06 GND pin is connected to Arduino GND pin.
- LED 1, LED 2, and LED 3 are connected to Arduino digital pin 12, 11 and 10.

You can see my wiring implementation, as shown in Figure 4-28.

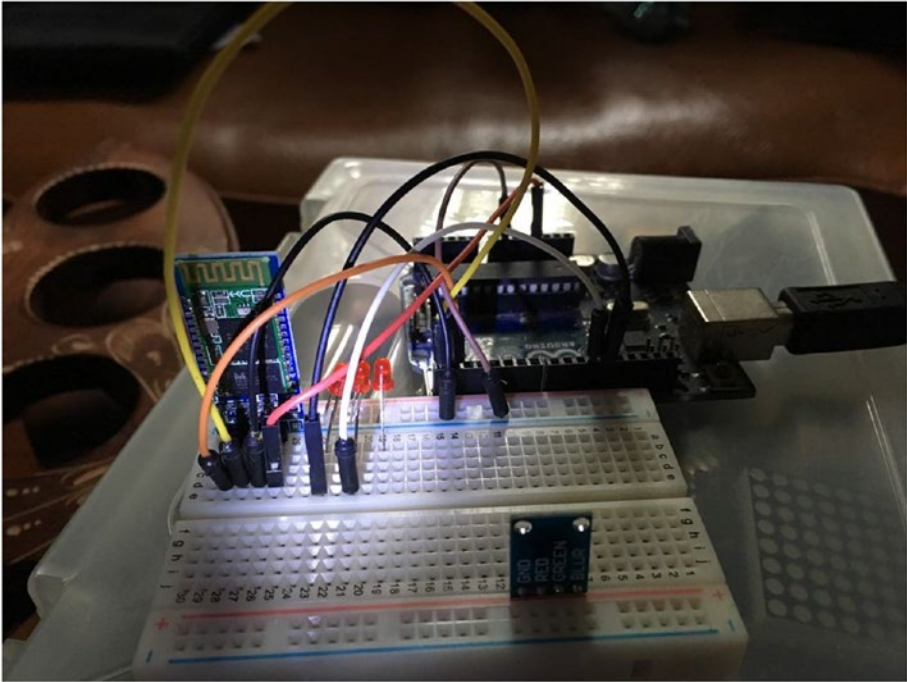


Figure 4-28. Wiring for HC-06, LED and Arduino Uno

Pairing Arduino Bluetooth and Computer

The next step is to pair Arduino Bluetooth to computer Bluetooth so they can communicate with each other. After the Arduino board is turned on, the Arduino Bluetooth module (HC-06) will broadcast Bluetooth signals. On the computer side, we should turn on the Bluetooth device, too.

On Windows 10, we can see a Manage Bluetooth device dialog in Settings. You should see our Arduino Bluetooth on this dashboard. You can see my HC-06 Bluetooth, which is recognized in my Windows 10 Desktop in Figure 4-29.

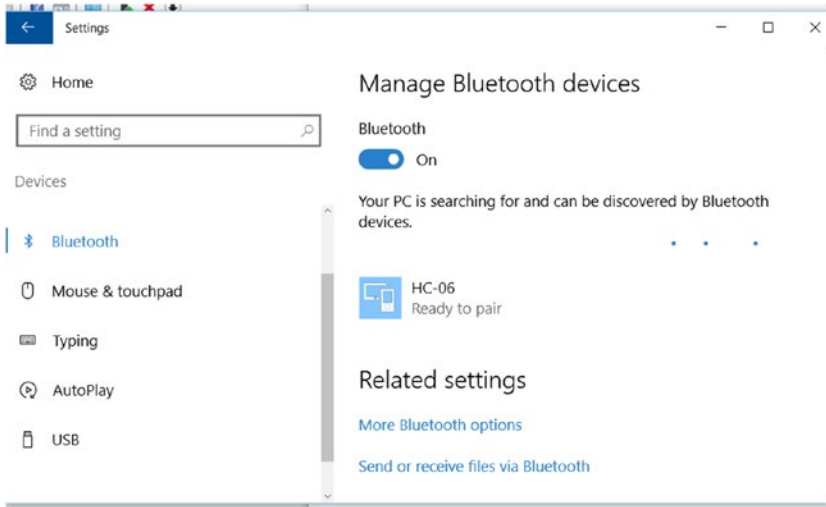


Figure 4-29. Pair Bluetooth device on Windows 10 desktop

You can pair HC-06 Bluetooth with computer Bluetooth. The default key for pairing HC-06 is 1234. After it's paired, you should see Arduino Bluetooth in Device Manager. For instance, my HC-06 is recognized as HC-06 on Device Manager, which is shown in Figure 4-30.

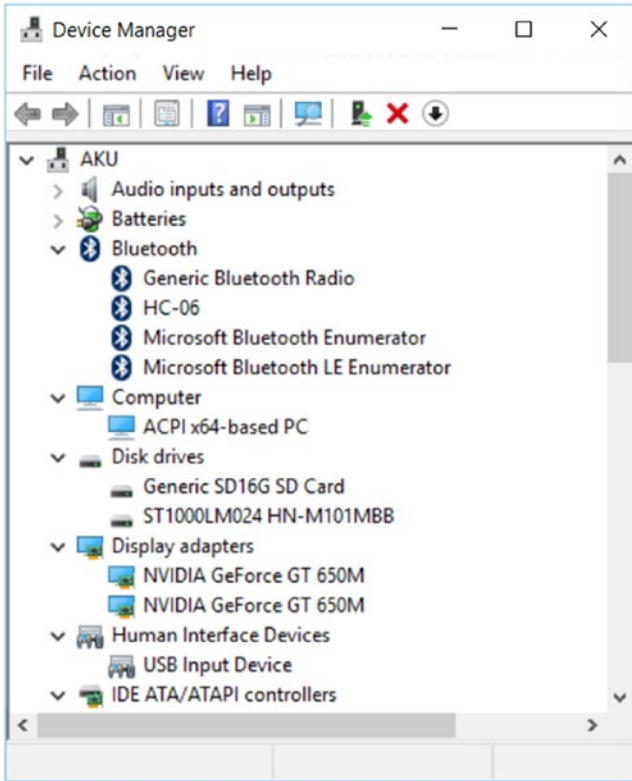


Figure 4-30. Paired Bluetooth HC-06 is shown in Device Manager

Creating a UWP Project

Using Visual Studio 2015, you can create a new project with a Blank App (Universal Windows) template. For instance, the project name is “BlinkingLedBL”. After that, install Windows Remote Arduino library. Then, you also should configure the Package.appxmanifest file to additional capability to access Bluetooth USB. You can write these bold codes as follows:

```
<Capabilities>
  <Capability Name="internetClient" />
  <DeviceCapability Name="bluetooth.rfcomm">
    <Device Id="any">
      <Function Type="name:serialPort" />
    </Device>
  </DeviceCapability>
</Capabilities>
```

Save this program. The next step is to write an Arduino and a .NET program.

Writing an Arduino Program

To work with a UWP program, you should install a Firmata Sketch program in Arduino. Select StandardFirmata. Then, flash the Firmata Sketch program into the Arduino board. Before you perform flashing Firmata Sketch, you should plug out the Bluetooth module from Arduino. Since the Bluetooth module uses Arduino UART, we can't upload the program to an Arduino board. After completing flashing, you could plug in the Bluetooth module again to an Arduino board.

Writing a UWP Program

Basically, we develop BlinkingLedBL like BlinkingLed, but this program uses Bluetooth for communicating between the Arduino board and the computer.

Build your application UI like the BlinkingLed application. Then, we modify the MainPage.xaml.cs file. We just replace Class type for connection variable. We declare connection variable as BluetoothSerial.

```
public sealed partial class MainPage : Page
{
    private BluetoothSerial connection;
    private RemoteDevice arduino;
    private const byte LED1 = 12;
    private const byte LED2 = 11;
```

In InitWRA() method, we modify for instantiating the BluetoothSerial object by passing a paired Bluetooth device name. You should change baudrate too. In this case, my Bluetooth device uses baudrate 9600.

```
private void InitWRA()
{
    connection = new BluetoothSerial("Dev B"); // HC-06
    arduino = new RemoteDevice(connection);

    connection.ConnectionEstablished += Connection_ConnectionEstablished;
    connection.ConnectionFailed += Connection_ConnectionFailed;
    connection.ConnectionLost += Connection_ConnectionLost;
    connection.begin(9600, 0);
}
}
```

You can see that the BluetoothSerial object needs a Bluetooth serial name. For instance, my HC-06 module is recognized with the name "Dev B". How to get this name? First, open the Property of the paired Bluetooth on the computer. A sample of a paired Bluetooth is shown in Figure 4-31.

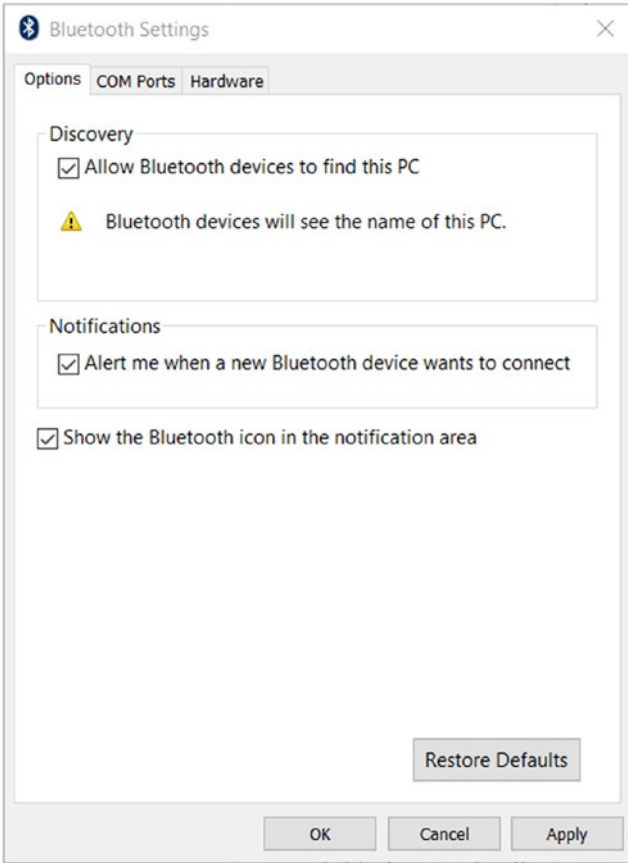


Figure 4-31. Property of paired Bluetooth on Windows 10

Click **COM Ports** property. You should see a paired Bluetooth. It shows COM incoming and outgoing. It can be seen in Figure 4-32. You can see “Dev B” for COM outgoing, which is used in our application.

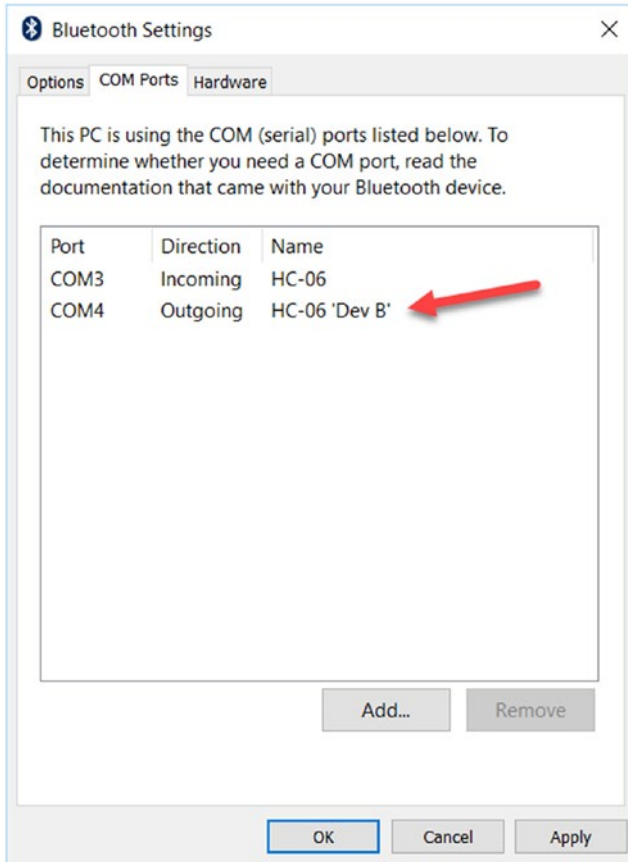


Figure 4-32. COM ports for paired Bluetooth

Testing

Now you can test the program after it's compiled and deployed. You can turn on LEDs with checking the box. You should see the lighting LED.

Summary

In this chapter, we learned how to build Windows Remote Arduino (WRA) programs on Windows Universal Platform. We started from controlling digital and analog I/O until accessing Arduino over I2C bus. We also learned how to configure WRA over Bluetooth, so we can control Arduino from a UWP programs-based Bluetooth stack. In the next chapter, we will explore how our Arduino interacts with a Cloud server.

CHAPTER 5



Building Your Own IoT Using Arduino and .NET

In general, Internet of Things (IoT) boards have limited storage to store your sensor data. We can't keep the sensor data in local storage. In this chapter, we learn how to publish our sensor data so the data can be consumed by another system. We propose web server, custom server, and cloud server to store your data and the publish it, starting with how to connect your Arduino board to an Internet network and doing sensing and publishing data. Several techniques to communicate with servers are introduced with several communication protocols. We will also learn how to interact between a .NET application and Arduino to communicate over a network.

In this chapter we'll explore the following topics:

- Introduction to Internet of Things and Arduino
- Connecting Arduino to Internet network
- Accessing Arduino over network from .NET application
- Windows Remote Arduino (WRA) over WiFi
- RF communication for Arduino
- Building LoRa network for Arduino
- Location-based application for Arduino
- Arduino and cloud server

Introduction to Internet of Things and Arduino

IoT is a recent famous technology. We build boards which are connected to an Internet network. Each board has its own purpose to connect a network. Arduino, by default, does not provide network capabilities. Otherwise, several Arduino board types such as Arduino MKR1000, Arduino UNO WiFi, and Arduino YUN have network capabilities. These boards can communicate with others over a network. You can see the illustration of how Arduino connects to a network in Figure 5-1.

