Woolsey Workshop

Cheatsheet: Arduino Basics

Revisions 20

General

General Information

- Arduino is a popular open-source electronics platform based on easy-to-use hardware and software. According to Arduino, it allows "people around the world to easily access advanced technologies that interact with the physical world".
- The Arduino ecosystem provides the ability to develop and execute programs to control connected electronics using a wide variety of boards in a microcontroller agnostic way (for the most part).
- The Arduino UNO was the first board developed and released by Arduino and that line of boards continues to be extremely popular among hobbyists and electronics enthusiasts. Arduino has released many other boards geared for specific purposes or increased functionality and has even moved into the professional market.
- Third party manufacturers also provide boards for use within the Arduino ecosystem. However, it is generally recommended that only official Arduino boards should be used as many clones do not offer the same level of quality and support. Some companies, such as Adafruit and Sparkfun, provide their own lines of high quality boards and products that are known to work well within the Arduino ecosystem.
- Add-on boards, called shields, are also available by Arduino and others that provide additional hardware capabilities.
- The last program uploaded to an Arduino board is saved in on-board memory and will run automatically the next time power is applied to the board. This provides the ability to run an Arduino in the field without having to be connected to a computer.

Resources

- Arduino Website
- Arduino Store
- Arduino Hardware (contains product overviews, tech specs, and general documentation)
- Arduino Software (includes the IDE)
- Arduino Documentation
- Arduino Getting Started Guide
- Arduino Tutorials
- Arduino Forum (get help from Arduino and other users)

Recommended Project Start-Up Procedure

1. Disconnect your computer, all power sources, and connected electronics from your Arduino board.
2. Connect your Arduino board to your computer.
3. Open the Arduino IDE.
4. Upload the *BareMinimum* (Main Menu > File > Examples > 01.Basics > BareMinimum) sketch to reset all pins back to their default states. Although this is not technically necessary, it is a good habit to follow in order to avoid accidental damage to your board or connected electronics when applying power to the board for your next project as the board will automatically run the last sketch uploaded.
5. Disconnect the Arduino board from your computer. This avoids accidental damage during wiring.
6. Connect your associated electronics to your Arduino board.
7. Connect your optional power supplies to your Arduino board.
8. Connect the Arduino board to your computer.
9. Verify the correct Arduino board is selected within the Arduino IDE.
10. Create or open a sketch.

Recommended Project Shutdown Procedure

1. Save the current sketch.
2. Upload the *BareMinimum* sketch to reset all pins back to their default states.
3. Exit the Arduino IDE application.
4. Disconnect the Arduino board from your computer and any optional power sources.

Arduino IDE

General Information

- The Arduino IDE (Integrated Development Environment) is the application used to develop and upload programs to an Arduino or compatible development board. It is available for Linux, macOS, and Windows based operating systems.
- The IDE's front end (graphical user interface) is based on the open source *Theia* platform and *Electron* framework.
- The IDE's backend is powered by the Arduino CLI (*arduino-cli*) command line utility.
- The default locations for sketches per platform are
  - Linux/macOS: ~/Documents/Arduino
  - Windows: C:\Users\<username>\Documents\Arduino
- The *Arduino AVR Boards* core platform is preinstalled by the Arduino IDE and supports most of Arduino's classic AVR microcontroller based boards by default. You will need to install additional core packages from within the *Boards Manager* to support boards that use other microcontrollers. For instance, the *Arduino UNO R4 Boards* core is required to support the Arduino UNO R4 Minima and WiFi boards containing the Renesas RA4M1 family of microcontrollers.

Resources

- Arduino IDE 2 Documentation
- Arduino CLI Documentation

Navigation

- Get help documentation.
  - Main Menu > Help > …
- Exit the IDE application.
  - Main Menu > Arduino IDE > Quit Arduino IDE
- Update the IDE settings.
  - Main Menu > Arduino IDE > Settings…
- Install a core package for board support.
  - *BOARDS MANAGER* panel
  - Main Menu > Tools > Board > Boards Manager…
  - Search for and install the appropriate core platform for the supported board.
- Select a development board.
  - Auto-detect board/port selector in the toolbar
  - Main Menu > Tools > Board > …