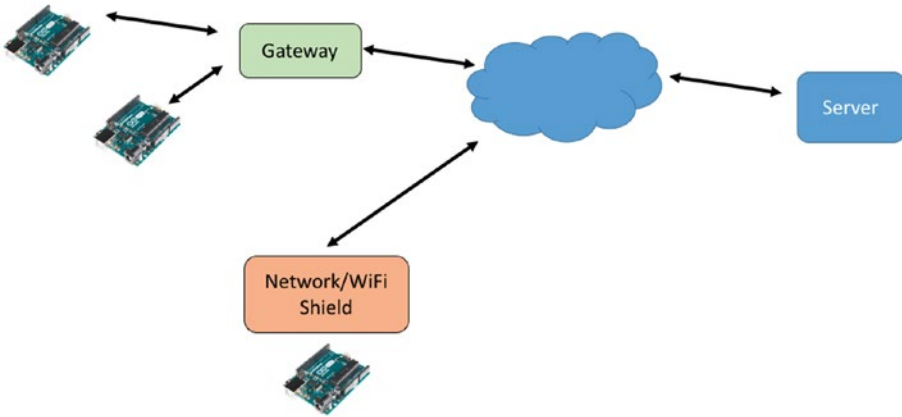


Figure 5-1. *Communication for Arduino boards*

In this chapter, I don't go into detail about IoT. You can read about a concept and architecture of IoT and how to build them. What we learn in this chapter is to explore how an Arduino board connects to another system over a network.

Network stacks such as wired and wireless networks can be used by Arduino boards. If an Arduino board doesn't have network capabilities, you can attach Arduino boards to a computer or mini-computer, such as Raspberry Pi, to be a gateway. This computer will be used to send and receive data from another system.

In the next section, we'll learn how Arduino connects to an Internet network and then send data to a network system.

Connecting Arduino to Internet Network

By default, Arduino boards don't have connectivity capability. However, some Arduino board models such as Arduino YUN, Arduino MKR1000, Arduino 101, and Arduino UNO WiFi have connectivity capability. As another approach to connect to an Internet network, we can use additional Arduino shields to enable our board to connect to an external network. In this section, we explore how an Arduino board connects to a network.

Connecting to a Wired Network

We can connect our Arduino board to a network through Ethernet. To apply this capability, we can add Ethernet shield, for instance, Arduino Ethernet shield 2 with PoE, which is shown in Figure 5-2. This shield is manufactured by Arduino SRL. You can find this shield in your local electronics store. You can buy it on the official website, <http://www.arduino.org/products/shields/arduino-ethernet-shield-2>. You also can buy the cheaper shield from a Chinese manufacturer.

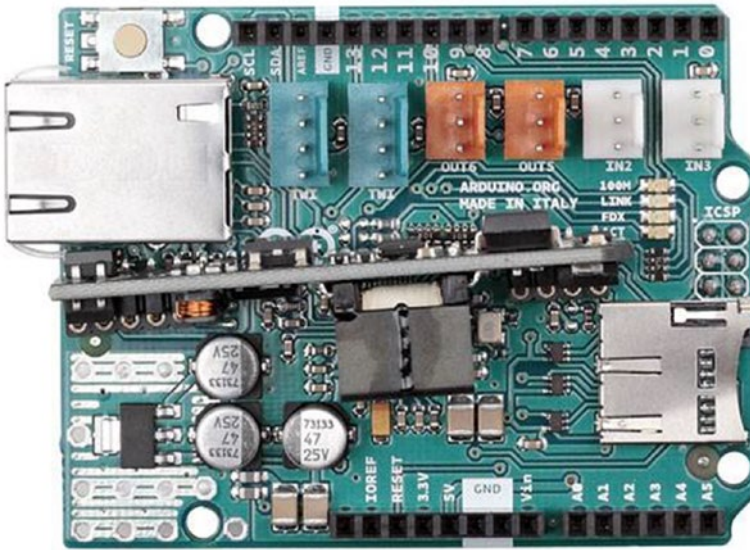


Figure 5-2. Arduino ETHERNET shield 2 WITH PoE

An Ethernet shield is easy to use. You just attach this shield on an Arduino board. You can also find this shield on the official website or buy an Ethernet shield from a Chinese manufacturer with low pricing. I have an Arduino Ethernet shield which is attached on Arduino UNO. You can see it in Figure 5-3.

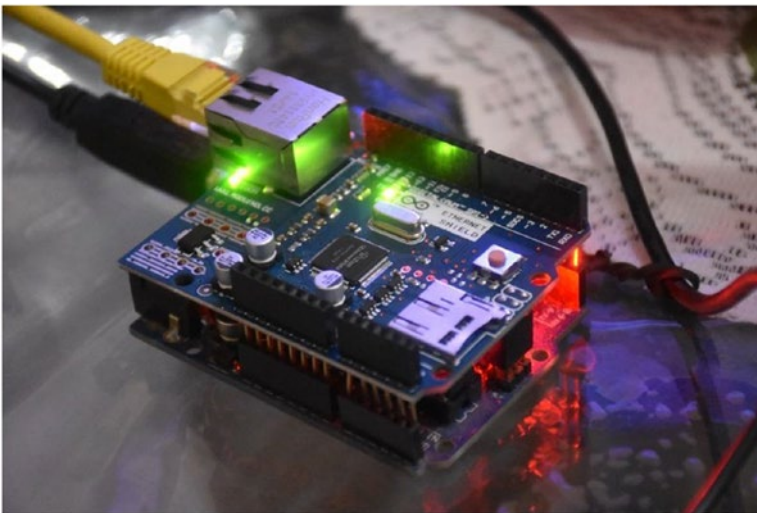


Figure 5-3. Arduino UNO and Ethernet shield

For testing, we use a sample Sketch from Arduino: WebServer. You can find it from Arduino IDE on menu File ► Examples ► Ethernet ► WebServer. After it's selected, you should should obtain the following Sketch:

```
#include <SPI.h>
#include <Ethernet.h>

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
//IPAddress ip(192, 168, 1, 177);

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(80);

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  // start the Ethernet connection and the server:
  Ethernet.begin(mac);
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        // if you've gotten to the end of the line (received a newline
        // character) and the line is blank, the http request has ended,
        // so you can send a reply
        if (c == '\n' && currentLineIsBlank) {
```

```

// send a standard http response header
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html");
client.println("Connection: close"); // the connection will be
closed after completion of the response
client.println("Refresh: 5"); // refresh the page automatically
every 5 sec
client.println();
client.println("<!DOCTYPE HTML>");
client.println("<html>");
// output the value of each analog input pin
for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
    int sensorReading = analogRead(analogChannel);
    client.print("analog input ");
    client.print(analogChannel);
    client.print(" is ");
    client.print(sensorReading);
    client.println("<br />");
}
client.println("</html>");
break;
}
if (c == '\n') {
    // you're starting a new line
    currentLineIsBlank = true;
} else if (c != '\r') {
    // you've gotten a character on the current line
    currentLineIsBlank = false;
}
}
}
// give the web browser time to receive the data
delay(1);
// close the connection:
client.stop();
Serial.println("client disconnected");
}
}
}

```

Save this program. Try to compile and upload the Sketch program to an Arduino board. Make sure your Ethernet shield is connected to a network cable.

By default, the program runs as DHCP client to retrieve an IP Address. Your network should have a DHCP server in the network. You can use a router which is connected to a network.

After running the program, open Serial Monitor to see the IP address of your Arduino. You can see my IP Address from my Arduino UNO, as shown in Figure 5-4.

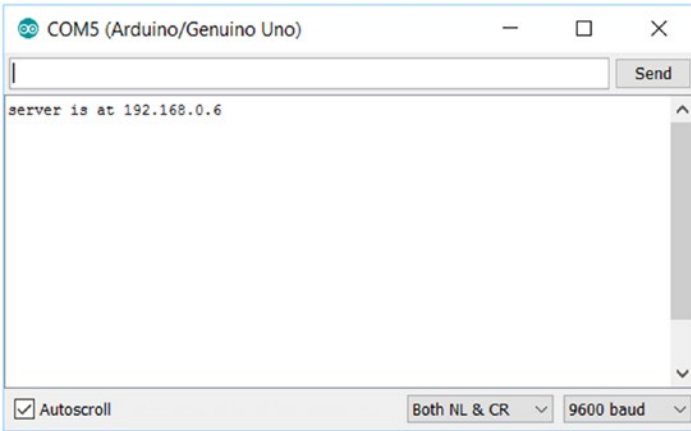


Figure 5-4. Display IP Address on Serial Monitor tool

After you know the IP Address from your Arduino board, you can open a browser and navigate to the IP Address of Arduino. If you succeed, you should see information about analog input on A0...A5. You can see the sample output in Figure 5-5.



Figure 5-5. Analog information from Arduino over browser

Connecting to a WiFi Network

Connecting an Arduino board to a wired network makes your board unable to move anywhere because the board should be connected to a network through a cable. If you want your Arduino board to be placed in any certain area, you can consider connecting your Arduino board to a WiFi network.

As I explained, by default, Arduino board does not have a WiFi module. We should add a WiFi Shield (for instance, Arduino Yún Shield), which is shown in Figure 5-6. We also could use Arduino MKR1000 to connect the existing WiFi network. Arduino MKR1000 has a built-in WiFi module. You can review this board on this site: <https://www.arduino.cc/en/Main/ArduinoMKR1000>.



Figure 5-6. *Arduino Yún Shield*

You can also use Arduino boards such as Arduino YUN and Arduino MKR1000, which are ready for WiFi connection. You can use Arduino Yun Shield for an Arduino board (<https://www.arduino.cc/en/Main/ArduinoYunShield>), as shown in Figure 5-6. For testing, I use Arduino MKR1000 to show how an Arduino board connects to the existing WiFi hotspot.

In order for Arduino MKR1000 to work with Arduino software, we should install **Arduino SAMD Boards**. You can open Boards Manager and search “Arduino SAMD Boards”. If it’s found, you can install it. You can see it in Figure 5-7. Make sure your computer is already connected to an Internet network.

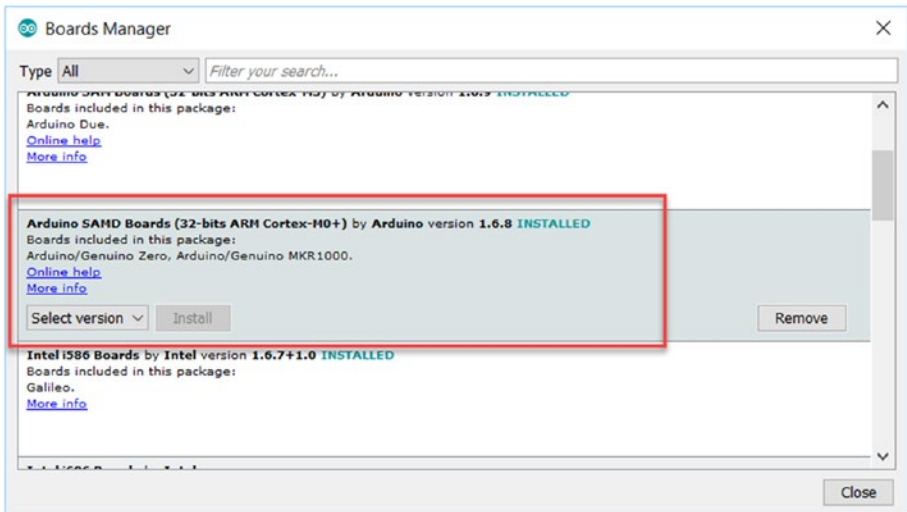


Figure 5-7. *Enable Arduino software for Arduino MKR1000*

After it's installed, you can connect Arduino MKR1000 to your computer. You can see my Arduino MKR1000 connected to my computer, as shown in Figure 5-8.

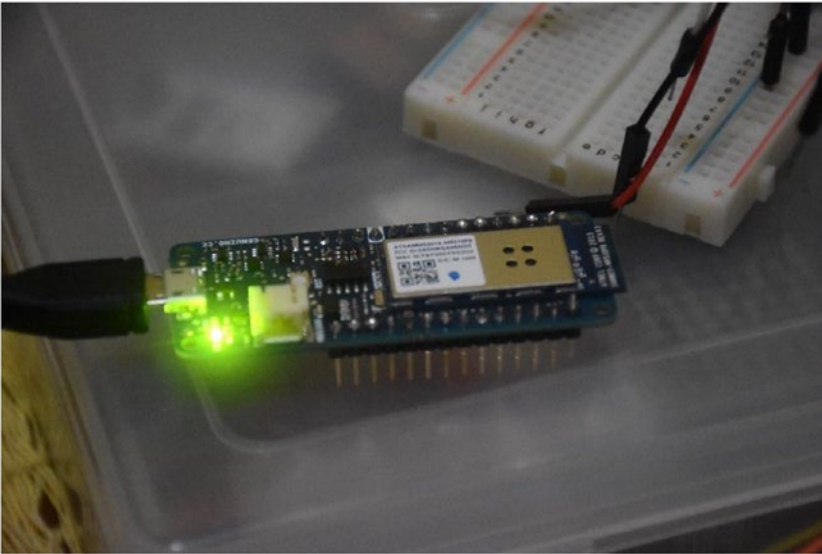


Figure 5-8. Connect Arduino MKR1000 to a computer

After Arduino MKR1000 is connected to a computer with Windows OS, you can see the board will be recognized as COMx. My Arduino MKR1000 board is detected as COM7, which is shown in Figure 5-9.

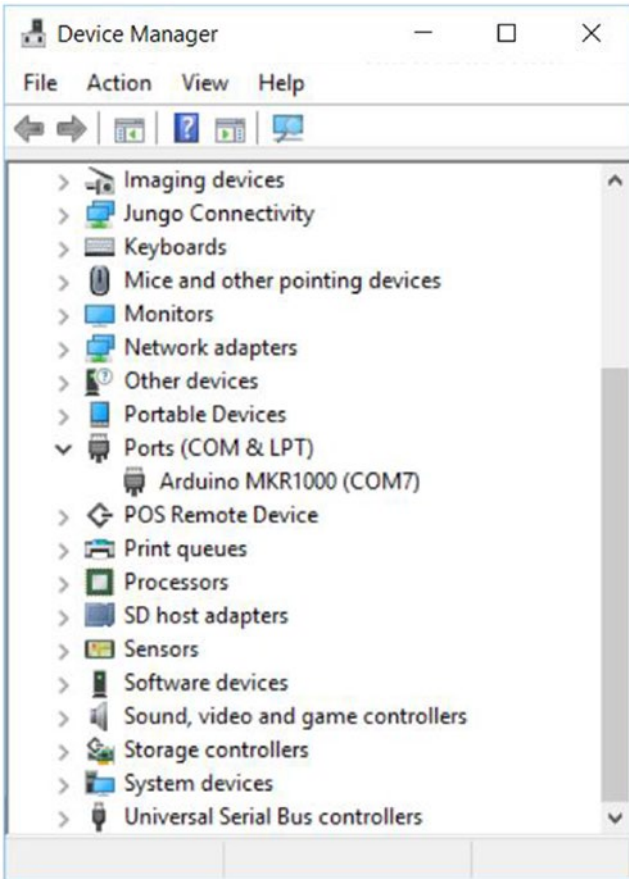


Figure 5-9. *Arduino MKR1000 is recognized as COM7 in Device Manager*

The next step is to start to build a program. In order to access a WiFi network, we need a WiFi library for the Arduino MKR1000 board. We can use WiFi101 library. You can install WiFi101 library from Library Manager. Find and install WiFi101 library. You can see this library in Figure 5-10.

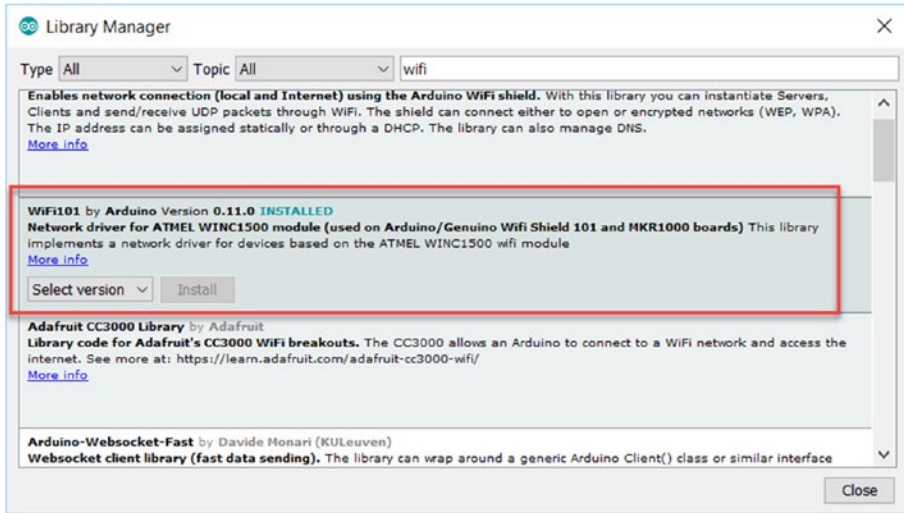


Figure 5-10. Install WiFi101 library for Arduino MKR1000

We build a Sketch program to make the Arduino MKR1000 board connect to an existing WiFi network. For instance, we connect to a WiFi hotspot with WPA security. For testing, we can use a program sample from Arduino Software, ConnectWithWPA.

After loading the Sketch program for ConnectWithWPA, you can modify values on ssid and pass for WiFi SSID and WiFi key. You can see the following Sketch program:

```
#include <SPI.h>
#include <WiFi101.h>

char ssid[] = "wifi";           // your network SSID (name)
char pass[] = "wifi_key";      // your network password
int status = WL_IDLE_STATUS;   // the Wifi radio's status

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  // check for the presence of the shield:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // don't continue:
    while (true);
  }
}
```



```

// attempt to connect to Wifi network:
while ( status != WL_CONNECTED) {
  Serial.print("Attempting to connect to WPA SSID: ");
  Serial.println(ssid);
  // Connect to WPA/WPA2 network:
  status = WiFi.begin(ssid, pass);

  // wait 10 seconds for connection:
  delay(10000);
}

// you're connected now, so print out the data:
Serial.print("You're connected to the network");
printCurrentNet();
printWifiData();
}

void loop() {
  // check the network connection once every 10 seconds:
  delay(10000);
  printCurrentNet();
}

void printWifiData() {
  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
  Serial.println(ip);

  // print your MAC address:
  byte mac[6];
  WiFi.macAddress(mac);
  Serial.print("MAC address: ");
  Serial.print(mac[5], HEX);
  Serial.print(":");
  Serial.print(mac[4], HEX);
  Serial.print(":");
  Serial.print(mac[3], HEX);
  Serial.print(":");
  Serial.print(mac[2], HEX);
  Serial.print(":");
  Serial.print(mac[1], HEX);
  Serial.print(":");
  Serial.println(mac[0], HEX);
}

```

```

void printCurrentNet() {
    // print the SSID of the network you're attached to:
    Serial.print("SSID: ");
    Serial.println(WiFi.SSID());

    // print the MAC address of the router you're attached to:
    byte bssid[6];
    WiFi.BSSID(bssid);
    Serial.print("BSSID: ");
    Serial.print(bssid[5], HEX);
    Serial.print(":");
    Serial.print(bssid[4], HEX);
    Serial.print(":");
    Serial.print(bssid[3], HEX);
    Serial.print(":");
    Serial.print(bssid[2], HEX);
    Serial.print(":");
    Serial.print(bssid[1], HEX);
    Serial.print(":");
    Serial.println(bssid[0], HEX);

    // print the received signal strength:
    long rssi = WiFi.RSSI();
    Serial.print("signal strength (RSSI):");
    Serial.println(rssi);

    // print the encryption type:
    byte encryption = WiFi.encryptionType();
    Serial.print("Encryption Type:");
    Serial.println(encryption, HEX);
    Serial.println();
}

```

Save the program. Now you can compile and upload the program to Arduino MKR1000. Open the Serial Monitor tool to see the IP Address of Arduino MKR1000. You can see my board output sample in [Figure 5-11](#).

```

COM9 (Arduino/Genuino MKR1000)
Attempting to connect to WPA SSID: b003f6
You're connected to the networkSSID: b003f6
BSSID: 60:2:92:66:5F:D2
signal strength (RSSI):-72
Encryption Type:2

IP Address: 192.168.0.10
192.168.0.10
MAC address: F8:F0:5:F5:D2:2
SSID: b003f6
  
```

Figure 5-11. Program output on Serial Monitor

If you have problems on Arduino MKR1000, you may update the WiFi module on Arduino MKR1000. You can follow this instruction to update the Arduino MKR1000 firmware: <https://www.arduino.cc/en/Tutorial/FirmwareUpdater>.

Accessing Arduino over a Network from .NET Application

In the previous section, we have learned how to make our Arduino boards connect to an existing network through Ethernet and a WiFi module. Now we try to build a .NET application to access Arduino boards over air.

Our scenario is to build a .NET application (UWP app) to control LED through WiFi. We use Arduino MKR1000 as a development board. To build our lab, we need the following devices:

- Arduino MKR1000
- Three LEDs
- Computer with installed Windows and Visual Studio 2015 or later

We will turn on/off LEDs through WiFi from a UWP application. We apply HTTP GET to turn on/off our LEDs. You can see our HTTP GET URL mapping for demo in Table 5-1. We implement this URL mapping to our demo.

Table 5-1. HTTP Get requests for turning on/off LEDs

URL	Note
http://<arduino IP>/gpio1/1	Turn on LED 1
http://<arduino IP>/gpio1/0	Turn off LED 1
http://<arduino IP>/gpio2/1	Turn on LED 2
http://<arduino IP>/gpio2/0	Turn off LED 2
http://<arduino IP>/gpio3/1	Turn on LED 3
http://<arduino IP>/gpio3/0	Turn off LED 3

To implement our program, we will build a sketch program and a UWP application. We need a sketch program in order to interact with an external application such as a UWP application. The sketch program will listen for incoming commands (HTTP GET) to be executed. Furthermore, a UWP application is a client app which sends commands to Arduino to perform something.

Let's start to develop.

Wiring

The following is the demo wiring:

- LED 1 is connected to digital pin 5 of Arduino MKR1000.
- LED 2 is connected to digital pin 4 of Arduino MKR1000.
- LED 3 is connected to digital pin 3 of Arduino MKR1000.
- All LED GND pins are connected to GND pin on the board.

If you're worrying about your LEDs, you can put a resistor on each LED to prevent a high voltage. You can see the wiring in Figure 5-12.

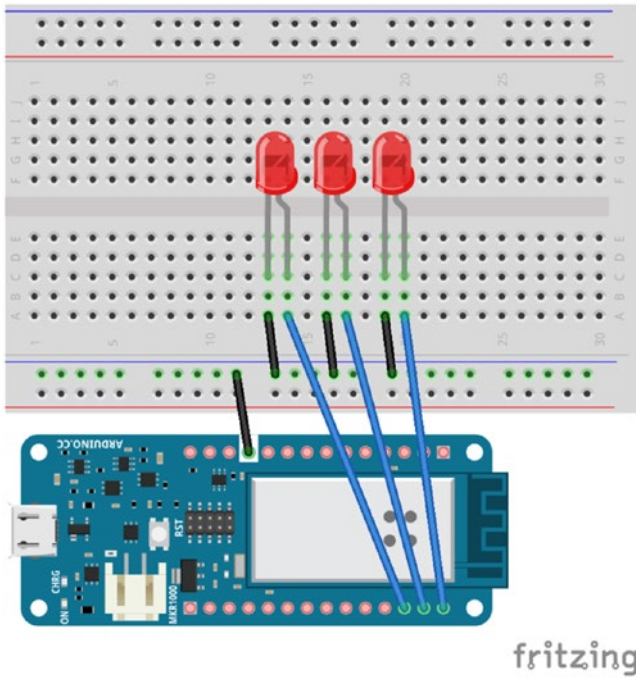


Figure 5-12. Wiring for LEDs and Arduino MKR1000

You can see my wiring implementation, as shown in Figure 5-13. Three LEDs are attached to an Arduino MKR1000 board.

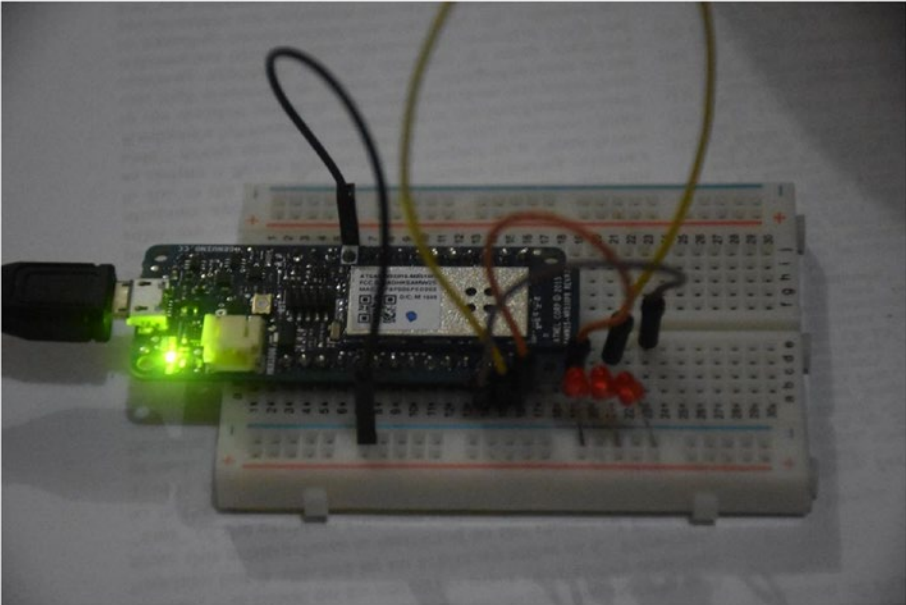


Figure 5-13. Wiring implementation for LEDs and MKR1000

Building a Sketch Program

We build a Sketch program in order to receive commands to turn on/off LEDs based on URL mapping, which is described in Table 5-1. You can write the scripts in Arduino software:

```
#include <WiFi101.h>

int led1 = 5;
int led2 = 4;
int led3 = 3;

const char* ssid = "ssid";
const char* password = "ssid_key";
int status = WL_IDLE_STATUS;

WiFiServer server(80);

void setup() {
  Serial.begin(9600);
  delay(10);
```

```

// prepare GPIO5
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
pinMode(led3, OUTPUT);
digitalWrite(led1, 0);
digitalWrite(led2, 0);
digitalWrite(led3, 0);

// Connect to WiFi network
while (status != WL_CONNECTED) {
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);
  status = WiFi.begin(ssid, password);

  // wait 10 seconds for connection:
  delay(10000);
}
Serial.println("");
Serial.println("WiFi connected");

// Start the server
server.begin();
Serial.println("Server started");

// Print the IP address
char ips[24];
IPAddress ip = WiFi.localIP();
sprintf(ips, "%d.%d.%d.%d", ip[0], ip[1], ip[2], ip[3]);
Serial.println(ips);
}

void loop() {
  // Check if a client has connected
  WiFiClient client = server.available();
  if (!client) {
    return;
  }

  // Wait until the client sends some data
  Serial.println("new client");
  while(!client.available()){
    delay(1);
  }

  // Read the first line of the request
  String req = client.readStringUntil('\r');
  Serial.println(req);
  client.flush();
}

```

```

// Match the request
int val1 = 0;
int val2 = 0;
int val3 = 0;
int ledreq = 0;
if (req.indexOf("/gpio1/0") != -1) {
    val1 = 0;
    ledreq = 1;
}
else if (req.indexOf("/gpio1/1") != -1) {
    val1 = 1;
    ledreq = 1;
}
else if (req.indexOf("/gpio2/0") != -1) {
    val2 = 0;
    ledreq = 2;
}
else if (req.indexOf("/gpio2/1") != -1) {
    val2 = 1;
    ledreq = 2;
}
else if (req.indexOf("/gpio3/0") != -1) {
    val3 = 0;
    ledreq = 3;
}
else if (req.indexOf("/gpio3/1") != -1) {
    val3 = 1;
    ledreq = 3;
}
else {
    Serial.println("invalid request");
    client.stop();
    return;
}

// Set GPIO2 according to the request
if(ledreq==1)
    digitalWrite(led1, val1);
if(ledreq==2)
    digitalWrite(led2, val2);
if(ledreq==3)
    digitalWrite(led3, val3);

client.flush();

// Prepare the response
String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE
HTML>\r\n<html>\r\n";

```



```

if(ledreq==1) {
    s += "LED1 is ";
    s += (val1)? "ON": "OFF";
}else if(ledreq==2) {
    s += "LED2 is ";
    s += (val2)? "ON": "OFF";
}else if(ledreq==3) {
    s += "LED3 is ";
    s += (val3)? "ON": "OFF";
}
s += "</html>\n";

// Send the response to the client
client.print(s);
delay(1);
client.stop();
Serial.println("Client disconnected");
}

```

You should change `ssid` and `ssid_key` for SSID and SSID key. Save these scripts as “IoTDemo”. Try to compile and upload a Sketch program to Arduino MKR1000. If it’s done, you can open a Serial Monitor tool to see the IP Address of Arduino MKR1000. You can see my IP Address of the board in Figure 5-14.