- Select a serial port.
  - Auto-detect board/port selector in the toolbar
  - Main Menu > Tools > Port > …
- Create a new sketch.
  - *New Sketch* button within the *SKETCHBOOK* panel
  - Main Menu > File > New Sketch
- Open an existing sketch.
  - Ellipsis (…) button next to sketch name within the *SKETCHBOOK* panel
  - Main Menu > File > Sketchbook > …
  - Main Menu > File > Open…
- Open a recent sketch.
  - Main Menu > File > Open Recent > …
- Open an example sketch (contains basic and library specific examples).
  - Main Menu > File > Examples > …
- Save a sketch.
  - Main Menu > File > Save
- Save a sketch with a new name.
  - Main Menu > File > Save As…
- Close a sketch.
  - *Close* (x) button in the window title bar
  - Main Menu > File > Close
- Compile and verify a sketch.
  - *Verify* (checkmark) button in the toolbar
  - Main Menu > Sketch > Verify/Compile
- Upload and run a sketch on the board.
  - *Upload* (right arrow) button in the toolbar
  - Main Menu > Sketch > Upload
- Auto-format the sketch.
  - Main Menu > Tools > Auto Format
  - Main Menu > Edit > Auto Format
- Add a new file to a sketch.
  - Click the ellipsis (…) button on the right side of the tab bar and then select *New Tab* from pop-up menu.
- Add an existing file to a sketch.
  - Main Menu > Sketch > Add File…
- Install libraries.
  - *LIBRARY MANAGER* panel
  - Main Menu > Tools > Manage Libraries…
  - Main Menu > Sketch > Include Library > Manage Libraries…
  - Search for and install the library.

- Include a library in a sketch.
  - Main Menu > Sketch > Include Library > …
  - #include <*library header*>
- Open the *Serial Monitor* to view the printed serial output.
  - *Serial Monitor* (magnifying glass) button in the toolbar
  - Main Menu > Tools > Serial Monitor
- Open the *Serial Plotter* to view the graphical serial output.
  - *Serial Plotter* (waveform) button in the toolbar
  - Main Menu > Tools > Serial Plotter

Sketch Programming

General Information

- The programming language used by the Arduino IDE is called Sketch and is based on C++. This allows the use of most standard C and C++ language constructs and libraries. The Sketch language adds additional constructs and routines to interface with the microcontroller's general purpose input/output (GPIO) pins and hardware.
- A file containing a Sketch program is called a sketch and has a *.ino* file extension.
- Sketches are saved as a project directory containing a source file with the same name as the project and a *.ino* file extension. For example, a sketch named *MySketch* will be saved in a directory named *MySketch* that contains the sketch source file named *MySketch.ino*.
- The `setup()` function runs only once and is where all initialization code is placed.
- The `loop()` function runs repeatedly after the `setup()` function has completed and is where code that is always in action is placed.
- For boards containing a built-in LED, it can be referenced with the `LED_BUILTIN` constant.
- Digital pins are typically referenced as just a number (`0`, `1`, `2`, ...), but can sometimes also be referenced with the D prefix (`D0`, `D1`, `D2`, ...) depending on the configuration of the underlying core platform.
- Analog pins are referenced with the A prefix (`A0`, `A1`, `A2`, ...).
- Arduino provides a selection of officially supported libraries for common electrical interfaces and added general functionality. Many third party libraries are also available.
- Printing with `Serial.print()` and `Serial.println()` will be displayed in the *Serial Monitor*. The *Serial Plotter* can be used to display waveforms of printed data.
- The preferred approach to defining constants is to use the `const` keyword, versus the `#define` preprocessor directive, in order to provide better type checking.
```
const uint8_t RedLED = 2;
```
- The `String` datatype should generally not be used for working with strings as it uses more memory and can lead to memory fragmentation. Standard C-style strings are preferred.
- Be careful using a lot of long strings in your sketch as they can take up a lot of the available memory on some of the smaller microcontrollers. If you don't need to modify the strings or data while your sketch is running, consider storing them in flash (program) memory instead of SRAM. Do this by using the `PROGMEM` keyword when storing or the `F()` macro when printing.
```
const static PROGMEM char long_str[] = "This is a long string.";
Serial.println(F("This is a long string."));
```

- Utilize preprocessor directives to distinguish between different boards.

```
#if defined(ARDUINO_AVR_UNO)
   // Arduino UNO R3 board specific code
#elif defined(ARDUINO_UNOWIFIR4)
   // Arduino UNO R4 WiFi board specific code
#else
   // code for all other boards
#endif
```

Resources

- Arduino Language Reference Guide
- Arduino Libraries Documentation
- Blink: Making An LED Blink On An Arduino Uno Tutorial On Woolsey Workshop
- How To Use Buttons With Your Arduino Tutorial on Woolsey Workshop
- Using The Arduino Serial Plotter To Visualize Real Time Data Tutorial on Woolsey Workshop
- Using The Arduino Command Line Tutorial on Woolsey Workshop
- Arduino Command Line Cheatsheet on Woolsey Workshop
- All Arduino Based Tutorials On Woolsey Workshop

Common Operations

- Configure the mode of a digital pin.
  - `pinMode(<pin>, <mode>)`
    - `<pin>` is the pin number, i.e. `0`, `1`, `2`, …
    - `<mode>` is the pin mode, i.e. `INPUT`, `INPUT_PULLUP`, or `OUTPUT`.
- Read the value of a digital pin.
  - `digitalRead(<pin>)`
    - `<pin>` is the pin number, i.e. `0`, `1`, `2`, …
    - Returns a value of `HIGH` or `LOW`.
- Write a value to a digital output pin.
  - `digitalWrite(<pin>, <value>)`
    - `<pin>` is the pin number, i.e. `0`, `1`, `2`, …
    - `<value>` is the pin value, i.e. `HIGH` or `LOW`.
- Read the value of an analog (ADC) pin.
  - `analogRead(<pin>)`
    - `<pin>` is the pin number, e.g. `A0`, `A1`, `A2`, …
    - Returns a value in the range of 0-1023 (10-bit) by default.
    - The resolution can be changed with the `analogReadResolution()` function on supported boards.

- Write an analog value to an analog (DAC) or digital (PWM) output pin.
  - `analogWrite(<pin>, <value>)`
    - `<pin>` is the pin number, i.e. `A0`, `A1`, `A2`, … (analog) or `0`, `1`, `2`, … (digital).
    - `<value>` is the pin value, e.g. 0-255 (default) for 8-bit DAC and PWM.
    - Values written to a digital pin use pulse width modulation (PWM) to simulate an analog value.
    - The resolution can be changed with the `analogWriteResolution()` function on supported boards.
- Set the analog reference voltage.
  - `analogReference(<type>)`
    - `<type>` is the voltage reference type, e.g. `DEFAULT`, `INTERNAL`, `EXTERNAL`, etc.
    - See the documentation for the supported types for specific boards.
- Set the reading resolution for analog (ADC) input pins.
  - `analogReadResolution(<bits>)`
    - `<bits>` is the resolution of the pins, e.g. `10` (0-1023), `12` (0-4095), `14` (0-16,384), etc.
    - The default is `10` bits.
    - The available resolutions are dependent on the board's capabilities.
- Set the writing resolution for analog (DAC) and digital (PWM) output pins.
  - `analogWriteResolution(<bits>)`
    - `<bits>` is the resolution of the pins, e.g. `8` (0-255), `10` (0-1023), `12` (0-4095), etc.
    - The default is `8` bits.
    - The available resolutions are dependent on the board's capabilities.
- Pause execution of the program for a specific amount of time.
  - `delay(<ms>)`
    - `<ms>` is the time in milliseconds.
  - `delayMicroseconds(<us>)`
    - `<us>` is the time in microseconds.
- Get the amount of time that the program has been running.
  - `millis()`
    - Returns the time in milliseconds.
    - The value will overflow after about 50 days.
  - `micros()`
    - Returns the time in microseconds.
    - The value will overflow after about 70 minutes.
- Read a specific bit of a numeric value.
  - `bitRead(<value>, <bit>)`
    - `<value>` is the value.
    - `<bit>` is the bit position.
    - Returns the value of the bit, i.e. `0` or `1`.