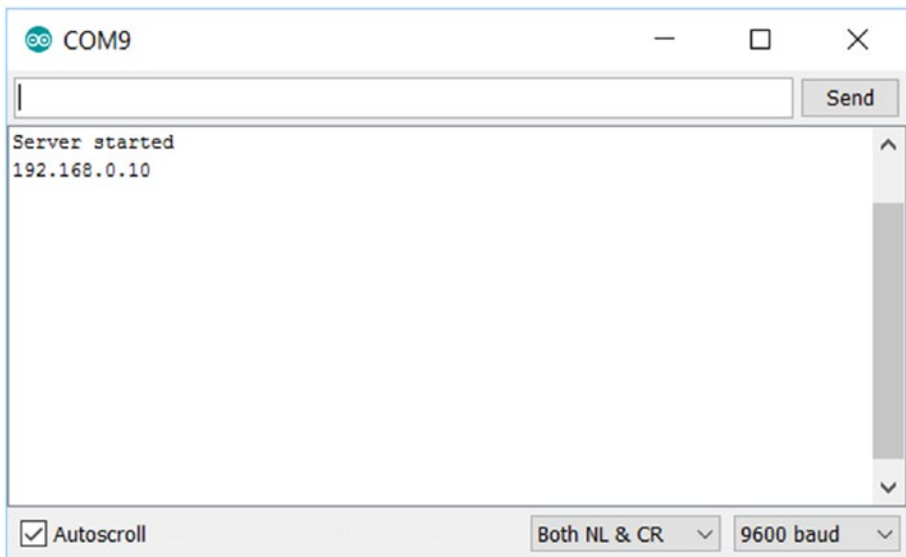


Figure 5-14. Serial monitor tool shows IP Address for Arduino MKR1000

Now you can test by opening a browser. Navigate to `http://<server>/gpio1/1` to turn on LED 1. You should change `<server>` to the IP Address of Arduino MKR1000.

If you succeed, you should see a response from Arduino MKR1000. For a sample, you can see a response from my Arduino, which is shown in Figure 5-15.



Figure 5-15. Response from Arduino on a browser

You can see your lighting LED for LED 1 in Figure 5-16 while you're sending a command to turn on LED 1.

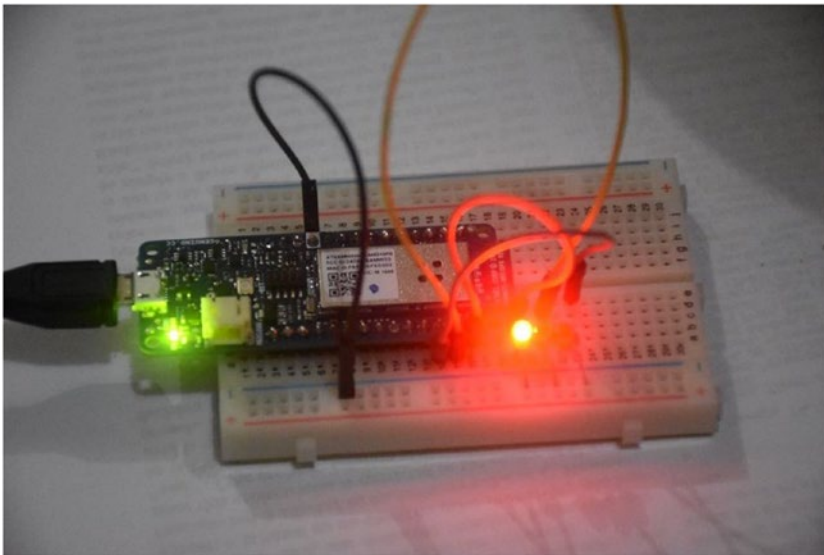


Figure 5-16. LED 1 is lighting

If you open Serial Monitor tool from Arduino, you should see the requests on Arduino MKR1000. You can see it's a program output, which is shown in Figure 5-17.

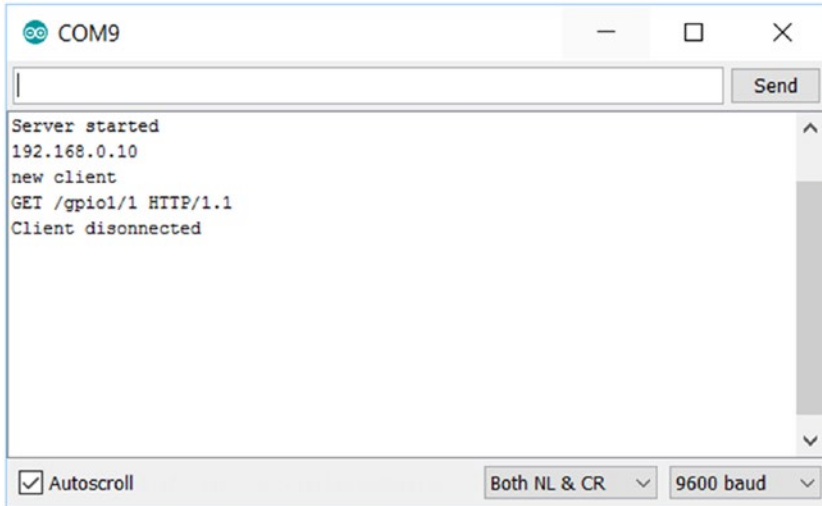


Figure 5-17. Response from Arduino on Serial monitor tool

Building a UWP Application

In this section, we develop a UWP application. This program will work as a “client app” which sends commands to Arduino to turn on/off LEDs. Our sketch program on Arduino works as a web server which receives HTTP GET commands.

Create a UWP project with project name “WiFiLed”. Then build the following UI. You can see our UI, as shown in Figure 5-18.

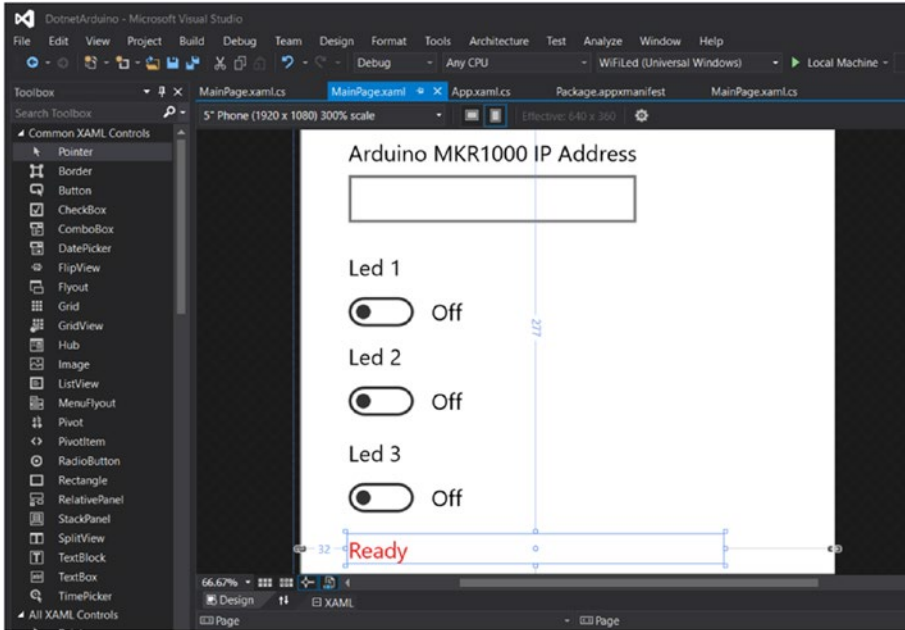


Figure 5-18. Design UI for WiFiLed application

We put TextBox and ToggleSwitch on a UI canvas. You can write the following scripts to build our UI:

```
<Page
  x:Class="WiFiLed.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:WiFiLed"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d">
  <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <TextBox x:Name="txtIP" HorizontalAlignment="Left"
      Margin="32,35,0,0" TextWrapping="Wrap" Text=""
      VerticalAlignment="Top" Width="194"/>
    <TextBlock x:Name="textBlock" HorizontalAlignment="Left"
      Margin="32,10,0,0" TextWrapping="Wrap" Text="Arduino MKR1000 IP
      Address" VerticalAlignment="Top"/>
    <ToggleSwitch x:Name="toggleLed1" Header="Led 1"
      HorizontalAlignment="Left" Margin="32,87,0,0"
      VerticalAlignment="Top" Width="94" Toggled="ToggleLed1"/>
```

```

<ToggleSwitch x:Name="toggleLed2" Header="Led 2"
HorizontalAlignment="Left" Margin="32,147,0,0"
VerticalAlignment="Top" Toggled="ToggleLed2"/>
<ToggleSwitch x:Name="toggleLed3" Header="Led 3"
HorizontalAlignment="Left" Margin="32,212,0,0"
VerticalAlignment="Top" Toggled="ToggleLed3"/>
<TextBlock x:Name="txtStatus" HorizontalAlignment="Left"
Margin="32,277,0,0" TextWrapping="Wrap" Text="Ready"
VerticalAlignment="Top" Width="253" FocusVisualPrimaryBrush="#FFD71C
1C" Foreground="#FFEE2121"/>

</Grid>
</Page>

```

Now we work on `MainPage.xaml.cs`. First, we add our requirement namespace for our program.

```
using Windows.Web.Http;
```

If you see XAML scripts, we pass a method on a `Toggled` event. We build three methods: `ToggleLed1`, `ToggleLed2`, and `ToggleLed3`. These methods are to listen to the LED state. If they're toggled, we turn on the LED. The following is the implementation of these methods:

```

private void ToggleLed1(object sender, RoutedEventArgs e)
{
    string svr = txtIP.Text;
    string url = string.Format("http://{0}/gpio1/0", svr);
    int state = 0;

    if(toggleLed1.IsOn)
    {
        url = string.Format("http://{0}/gpio1/1", svr);
        state = 1;
    }

    SendCommand(url);
    UpdateStatus(1, state);
}

private void ToggleLed2(object sender, RoutedEventArgs e)
{
    string svr = txtIP.Text;
    string url = string.Format("http://{0}/gpio2/0", svr);
    int state = 0;

```

```

        if (toggleLed2.IsOn)
        {
            url = string.Format("http://{0}/gpio2/1", svr);
            state = 1;
        }

        SendCommand(url);
        UpdateStatus(2, state);
    }

private void ToggleLed3(object sender, RoutedEventArgs e)
{
    string svr = txtIP.Text;
    string url = string.Format("http://{0}/gpio3/0", svr);
    int state = 0;

    if (toggleLed3.IsOn)
    {
        url = string.Format("http://{0}/gpio3/1", svr);
        state = 1;
    }

    SendCommand(url);
    UpdateStatus(3, state);
}

```

As you can see, we call each Toggled method `SendCommand()` to send data to Arduino MKR1000 and `UpdateStatus()` to update our UI. The following is our implementation:

```

private void UpdateStatus(int led, int state)
{
    if (state == 1)
        txtStatus.Text = string.Format("LED {0} is ON", led);
    else
        txtStatus.Text = string.Format("LED {0} is OFF", led);
}

private async void SendCommand(string url)
{
    HttpClient httpClient = new HttpClient();
    var headers = httpClient.DefaultRequestHeaders;

    Uri requestUri = new Uri(url);
    HttpResponseMessage httpResponse = new HttpResponseMessage();

    try
    {
        //Send the GET request
    }
}

```

```

    httpResponse = await httpClient.GetAsync(requestUri,HttpCompletion
    Option.ResponseHeadersRead);
    httpResponse.EnsureSuccessStatusCode();

}
catch (Exception ex)
{
    System.Diagnostics.Debug.WriteLine(ex.Message);
}
}

```

Save all programs. You can also add additional capabilities on a UWP application. Add an **internetClient** feature in our application.

```

<Capabilities>
  <Capability Name="internetClient" />
</Capabilities>

```

Testing

Save your UWP project. Then, try to compile and run the program. Please fill in the IP Address of Arduino MKR1000. Now you can turn on/off LEDs by clicking the toggled controls.

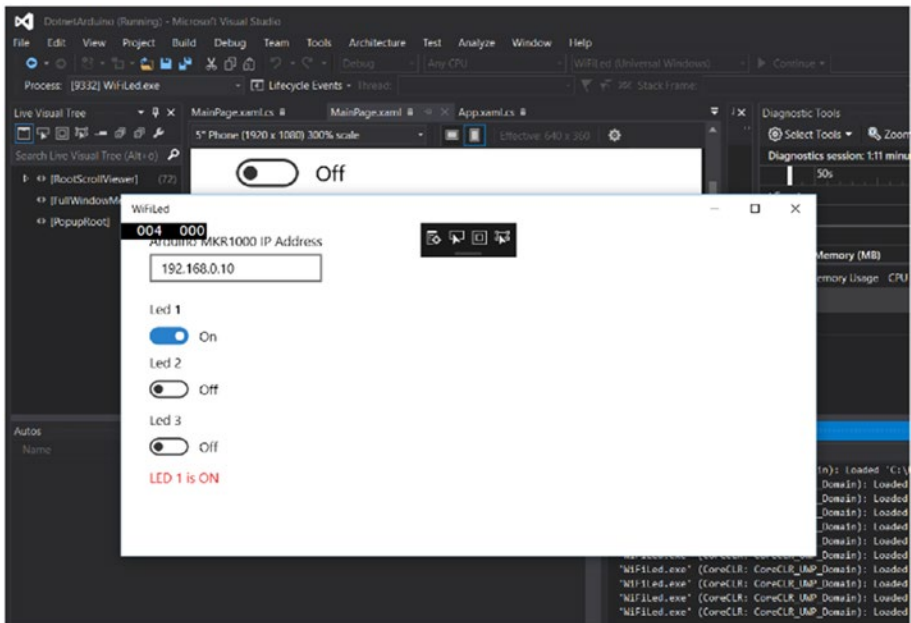


Figure 5-19. WiFiLed application is running

Windows Remote Arduino (WRA) over WiFi

In Chapter 3, we learned how to work with Windows Remote Arduino (WRA) through USB and Bluetooth. In this section, we try to build a UWP application to access Arduino through Windows Remote Arduino over WiFi.

For testing, we use an Arduino MKR1000 board. We will use the same problem from the previous section: we turn on/off LEDs over WiFi.

Let's start!

Configure Arduino for WRA over WiFi

To enable our Arduino board to be controlled over WiFi, you should deploy a StandardFirmataWiFi Sketch program into the board. You can find it from menu File ► Examples ► Firmata ► StandardFirmataWiFi. Then, you should see the sketch program, as shown in Figure 5-20.

```

boards:
  - Arduino Due or Zero: (D5, D7, D10)
  - Arduino Mega: (D5, D7, D10, D50, D52, D53)
*/

#include <Servo.h>
#include <Wire.h>
#include <Firmata.h>

/*
 * Uncomment the #define SERIAL_DEBUG line below to receive serial
 * connection that may help in the event of connection issues. If
 * executing this sketch until the Serial console is opened.
 */
// #define SERIAL_DEBUG
#include "utility/firmataDebug.h"

```

Arduino/Genuino MKR1000 on COM9

Figure 5-20. Sketch program for StandardFirmataWiFi

The next step is to configure WiFi settings for Arduino. We configure it on `wifiConfig.h`. Since we use Arduino MKR1000, you should uncomment for `WIFI_101`. You should change SSID and SSID key. The default port for server is 3030. You can change it if you want.

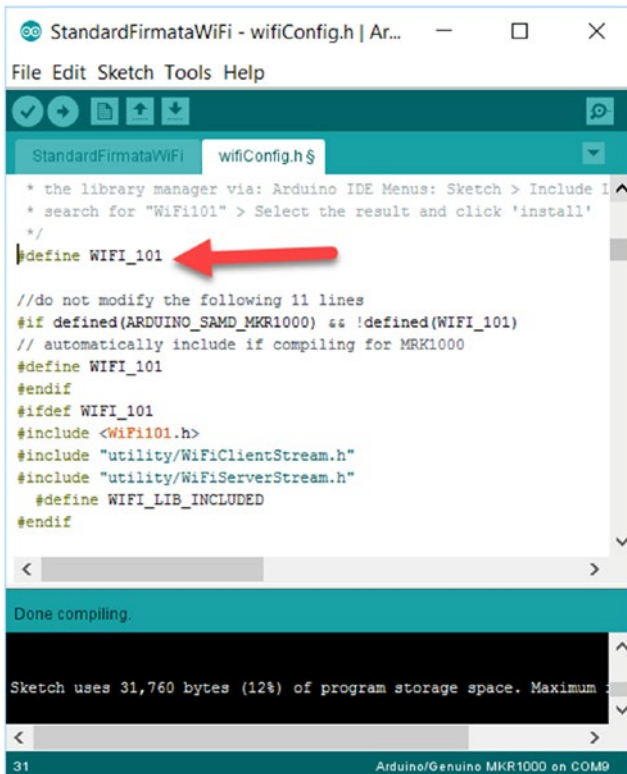


Figure 5-21. Configure WiFi for StandardFirmataWiFi sketch

If you finish configuring WiFi settings, you can compile and upload a Sketch program into Arduino MKR1000. Now your board is ready to receive commands from a UWP application.

Building a UWP Application

Now you can create a UWP application called “WiFiWRA” using Visual Studio. Install WRA through Nuget. Read how to deploy it in Chapter 3. We build a UI for a UWP application, which is shown in Figure 5-22.

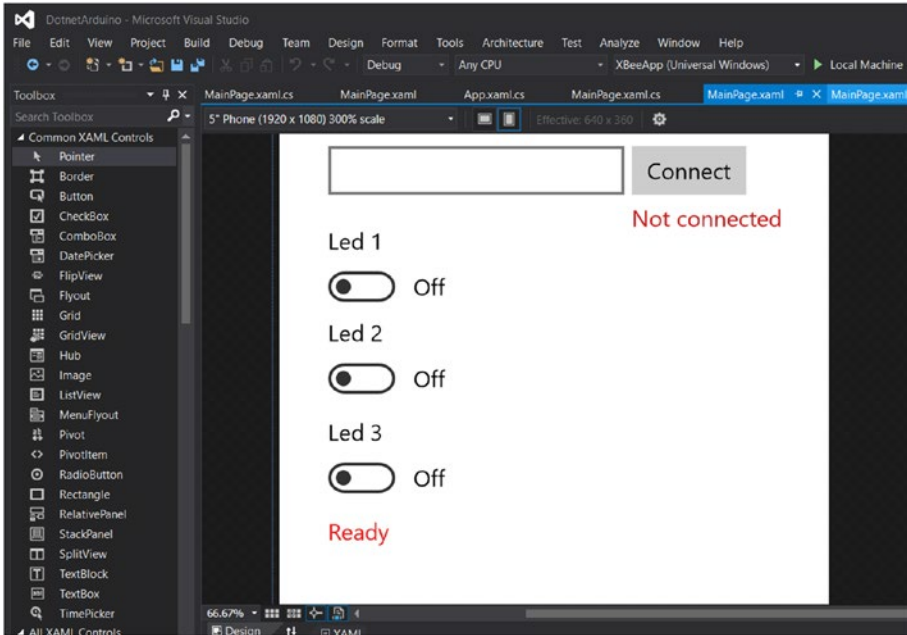


Figure 5-22. Build UI for UWP app

We build the same UI as in the previous section, but we add a button to connect Arduino over WiFi. The following are XAML scripts for WiFiWRA application:

```
<Page
  x:Class="WiFiWRA.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:WiFiWRA"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d">
  <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <TextBox x:Name="txtIP" HorizontalAlignment="Left"
      Margin="32,35,0,0" TextWrapping="Wrap" Text=""
      VerticalAlignment="Top" Width="194"/>
    <TextBlock x:Name="textBlock" HorizontalAlignment="Left"
      Margin="32,10,0,0" TextWrapping="Wrap" Text="Arduino MKR1000 IP
      Address" VerticalAlignment="Top"/>
  </Grid>
</Page>
```

```

<ToggleSwitch x:Name="toggleLed1" Header="Led 1"
HorizontalAlignment="Left" Margin="32,87,0,0"
VerticalAlignment="Top" Width="94" Toggled="ToggleLed1"/>
<ToggleSwitch x:Name="toggleLed2" Header="Led 2"
HorizontalAlignment="Left" Margin="32,147,0,0"
VerticalAlignment="Top" Toggled="ToggleLed2"/>
<ToggleSwitch x:Name="toggleLed3" Header="Led 3"
HorizontalAlignment="Left" Margin="32,212,0,0"
VerticalAlignment="Top" Toggled="ToggleLed3"/>
<TextBlock x:Name="txtStatus" HorizontalAlignment="Left"
Margin="32,277,0,0" TextWrapping="Wrap" Text="Ready"
VerticalAlignment="Top" Width="253" FocusVisualPrimaryBrush="#FFD71C
1C" Foreground="#FFEE2121"/>
<Button x:Name="button" Content="Connect" HorizontalAlignment="Left"
Margin="231,35,0,0" VerticalAlignment="Top"
Click="ConnectToServer"/>
<TextBlock x:Name="txtServerState" HorizontalAlignment="Left"
Margin="231,72,0,0" TextWrapping="Wrap" Text="Not connected"
VerticalAlignment="Top" FocusVisualPrimaryBrush="#FFCE2E2E"
Foreground="#FFCA1A1A"/>

</Grid>
</Page>

```

On MainPage.xaml.cs, we add codes to catch a clicked event on button and toggled controls. First, we add a required namespace.

```

using Microsoft.Maker.Serial;
using Microsoft.Maker.RemoteWiring;
using Windows.Networking;

```

We add several variables for three LEDs. Write these codes:

```

public sealed partial class MainPage : Page
{
    private NetworkSerial connection;
    private RemoteDevice arduino;

    private const byte LED1 = 5;
    private const byte LED2 = 4;
    private const byte LED3 = 3;
    private byte currentLed;
    private int currentState;

```

When the “Connect” button is clicked, we try to connect to Arduino MKR1000 over WiFi. After it's connected, we set the pin mode for three LEDs. The following is our code implementation:

```
private void ConnectToServer(object sender, RoutedEventArgs e)
{
    InitWRA();
}
private void InitWRA()
{
    string svr = txtIP.Text;
    connection = new NetworkSerial(new HostName(svr), 3030);
    arduino = new RemoteDevice(connection);

    arduino.DeviceConnectionLost += Arduino_DeviceConnectionLost;

    connection.ConnectionEstablished += Connection_ConnectionEstablished;
    connection.ConnectionFailed += Connection_ConnectionFailed;

    connection.begin(115200, SerialConfig.SERIAL_8N1);
}

private void Arduino_DeviceConnectionLost(string message)
{
    System.Diagnostics.Debug.WriteLine("Device is connection lost");
}

private void Connection_ConnectionFailed(string message)
{
    txtServerState.Text = "Not connected";
}

private void Connection_ConnectionEstablished()
{
    txtServerState.Text = "Connected";
    System.Diagnostics.Debug.WriteLine("Connected");

    arduino.pinMode(LED1, PinMode.OUTPUT);
    arduino.pinMode(LED2, PinMode.OUTPUT);
    arduino.pinMode(LED3, PinMode.OUTPUT);
}
}
```

On Toggled events, we execute to turn on/off LEDs by calling `Execute()` and `UpdateStatus()` methods.

```
private void ToggleLed1(object sender, RoutedEventArgs e)
{
    currentState = 0;

    if (toggleLed1.IsOn)
        currentState = 1;
        currentLed = 1;

    Execute();
    UpdateStatus(1, currentState);
}
```

```
private void ToggleLed2(object sender, RoutedEventArgs e)
{
    currentState = 0;

    if (toggleLed2.IsOn)
        currentState = 1;
        currentLed = 2;

    Execute();
    UpdateStatus(2, currentState);
}
```

```
private void ToggleLed3(object sender, RoutedEventArgs e)
{
    string svr = txtIP.Text;
    currentState = 0;

    if (toggleLed3.IsOn)
        currentState = 1;
        currentLed = 3;

    Execute();
    UpdateStatus(3, currentState);
}
```

Execute() and UpdateStatus() methods are used to execute turn on/off LEDs and update UI on a UWP application.

```
private void UpdateStatus(int led, int state)
{
    if (state == 1)
        txtStatus.Text = string.Format("LED {0} is ON", led);
    else
        txtStatus.Text = string.Format("LED {0} is OFF", led);
}
```

```
private void Execute()
{
    if (currentState == 1)
        arduino.digitalWrite(currentLed, PinState.HIGH);
    else
        arduino.digitalWrite(currentLed, PinState.LOW);

    System.Diagnostics.Debug.WriteLine("Written command to Arduino");
}
```

Save the program. You also should add capabilities on a UWP application for accessing the Internet.

```
<Capabilities>
  <Capability Name="internetClient" />
  <Capability Name="internetClientServer" />
  <Capability Name="privateNetworkClientServer" />
</Capabilities>
```

Testing

Now you can run the program. Fill in the IP Address of Arduino MKR1000. Then, click the Connect button. If it's connected, you can turn on/off LEDs by clicking toggled controls (Figure 5-23).

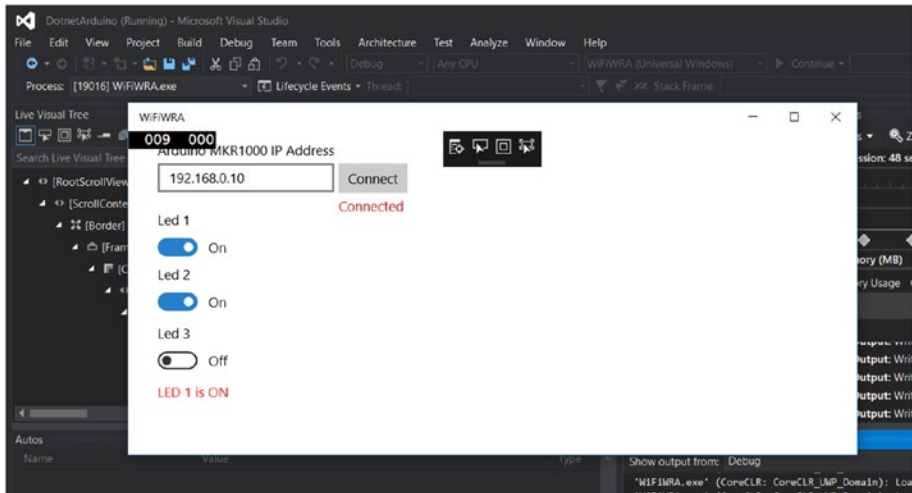


Figure 5-23. Running application for controlling LED over WRA WiFi

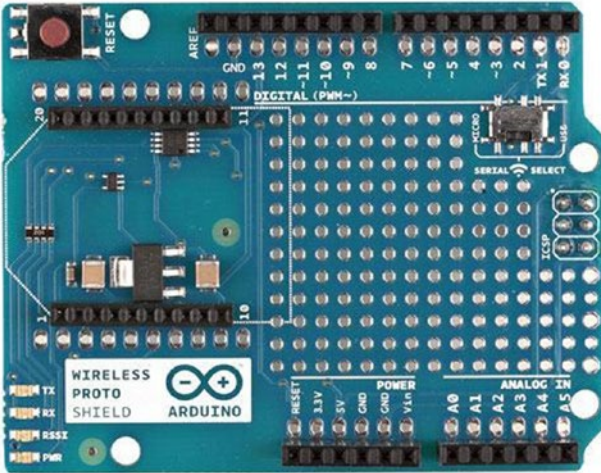


Figure 5-25. *Arduino Wireless SD Shield*

In this section, we build an XBee application on Arduino and a computer. The XBee Arduino application will send a random number to a computer over an XBee module. For testing, we build our demo with the following items:

- Arduino board
- XBee shield for Arduino
- Two XBee IEEE 802.15.4 devices
- XBee USB module

As another option, you can use XBee Wireless kit from SparkFun. You can check this product on this site: <https://www.sparkfun.com/products/13197>. You can see this kit in Figure 5-26.

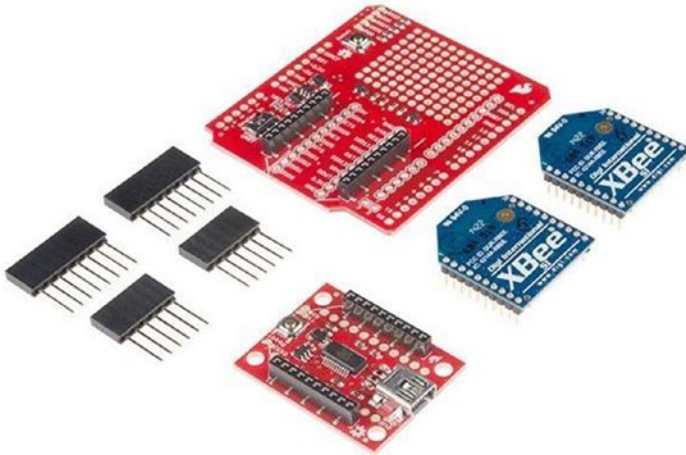


Figure 5-26. SparkFun XBee Wireless Kit

Now we try to build RF communication using XBee IEEE 802.15.4 modules on Arduino. We implement a simple application to send a random number. We need two XBee modules for demo. One module will work as a sender. Otherwise, it works as a receiver. In the next sub-section, we will implement our demo.

Configuring XBee IEEE 802.15.4

For demo implementation, I use Arduino UNO with XBee shield and XBee IEEE 802.15.4 module with XBee USB. You can see my demo devices in Figure 5-27.



Figure 5-27. Demo sample for XBee and Arduino

Before you attach an XBee module into an XBee shield, we should configure all XBee modules. We can use an XCTU application from Digi International. You can download it on <https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu>. Download and install this program.

Run the XCTU program. Attach your XBee module on XBee USB so your computer detects your modules. Then, add an XBee module into XCTU. You can see it in Figure 5-28. You should set XBee features, which is shown in Table 5-2. If you're finished, you save by clicking the **Write** icon in order to write these values into XBee firmware.