- Clear a specific bit of a numeric variable.
  - `bitClear(<var>, <bit>)`
    - `<var>` is the variable.
    - `<bit>` is the bit position.
- Set a specific bit of a numeric variable.
  - `bitSet(<var>, <bit>)`
    - `<var>` is the variable.
    - `<bit>` is the bit position.
- Write a specific bit of a numeric variable.
  - `bitWrite(<var>, <bit>, <value>)`
    - `<var>` is the variable.
    - `<bit>` is the bit position.
    - `<value>` is the value to write, i.e. `0` or `1`.
- Get a random integer number.
  - `random([<min>, ]<max>)`
    - `<min>` is the optional minimum value (inclusive); defaults to `0`.
    - `<max>` is the maximum returned value (exclusive).
    - Returns a random integer within the specified range.
- Initialize the serial port.
  - `Serial.begin(<speed>)`
    - `<speed>` is the baud rate for serial communication. A value of `9600` is typically used and must match the baud rate specified by the *Serial Monitor* to properly view text.
- Print data as human-readable text to the serial port.
  - `Serial.print(<data>[, <format>])` -- does not include new line
  - `Serial.println(<data>[, <format>])` -- includes new line
    - `<data>` is the data to print, e.g. characters, strings, numbers, etc.
    - `<format>` is the optional format to use when printing the data.
      - Use a base value of `BIN` (binary), `OCT` (octal), `DEC` (decimal), or `HEX` (hexadecimal) for integer formatting; defaults to `DEC`.
      - Use an integer value to specify the number of decimal digits for floating point formatting; defaults to `2`.
  - The serial port needs to be initialized before it can be used for printing data.
  - View serial output with the *Serial Monitor* (text) or *Serial Plotter* (graph).
- Generate a square wave on a digital output pin.
  - `tone(<pin>, <freq>[, <time>])`
    - `<pin>` is the pin number, i.e. `0`, `1`, `2`, ...
    - `<freq>` is the frequency in Hz.
    - `<time>` is the optional tone duration in milliseconds; defaults to forever.

- Stop generating a square wave on a digital output pin.
  - `noTone(<pin>)`
    - `<pin>` is the digital pin number, i.e. `0`, `1`, `2`, ...
- Capture the length of a pulse on a digital input pin.
  - `pulseIn(<pin>, <type>[, <timeout>])` -- better for shorter pulses
  - `pulseInLong(<pin>, <type>[, <timeout>])` -- better for longer pulses
    - `<pin>` is the pin number, i.e. `0`, `1`, `2`, ...
    - `<type>` is the type of pulse, i.e. `HIGH` or `LOW`.
    - `<timeout>` is the optional number of microseconds to wait for the pulse to start; defaults to one second.
    - Returns the duration of the captured pulse in microseconds.
    - Pulses from 10 microseconds to 3 minutes in length are supported.
- Shift in a byte of data from a digital input pin.
  - `shiftIn(<data_pin>, <clock_pin>, <bit_order>)`
    - `<data_pin>` is the input data pin number, i.e. `0`, `1`, `2`, ...
    - `<clock_pin>` is the output clock pin number, i.e. `0`, `1`, `2`, ...
    - `<bit_order>` is the bit order of the shifted bits, i.e. `MSBFIRST` (most significant bit first) or `LSBFIRST` (least significant bit first).
    - Returns a byte of the data shifted in.
- Shift out a byte of data to a digital output pin.
  - `shiftOut(<data_pin>, <clock_pin>, <bit_order>)`
    - `<data_pin>` is the output data pin number, i.e. `0`, `1`, `2`, ...
    - `<clock_pin>` is the output clock pin number, i.e. `0`, `1`, `2`, ...
    - `<bit_order>` is the bit order of the shifted bits, i.e. `MSBFIRST` (most significant bit first) or `LSBFIRST` (least significant bit first).

- Attach (enable) an interrupt service routine (ISR) to a digital input pin.
  - `attachInterrupt(digitalPinToInterrupt(<pin>), <isr>, <type>)`
    - `<pin>` is the pin number, i.e. `0`, `1`, `2`, …
    - `<isr>` is the interrupt service routine (function) to call on interrupt.
    - `<type>` is the type of interrupt.
      - `RISING` triggers the interrupt when the pin transitions from a low to a high state.
      - `FALLING` triggers the interrupt when the pin transitions from a high to a low state.
      - `CHANGE` triggers the interrupt when the pin changes state.
      - `LOW` triggers the interrupt when the pin is in a low state. Not typically used since the interrupt will continue to fire if the pin remains in a low state.
      - `HIGH` triggers the interrupt when the pin is in a high state. Not typically used since the interrupt will continue to fire if the pin remains in a high state.
    - ISR functions should generally be very short and only be used to set global variables (flags) that are checked within the main `loop()`.
    - Global variables changed within an ISR should generally have a datatype size of 8-bits or less, e.g. `bool` or `uint8_t`, and be declared as `volatile` since their values can be changed unexpectedly by the ISR.
- Detach (disable) an interrupt service routine (ISR) from a digital input pin.
  - `detachInterrupt(digitalPinToInterrupt(<pin>))`
    - `<pin>` is the pin number, i.e. `0`, `1`, `2`, …