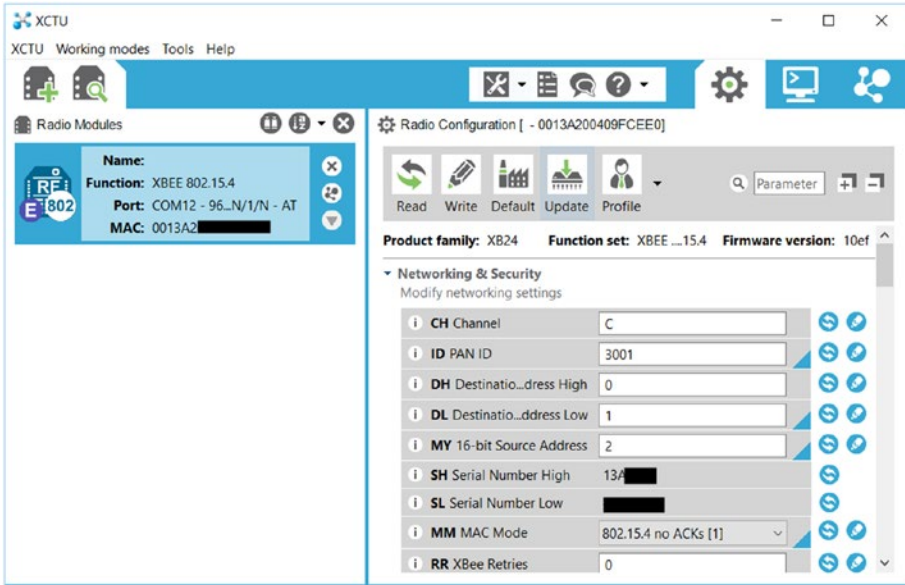


Figure 5-28. Configure XBee using XCTU

Table 5-2. XBee configuration in XCTU application

XBee Module	Features
XBee 1	ID = 3001
	MY = 1
	DL = 2
	MM = 1
	AP = 0
XBee 2	ID = 3001
	MY = 2
	DL = 1
	MM = 1
	AP = 0

Building an Arduino Sketch Program

Basically, after we configured our XBee IEEE 802.15.4 modules in the previous section, the modules can communicate with others in transparent mode. It means when a XBee module broadcasts a message on UART, other XBee modules will receive this message from UART. From this scenario, we build a Sketch program to send a random number from Arduino to a computer through XBee IEEE 802.15.4. You can write the following Sketch program:

```
long randomNumber;
void setup() {
  Serial.begin(9600);
  randomSeed(analogRead(0));
}

void loop() {
  randomNumber = random(20, 80);
  Serial.println(randomNumber);
  delay(2000);
}
```

Save this program as “XBeeArduino”. Compile and upload the program to the Arduino board.

Now you can attach XBee 1 into an XBee shield and then put the shield into an Arduino board. Furthermore, XBee 2 is attached into XBee USB and then plugged into a computer. Using Serial monitor tool, navigate this tool to the serial port to which XBee USB is attached. You should see an incoming number on the Serial Monitor tool. You can see it in Figure 5-29. This values comes from Arduino.

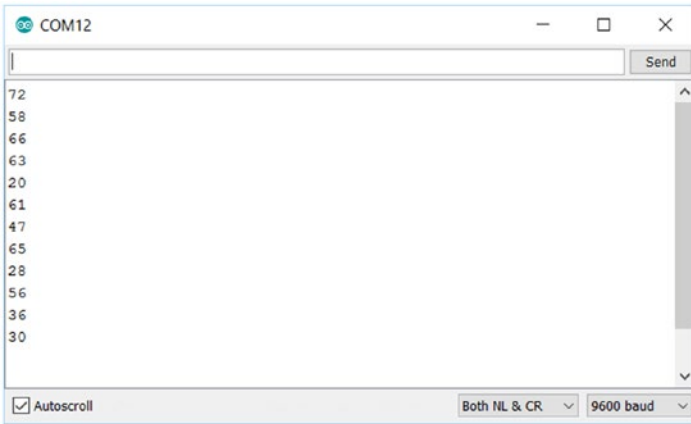


Figure 5-29. Program output from XBee on Serial Monitor tool

Building a UWP Program

In this section, we build a UWP program. This program will work as the XBee receiver which listens to incoming messages from a serial port on the computer. You attach the XBee module and USB breakout in order to plug into a computer through a USB cable. We use `SerialDevice` object to communicate with the serial port.

Now we can create a new project for a UWP template, called “XBeeApp”. Then, build a UI, which is shown in Figure 5-30. We add **ListBox** and **Button**. XBee data is displayed in `Text`.

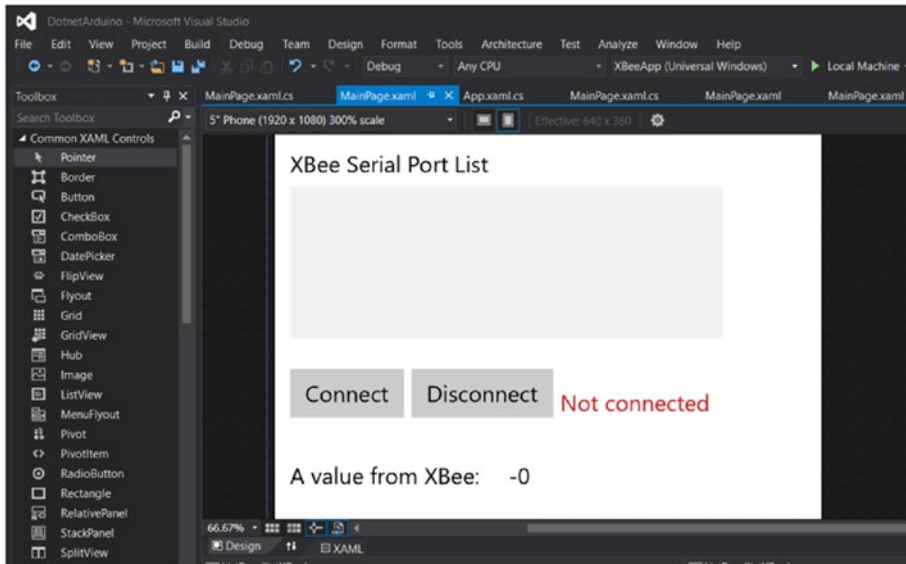


Figure 5-30. UI for XBee UWP application

The implementation of XAML UI can be written in the following scripts:

```
<Page
  x:Class="XBeeApp.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:XBeeApp"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d">

  <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <ListBox x:Name="listXBee" HorizontalAlignment="Left" Height="100"
      Margin="10,45,0,0" VerticalAlignment="Top" Width="285">
      <ListBox.ItemTemplate>
        <DataTemplate>
          <TextBlock Text="{Binding Name}" />
        </DataTemplate>
      </ListBox.ItemTemplate>
    </ListBox>
    <TextBlock x:Name="textBlock" HorizontalAlignment="Left"
      Margin="10,20,0,0" TextWrapping="Wrap" Text="XBee Serial Port List"
      VerticalAlignment="Top"/>
    <Button x:Name="btnConnect" Content="Connect"
      HorizontalAlignment="Left" Margin="10,165,0,0"
      VerticalAlignment="Top" Click="ConnectToXBee"/>
    <Button x:Name="btnDisconnect" Content="Disconnect"
      HorizontalAlignment="Left" Margin="90,165,0,0"
      VerticalAlignment="Top" Click="Disconnect"/>
    <TextBlock x:Name="txtState" HorizontalAlignment="Left"
      Margin="188,177,0,0" TextWrapping="Wrap" Text="Not connected"
      VerticalAlignment="Top" Foreground="#FFCA1E1E"/>
    <TextBlock x:Name="textBlock1" HorizontalAlignment="Left"
      Margin="10,225,0,0" TextWrapping="Wrap" Text="A value from XBee:"
      VerticalAlignment="Top"/>
    <TextBlock x:Name="txtValue" HorizontalAlignment="Left"
      Margin="154,225,0,0" TextWrapping="Wrap" Text="-0"
      VerticalAlignment="Top"/>

  </Grid>
</Page>
```

On MainPage.xaml.cs, we add codes. First, we add the required namespace as follows:

```
using System.Collections.ObjectModel;
using Windows.Devices.Enumeration;
using Windows.Devices.SerialCommunication;
```

```
using Windows.Storage.Streams;
using System.Threading;
using System.Threading.Tasks;
```

Then, we add variables.

```
public sealed partial class MainPage : Page
{
    private SerialDevice serialPort = null;
    private DataReader dataReaderObject = null;
    private ObservableCollection<DeviceInformation> listUART;
    private CancellationTokenSource ReadCancellationTokenSource;

```

We retrieve a list of serial ports while calling in constructor.

```
public MainPage()
{
    this.InitializeComponent();
    listUART = new ObservableCollection<DeviceInformation>();
    GetListOfXBee();
}

private async void GetListOfXBee()
{
    try
    {
        string aqs = SerialDevice.GetDeviceSelector();
        var dis = await DeviceInformation.FindAllAsync(aqs);

        for (int i = 0; i < dis.Count; i++)
        {
            listUART.Add(dis[i]);
        }
        listXBee.ItemsSource = listUART;
        listXBee.SelectedIndex = -1;

        btnConnect.IsEnabled = true;
        btnDisconnect.IsEnabled = false;
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine(ex.Message);
    }
}
}
```

We connect to XBee USB by clicking the Connect button. If you want to disconnect from XBee USB, click the Disconnect button. We implement these codes:

```
private async void ConnectToXBee(object sender, RoutedEventArgs e)
{
    var selection = listXBee.SelectedItems;

    if (selection.Count <= 0)
    {
        txtState.Text = "Select a Serial Port!";
        return;
    }

    DeviceInformation entry = (DeviceInformation)selection[0];

    try
    {
        serialPort = await SerialDevice.FromIdAsync(entry.Id);

        // Configure serial settings
        serialPort.WriteTimeout = TimeSpan.FromMilliseconds(1000);
        serialPort.ReadTimeout = TimeSpan.FromMilliseconds(1000);
        serialPort.BaudRate = 9600;
        serialPort.Parity = SerialParity.None;
        serialPort.StopBits = SerialStopBitCount.One;
        serialPort.DataBits = 8;
        serialPort.Handshake = SerialHandshake.None;

        // Create cancellation token
        ReadCancellationTokenSource = new CancellationTokenSource();

        btnConnect.IsEnabled = false;
        btnDisconnect.IsEnabled = true;
        txtState.Text = "Connected";
        ReadDataXBee();
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine(ex.Message);
        btnConnect.IsEnabled = true;
    }
}

private void Disconnect(object sender, RoutedEventArgs e)
{
    CloseXBee();
    txtState.Text = "Not connected";
}
```

Inside clicked event on **Connect** and **Disconnect** buttons we call `CloseXBee()` and `ReadDataXBee()` methods. We implement these codes as follows:

```
private void CloseXBee()
{
    try
    {
        CancelReadTask();
    }
    catch (Exception){}

    if (serialPort != null)
    {
        serialPort.Dispose();
    }
    serialPort = null;

    btnConnect.IsEnabled = true;
    btnDisconnect.IsEnabled = false;
}

private async void ReadDataXBee()
{
    try
    {
        if (serialPort != null)
        {
            dataReaderObject = new DataReader(serialPort.InputStream);

            // keep reading the serial input
            while (true)
            {
                await ReadXBeeAsync(ReadCancellationTokenSource.Token);
            }
        }
    }
    catch (Exception ex)
    {
        CloseXBee();
        System.Diagnostics.Debug.WriteLine(ex.Message);
    }
    finally
    {
        // Cleanup once complete
        if (dataReaderObject != null)
        {
            dataReaderObject.DetachStream();
            dataReaderObject = null;
        }
    }
}
```



```

    }
}
private void CancelReadTask()
{
    if (ReadCancellationTokenSource != null)
    {
        if (!ReadCancellationTokenSource.IsCancellationRequested)
        {
            ReadCancellationTokenSource.Cancel();
        }
    }
}

private async Task ReadXBeeAsync(Cancellation_token cancellationToken)
{
    Task<UInt32> loadAsyncTask;

    uint ReadBufferLength = 1024;

    // If task cancellation was requested, comply
    cancellationToken.ThrowIfCancellationRequested();
    dataReaderObject.InputStreamOptions = InputStreamOptions.Partial;
    loadAsyncTask = dataReaderObject.LoadAsync(ReadBufferLength).
    AsTask(cancellationToken);

    UInt32 bytesRead = await loadAsyncTask;
    if (bytesRead > 0)
    {
        string txt = dataReaderObject.ReadString(bytesRead);
        if (txt.IndexOf('\r') != -1)
        {
            string val = txt.Replace('\r', '\0');
            val = txt.Replace('\n', '\0');
            txtValue.Text = val;
        }
    }
}
}

```

Save this program.

Testing

Now you can run this program. You should see a list of attached serial ports on your computer. Please select a serial port for XBee. After it's selected, you should see incoming data on Text about a number which is sent from Arduino through XBee IEEE 802.15.4. A sample of the program output can be seen in Figure 5-31.

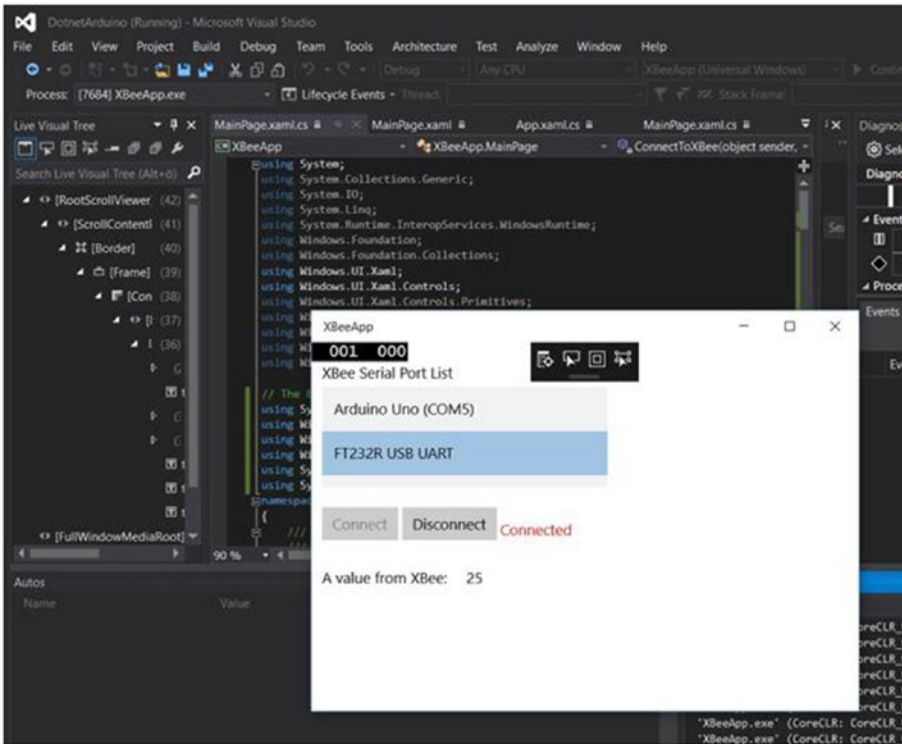


Figure 5-31. XBeepApp is running

Building a LoRa Network for Arduino

LoRa is a wireless technology developed to create the low-power, wide-area networks (LPWANs) required for machine-to-machine (M2M) and Internet of Things (IoT) applications. You can learn more about LoRa by visiting this site: <https://www.lora-alliance.org>. In this section, we learn how Arduino connects to another device through LoRa modules.

You can easily find LoRa modules or shields. In this section, we explore LoRa products from Dragino. Dragino Lora Shield is a long-range transceiver on an Arduino shield form factor and based on an open source library. You can see a list of Dragino Lora products on this site: <http://www.dragino.com/products/lora.html>. You can see a sample of Dragino LoRa shield for Arduino in Figure 5-32.

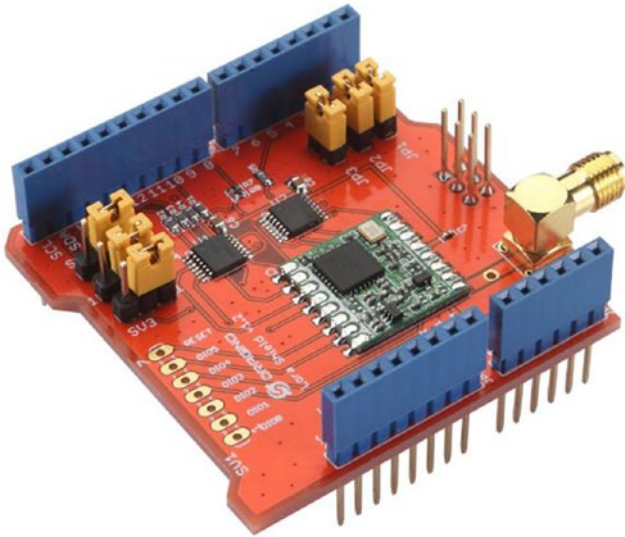


Figure 5-32. Dragino LoRa shield for Arduino boards

For testing in this section, I use Arduino Mega 2560 and Arduino Leonardo. I use two Dragino LoRa shields. You can see my implementation in Figure 5-33. To develop a LoRa application on Arduino, we can use Radiohead Packet Radio library. You can check it on this site: <http://www.airspayce.com/mikem/arduino/RadioHead/>.



Figure 5-33. Demo devices for my implementation

First, we download and extract Radiohead Packet Radio library (<http://www.airspayce.com/mikem/arduino/RadioHead/>) for Arduino board. You should ZIP the file. You can extract this library using Arduino software. Click menu Sketch ► Include Library ► Add .ZIP library. Then, navigate to the ZIP file for Radiohead Packet Radio library.

If it's done, your Radiohead library should be shown in Arduino library. You can see it in Figure 5-34. Now you restart Arduino software in order to reload all libraries on Arduino.

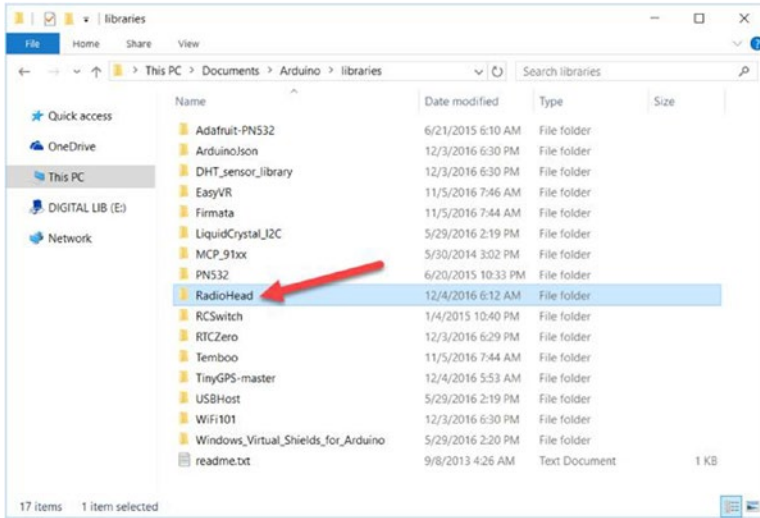


Figure 5-34. Adding RadioHead library on Arduino

Now you can start to write a sketch program. For testing, we use a program sample from Radiohead. The following are sketch scripts:

```
#include <SPI.h>
#include <RH_RF95.h>

// Singleton instance of the radio driver
RH_RF95 rf95;

void setup()
{
  Serial.begin(9600);
  while (!Serial) ; // Wait for serial port to be available
  if (!rf95.init())
    Serial.println("init failed");
}
```

```

void loop()
{
  Serial.println("Sending to rf95_server");
  // Send a message to rf95_server
  uint8_t data[] = "Hello World from Mega256!";
  rf95.send(data, sizeof(data));

  rf95.waitPacketSent();
  // Now wait for a reply
  uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);

  if (rf95.waitAvailableTimeout(3000))
  {
    // Should be a reply message for us now
    if (rf95.recv(buf, &len))
    {
      Serial.print("got reply: ");
      Serial.println((char*)buf);
    }
    else
    {
      Serial.println("recv failed");
    }
  }
  else
  {
    Serial.println("No reply, is rf95_server running?");
  }
  delay(400);
}

```

You can change the value on variable data, for instance, “Hello World from Mega256” for Arduino Mega 2560 and “Hello World from Leonardo!” for Arduino Leonardo. Save all programs.

Now you can compile and upload a sketch program to all Arduino boards. You should run one program for one computer. You can run two LoRa programs with Radiohead library on one computer. For testing, I run my Arduino Mega 2560 on Windows and Arduino Leonardo on Mac.

After uploading a sketch program, you can open Serial Monitor. You should see messages. You can see the program output in Figures 5-35 and 5-36.

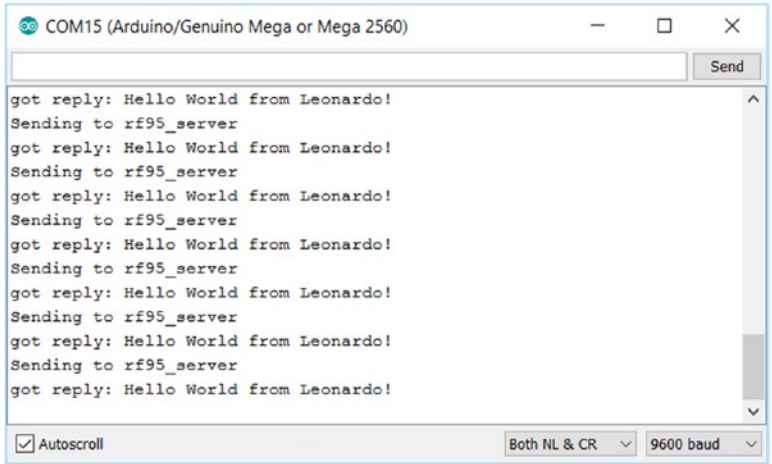


Figure 5-35. Program output on Arduino Mega 2560

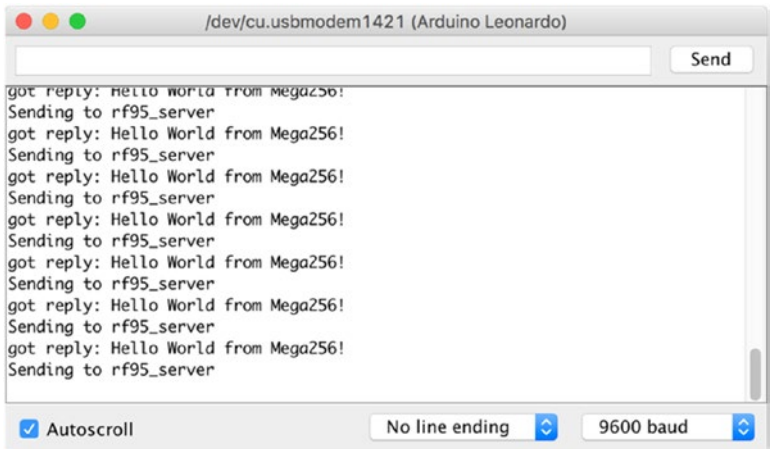


Figure 5-36. Program output on Arduino UNO

What's next? After we can communicate among Arduino over a LoRa network, we develop a UWP application to transfer data over a LoRa network through Arduino. You can also develop any application by applying a LoRa network.

Location-based Application for Arduino

In some cases you may need information about location; for instance, you want to retrieve information about temperature and humidity in a certain location. We can solve this case by implementing a location-based system.

One of a solution is applying a GPS module on our board. There are many GPS modules which work with Arduino boards. The famous GPS chip is U-blox (<https://www.u-blox.com/>). You can see a GPS module form in Figure 5-37.



Figure 5-37. GPS Module with U-blox device

You can find GPS modules in an online store, such as Aliexpress, eBay, SparkFun and Adafruit. One GPS module can be seen in Figure 5-37. This module can be used for Arduino and Raspberry Pi.

This GPS module has five pins: VCC, GND, TX, RX, and PPS. For testing, I attach this module into Arduino UNO as follows:

- GPS module VCC is connected to 5V/ V3.3 Arduino.
- GPS module GND is connected to GND Arduino.
- GPS module TX is connected to digital pin 5 Arduino.
- GPS module RX is connected to digital pin 11 Arduino.
- GPS module PPS is connected to GND Arduino.

You can see my wiring implementation in Figure 5-38.

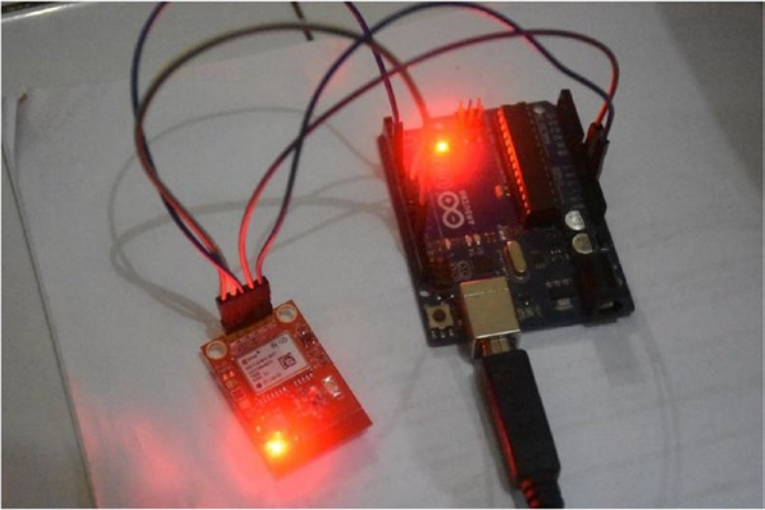


Figure 5-38. Wiring for GPS module and Arduino

To develop a GPS application for Arduino, we can use TinyGPS library (<https://github.com/mikalhart/TinyGPS>). You can download this library and then install it via Arduino software. If you're finished, you should see TinyGPS in the Arduino library path, which is shown in Figure 5-39. Restart your Arduino in order to reload all library paths.

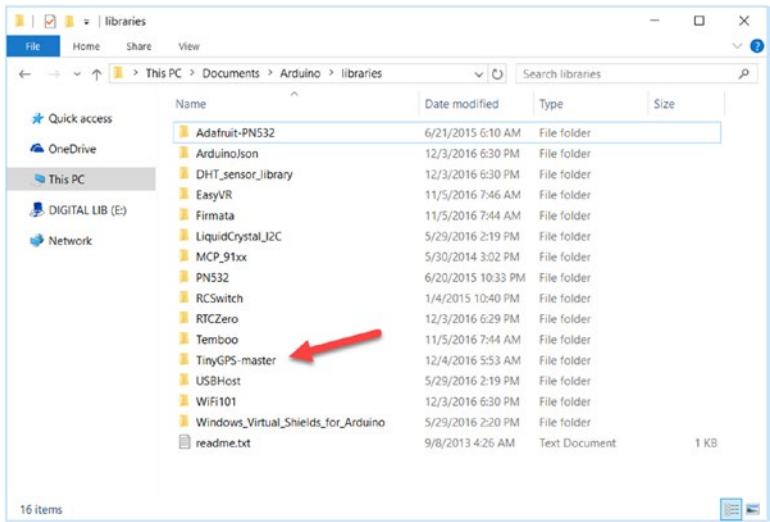


Figure 5-39. TinyGPS library in Arduino library path

Now we create a GPS application on Arduino software. This program will read GPS data on UART to retrieve the current position. After it's received, GPS data will be parsed and the result will be shown in UART. Write the following sketch:

```
#include <SoftwareSerial.h>
#include <TinyGPS.h>

// D5 >>> Rx, D11 >>> Tx
SoftwareSerial gps(5, 11); // RX, TX
char val;
TinyGPS gps_mod;

void setup() {
  Serial.begin(9600);
  pinMode(13, OUTPUT);
  gps.begin(9600);
  Serial.println("GPS On..");
}

void loop() {
  bool newData = false;
  unsigned long chars;
  unsigned short sentences, failed;

  // read GPS position every 3 seconds
  for (unsigned long start = millis(); millis() - start < 3000;) {
    while (gps.available()){
      char c = gps.read();
      //Serial.println(c);
      if (gps_mod.encode(c))
        newData = true;
    }
  }

  if (newData) {
    float flat, flon;
    unsigned long age;

    digitalWrite(13, HIGH);
    gps_mod.f_get_position(&flat, &flon, &age);
    print_data("LAT=");
    print_num_data(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6);
    print_data(" LON=");
    print_num_data(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6);
    print_data(" SAT=");
    print_num_data(gps_mod.satellites() == TinyGPS::GPS_INVALID_SATELLITES ?
    0 : gps_mod.satellites());
    print_data(" PREC=");
  }
}
```

```

    print_num_data(gps_mod.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 : gps_
mod.hdop());

    break_line();
    digitalWrite(13, LOW);
}
}

void print_data(char msg[30]) {
    Serial.print(msg);
}
void print_num_data(float msg,int n) {
    Serial.print(msg, n);
}
void print_num_data(int msg) {
    Serial.print(msg);
}
void break_line() {
    Serial.println("");
}
}

```

Save this program as “**GPSArduino**”.

Compile and upload the sketch program into Arduino. Open the Serial Monitor tool to see GPS data. You can see the sample output in Figure 5-40.