Basic Blink Sketch Example

```
const uint8_t RedLED = 2;  // reference digital pin 2 as RedLED

void setup() {
   pinMode(RedLED, OUTPUT);  // set RedLED pin as an output
}

void loop() {
   digitalWrite(RedLED, HIGH);  // turn on LED
   delay(1000);                 // wait a second
   digitalWrite(RedLED, LOW);   // turn off LED
   delay(1000);                 // wait a second
}
```

Nonblocking Blink Sketch Example

```
// Reference digital pin 2 as RedLED
const uint8_t RedLED = 2;
// Blinking period in milliseconds
const unsigned long BlinkPeriod = 2000;  // two seconds

void setup() {
   // Set RedLED pin as an output
   pinMode(RedLED, OUTPUT);
}

void loop() {
   // Capture initial previous time
   static unsigned long previousBlinkTime = 0;
   // Capture current time
   unsigned long currentTime = millis();
   // Run every half blinking period (1 second)
   if (currentTime - previousBlinkTime >= BlinkPeriod/2) {
      // Toggle on/off LED
      digitalWrite(RedLED, !digitalRead(RedLED));
      // Save previous time
      previousBlinkTime = currentTime;
   }
}
```

Serial Printing Sketch Example

```
void setup() {
   Serial.begin(9600);  // initialize serial port
   while (!Serial);     // wait for serial connection
}

void loop() {
   int var = 7;

   // Write to serial port and view output in the Serial Monitor
   Serial.print("Printing variable value of ");  // print text
   Serial.print(var);                            // print variable
   Serial.println(" to the Serial Monitor.");    // print text
   delay(1000);                                  // wait a second
}
```

External Hardware Interrupt Sketch Example

```
const uint8_t DeviceBusyPin = 3;           // device's BUSY pin
volatile bool deviceChangedState = false;  // device's status changed
volatile bool deviceIsBusy = true;         // device's current status

void deviceBusyPinChanged() { // interrupt service routine
   deviceChangedState = true;  // set changed flag
   if (digitalRead(DeviceBusyPin) == HIGH) {  // check busy status
      deviceIsBusy = true;  // mark device as busy
   } else {
      deviceIsBusy = false;  // mark device as free
   }
}

void setup() {
   Serial.begin(9600);  // initialize serial port
   while (!Serial);     // wait for serial connection

   pinMode(DeviceBusyPin, INPUT_PULLUP);

   // Attach ISR to detect changes in device's BUSY pin
   attachInterrupt(
      digitalPinToInterrupt(DeviceBusyPin),
      deviceBusyPinChanged, CHANGE);
}

void loop() {
   if (deviceChangedState) {  // check for state changes
      if (deviceIsBusy) {  // wait on device
         Serial.println("Device is busy.");
      } else {  // use device
         Serial.println("Device is free.");
      }
      deviceChangedState = false;  // reset changed flag
   }
}
```

Arduino UNO

General Information

- Power may be supplied to the board in one of three ways: USB, 9V DC jack, or *VIN* pin.
- Digital pins *0* (*RX*) and *1* (*TX*) are connected to the USB on the UNO R3 along with some other boards, e.g. Arduino Mega 2560. Do not use these pins when using the USB or are connected to the Arduino IDE on your computer as it will interfere with proper USB operations and printing to the *Serial Monitor*. The UNO WiFi Rev2 and UNO R4 (Minima and WiFi) boards do not have this limitation.
- The sum of all currents out of all general purpose input/output (GPIO) pins combined must not exceed 200 mA.
- Pins generally default to inputs in a high impedance state.

Resources

- Arduino UNO R3 Documentation
- Arduino UNO R4 Minima Documentation
- Arduino UNO R4 WiFi Documentation

Power Pins

- 5 V pin rated at 500 mA for USB or 1 A with external supply.
- 3.3 V pin rated at 50 mA.

Digital Input/Output Pins

- Twenty digital GPIO pins are available on pins *D0-D19* (*D14-D19* are shared with the *A0-A5* analog pins).
- Pins *D0* and *D1* are used for USB communication on the UNO R3 and should not generally be used on that board.
- Pins are rated 5 V @ 20 mA (absolute maximum of 40 mA) on the UNO R3 and 5 V @ 8 mA max on the UNO R4.
  - If more current is required for an output pin, a transistor may be used to drive the component powered by the *5V* pin or an external supply.
- Six pulse width modulation (PWM) outputs are available on pins *D3*, *D5*, *D6*, *D9*, *D10*, and *D11*.
- Two external hardware interrupts are available on pins *D2* (*EI0*) and *D3* (*EI1*).
- An on-board LED is connected to pin *D13* (*LED_BUILTIN*).
- Pins are standard TTL level compatible.
- Pins are tri-stated when reset.