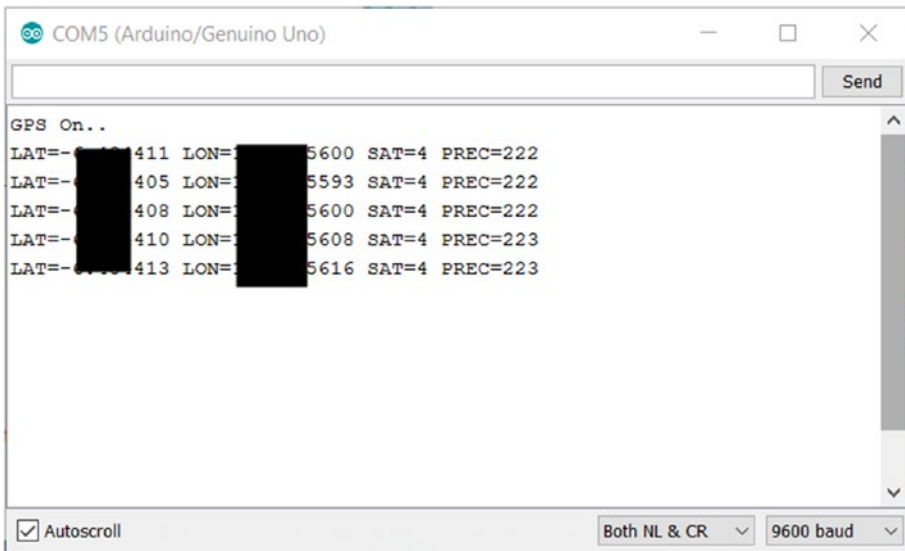


Figure 5-40. Running GPS application for Arduino

Arduino and Cloud Server

A server that you develop may not have a good performance if there are many requests from Arduino. Server scalability may not be achieved. Because of this situation, you could consider using a cloud server. This server is designed to handle many requests.

In this section, we review several cloud servers with which our Arduino can communicate.

Let's explore!

Arduino Cloud

Arduino cloud is a cloud server from Arduino LLC. You can access Arduino cloud on <https://cloud.arduino.cc/>. You need to register to enable accessing this code. You can see it in Figure 5-41.

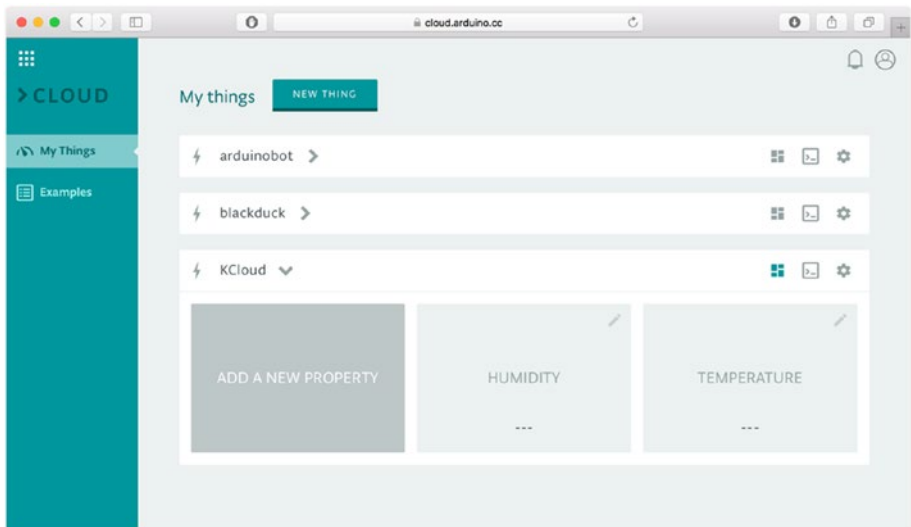


Figure 5-41. Arduino cloud

Currently, Arduino cloud is supported for Arduino MKR1000, WiFi 101 shield, and Arduino YUN. You need to register your Arduino board to this cloud. After it's registered, you should obtain an ID and password for your board.

After your board is registered, you should add a property to catch your data. For instance, I add two properties: Humidity and Temperature (see Figure 5-41). Arduino provides a Sketch sample for sending data. The following is a modified sketch program to send humidity and temperature to Arduino Cloud. I use an Arduino MKR1000 board. You can use it or Arduino with a WiFi 101 shield.

```

#include <WiFi101.h>
#include <ArduinoCloud.h>

//////// Wifi Settings //////////
char ssid[] = "ssid";
char pass[] = "ssid key";

// Arduino Cloud settings and credentials
const char userName[] = "<username>";
const char thingName[] = "<arduino name>";
const char thingId[] = "<board id>";
const char thingPsw[] = "<password>";

WiFiSSLClient sslClient;

// build a new object "KCloud"
ArduinoCloudThing KCloud;

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

  // attempt to connect to WiFi network:
  Serial.print("Attempting to connect to WPA SSID: ");
  Serial.println(ssid);

  while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
    // unsuccessful, retry in 4 seconds
    Serial.print("failed ... ");
    delay(4000);
    Serial.print("retrying ... ");
  }
  Serial.println("connected to wifi");

  KCloud.begin(thingName, userName, thingId, thingPsw, sslClient);
  KCloud.enableDebug();
  // define the properties
  KCloud.addProperty("humidity", FLOAT, R);
  KCloud.addProperty("temperature", FLOAT, R);

  Serial.println("connected");
  randomSeed(analogRead(0));
}

void loop() {
  KCloud.poll();

  long temp = random(10, 20);
  long humidity = random(20, 80);

```

```

temp = temp + 0.21 * temp;
humidity = humidity + 0.45 * humidity;

KCloud.writeProperty("humidity", String(humidity,2));
KCloud.writeProperty("temperature", String(temp,2));

delay(2000);

}

```

Change values for ssid, pass, userName, thingName, thingId, and thingPsw.

If you're done, compile and upload the Sketch program into Arduino. You should see Humidity and Temperature data on the Arduino Cloud.

You can combine this program with a UWP application to retrieve data from the Arduino cloud and show it on an application.

Summary

In this chapter, we have learned how our Arduino board connects to the Internet through a wired and WiFi connection. We built a simple IoT program to turn on/off LEDs which are attached on an Arduino board from a UWP program. We continued our journey to learn how to extend a WRA program over WiFi so our UWP program can communicate with Arduino through WiFi.

Moreover, we explored RF communication for Arduino. We applied XBee and LoRa modules to communicate with other systems. We also involved a GPS module to detect current location on an Arduino board.

In the last section, we connected our Arduino board to a Cloud server. We used Arduino Cloud as sample. Sending data to Arduino Cloud was done using a sketch program.

To upgrade your skill for Arduino development, you can try to use several sensor and actuator devices in your projects. Connect them to an external system using connectivity methods which we have learned. Now you can build your own projects in automation and IoT.

Index

■ A

- Actuator devices 45
 - .NET
 - COM5, 67
 - RGBControlApp, 65
 - RGB LED, 53–57, 65
 - SolidSoils4Arduino, 65, 67
 - Servo, 53–54
- Adafruit Feather 32u4 Adalogger, 8
- Atmel's Embedded Debugger (EDBG), 4

■ B

- BLE, 6
- Bluetooth
 - WRA over, 100
 - Arduino program, writing, 105
 - pairing Arduino Bluetooth and computer, 102–103
 - testing, 107
 - UWP program, writing, 105–106
 - UWP project, 104
 - wiring, 101–102

■ C

- Client app, 129
- Cloud, 161–163
- COM ports, Bluetooth, 106–107
- Control analog I/O, WRA, 85
 - Arduino program, 87
 - testing, 91–92
 - UWP project, 87
 - wiring, 86
 - writing UWP program, 87–91

■ D

- DHT22 module, 46–51
- DHT sensor library, 49–50
- DotnetDHT, 61
- Dragino LoRa shield, 152–153

■ E

- Ethernet shield
 - with PoE, 111
 - UNO and, 111
 - WebServer, 112–113

■ F

- Feather board models, 8
- Firmata protocol
 - add sketch program, 39
 - SolidArduino, 41
 - SolidSoils4Arduino, 42–44
 - standard sketch program, 40

■ G

- Genuino 101 boards, 6–7
- Genuino MKR1000, 5
- GPS modules
 - pins, 157
 - Serial Monitor tool, 160
 - TinyGPS library, 158
 - UART, 159–160
 - U-blox, 157
- Graphical Unit Interface (GUI), 70

■ H

- HC-06 Bluetooth module, 100–101, 103

■ I

- IDE, 9
- InitWRA() method, 105
- Inter-Integrated Circuit (I2C)
 - PCF8591 AD/DA module, 30–33
 - SparkFun Humidity and Temperature Sensor Breakout, 30
- wires, 29
- WRA through, 92
 - testing, 99
 - UWP project, 94
 - wiring, 93–94
 - writing UWP program, 95–99
- Internet, 4–7
 - WiFi, 114–121
 - wired, 110–114
- Internet of Things (IoT), 4–7, 109–110
- I/O communication
 - communication models, 22
 - I2C communication, 29–33
 - serial communication,
 - 22, 24–29
 - UNO, 21–22

■ J, K

- JSON, 58–59

■ L

- LED blinking
 - code, 12–13
 - compile program, 14
 - IDE configuration, 13
 - target and port, 13
 - UNO, 14
- Linux OS, 6
- LLC, 1, 5
- LoRa network, 152–156

■ M

- Master In Slave Out (MISO), 26–27
- Master Out Slave In (MOSI), 26–27
- Mega 2560, 3, 155–156
- MKR1000, 117
 - COMx, 116
 - ConnectWithWPA, 118–120
 - device manager, 117
 - enable software, 115

.NET

- demo wiring, 122
- HTTP GET URL mapping, 122
- LED 1 lighting, 128
- LEDs Sketch program, 124–127
- response on browser, 128
- Serial monitor tool, 127, 129
- URL mapping, 121–122, 124
- wiring for LEDs, 123–124
- Serial Monitor tool, 120
- WiFi101 library, 117–118

■ N

- .NET
 - actuating app, 65, 67–68
 - application program, 73–74
 - LEDs, 34
 - program, 35–36
 - Visual Studio, 36–37
 - wiring, 34
 - sensing app, 58–61, 63–64
 - writing program, 77–82
- Network stacks, 110

■ O

- Open source-based hardware, 1
- OpenWrt Linux, 6

■ P, Q

- PCF8591 AD/DA module, 30–33, 92–93, 96–97
- Pulse Width Modulation (PWM), 55, 85

■ R

- Radiohead Packet Radio library, 154–155
- ReadADC() methods, 98
- Really Bare Bones Board (RBBB), 8
- RF communication
 - defined, 141
 - Wireless SD Shield, 142
 - XBee IEEE 802.15.4 module,
 - 141–143
- RGBControlApp, 65, 68
- RGB LED
 - displays, 54
 - forms, 53
 - pin layout, 54–55

PMW, 55
 sketch program, 56–58
 wiring, 55–56

■ **S**

SAMD Boards, 115
 Sensor devices, 45
 DHT22 module, 46–47
 DHT sensor library, 49–50
 pins layout, 48
 program to read temperature and humidity, 50–52
 wiring, 48–49
 feature, 46
 .NET, 58–64
 SparkFun Electret Microphone Breakout, 47
 Serial communication
 Serial Monitor tool, 23–25
 sketch program, 23
 SparkFun Triple Axis Accelerometer Breakout, 26
 SPI communication, 26, 28
 Serial Monitor tool, 23–25
 Serial Peripheral Interface (SPI)
 library, 27, 29
 MISO and MISO pins, 27
 pins, 26
 sketch program, 28
 Servo, 53–54
 SolidArduino, 41
 SolidSoils4Arduino, 42–44
 SparkFun Bluetooth Mate Silver, 100
 SparkFun Electret Microphone Breakout, 47
 SparkFun Humidity and Temperature Sensor Breakout, 30
 SparkFun RedBoard, 7
 SparkFun Triple Axis Accelerometer Breakout, 26
 SRL, 1, 4
 StandardFirmataWiFi Sketch program, 134–135

■ **T**

TinyGPS library, 158
 TurnOffLeds method, 78
 Two-Wire Interface (TWI). *See* Inter-Integrated Circuit (I2C)

■ **U**

U-blox device, 157
 Universal Asynchronous Receiver Transmitter (UART). *See* Serial communication
 Universal Windows Platform (UWP), 69
 .NET application, 69
 program
 control analog I/O, 87–91
 I2C bus, 95–99
 WRA over Bluetooth, 105–106
 project
 control analog I/O, 87
 I2C bus, 94
 templates, 71
 WRA over Bluetooth, 104
 targets, 70
 technology, 70
 WRA, 70
 UNO boards
 beginners, 2–3
 LLC, 3
 R3 board, 2
 WiFi, 4–5
 UpdateData() method, 98
 UWP application, 121–122
 LEDs, 137–139
 MKR1000, 132–133
 testing, 133
 Toggled method, 131–132
 UI, 130–131
 WiFiWRA, 135–137

■ **V**

Visual Micro, 15
 Visual Studio 2015
 editor, 17
 project templates, 16
 running and debugging, 18
 serial port configuration, 18
 start page, 11
 Visual Micro, 15

■ **W**

WiFi101 library, 117
 WiFi network
 MKR1000, 114–121
 SAMD Boards, 115
 YUN Shield, 115

- WiFiWRA, 135–137
- Windows 10, 10, 70, 71, 102, 103, 106
- Windows Presentation Foundation (WPF), 70
- Windows Remote Arduino (WRA), 69
 - building first program, 71
 - adding library, 75–76
 - Arduino program, 72
 - .NET application program, 73–74
 - testing, 82–84
 - wiring, 71
 - writing .NET program, 77–82
 - control analog I/O, 85
 - Arduino program, 87
 - testing, 91–92
 - UWP project, 87
 - wiring, 86
 - writing UWP program, 87–91
 - I2C bus, through, 92
 - testing, 99
 - UWP project, 94
 - wiring, 93–94
 - writing UWP program, 95–99
 - over Bluetooth, 100
 - Arduino program, writing, 105
 - pairing Arduino Bluetooth and computer, 102–103
 - testing, 107
 - UWP program, writing, 105–106
 - UWP project, 104
 - wiring, 101–102

- setting up Arduino for, 70
- StandardFirmataWiFi Sketch program, 134–135
- testing, 140
- UWP application, 135–140

- Wired network, 110–114
- Wireless SD Shield, 141–142
- Wireless Sensor Network (WSN), 141

■ **X**

- XAML technology, 70
- XBeeApp, 146
- XBee IEEE 802.15.4 module, 143
 - configure, 143–144
 - sketch program, 145
 - SparkFun, 142–143
 - testing, 142, 151–152
 - UWP program, 146–151
 - WSN, 141
- XCTU application, 144–145

■ **Y**

- YUN board, 6
- Yun Shield, 115

■ **Z**

- Zero board, 4