Analog Input/Output Pins

- Six analog inputs (ADC) are connected to pins *A0-A5*.
- The analog reference is 5 V by default, but can be changed via the *AREF* pin or with the *analogReference()* function.
- 10-bit ADC resolution (0-1023) is the default. Resolutions can be changed to 12-bit (0-4096) or 14-bit (0-16,383) on the UNO R4 with the *analogReadResolution()* function.
- A 12-bit DAC (0-4096) is connected to pin *A0* on the UNO R4.
- An Op-Amp is connected to pins *A1* (+ input), *A2* (- input), and *A3* (output) on the UNO R4.
- Pins are tri-stated when reset.

Communication Pins

- A UART bus is connected to pins *D0* (*RX*) and *D1* (*TX*). Warning, these pins are used for USB communication on the UNO R3 and should not generally be used on that board.
- A 5 V I2C bus is connected to pins *A4* (*SDA*) and *A5* (*SCL*).
- A 3.3 V I2C bus is connected to the Qwiic connector (Stemma QT compatible) on the UNO R4.
- A CAN bus is connected to pins *D4* (*CANTX0*) and *D5* (*CANRX0*) on the UNO R4.
- An SPI bus is connected to pins *D10* (*SS*), *D11* (*COPI*), *D12* (*CIPO*), and *D13* (*SCK*).

Notable Pin Differences Between The UNO R3 And UNO R4

- Digital pins *D0* (*RX*) and *D1* (*TX*) on the R4 boards are not connected to the USB port and are available for general use. Do not use these pins on the R3 when the board is connected to a USB port.
- *CAN* bus is connected to pins *D4* (*CANTX0*) and *D5* (*CANRX0*) on the R4.
- *DAC* is connected to pin *A0* on the R4.
- *OPAMP* is connected to pins *A1* (+ input), *A2* (- input), and *A3* (output) on the R4.
- *I2C* (3.3 V) bus is connected to the Qwiic connector (Stemma QT compatible) on the UNO R4.

Pin-Out (UNO R3 & UNO R4)

- **3.3V** - 3.3 V Regulated Power Output
- **5V** - 5 V Regulated Power Output
- **GND** - Ground
- **GND** - Ground
- **AREF** - Analog Reference Voltage (input 0-5 V)
- **D0/RX** - Digital I/O 0 / UART Receive / USB Port (on R3)
- **D1/TX** - Digital I/O 1 / UART Transmit / USB Port (on R3)
- **D2/EI0** - Digital I/O 2 / External Interrupt 0
- **D3/EI1** - Digital I/O 3 (PWM capable) / External Interrupt 1
- **D4/CANTX0** - Digital I/O 4 / CAN Bus Transmit (on R4)
- **D5/CANRX0** - Digital I/O 5 (PWM capable) / CAN Bus Receive (on R4)
- **D6** - Digital I/O 6 (PWM capable)
- **D7** - Digital I/O 7
- **D8** - Digital I/O 8
- **D9** - Digital I/O 9 (PWM capable)
- **D10/SS** - Digital I/O 10 (PWM capable) / SPI SS
- **D11/COPI** - Digital I/O 11 (PWM capable) / SPI COPI
- **D12/CIPO** - Digital I/O 12 / SPI CIPO
- **D13/SCK/LED_BUILTIN** - Digital I/O 13 / SPI SCK / On Board LED
- **D14/A0/DAC** - Digital I/O 14 / Analog Input 0 / Digital To Analog Converter (on R4)
- **D15/A1/OPAMP+** - Digital I/O 15 / Analog Input 1 / Op-Amp + Input (on R4)
- **D16/A2/OPAMP-** - Digital I/O 16 / Analog Input 2 / Op-Amp - Input (on R4)
- **D17/A3/OPAMPOut** - Digital I/O 17 / Analog Input 3 / Op-Amp Output (on R4)
- **D18/A4/SDA** - Digital I/O 18 / Analog Input 4 / I2C SDA
- **D19/A5/SCL** - Digital I/O 19 / Analog Input 5 / I2C SCL
- **IOREF** - I/O Reference Voltage (tied to 5 V)
- **RESET** - Microcontroller Reset (active low)
- **VIN** - External Power Input (7-12 V, tied to barrel jack input, after a diode)

Revisions

Version	Date	Modifications
1.6	09/06/2023	- Added *Resources* subsections. - Updated *Navigation* section to include additional operations. - Added *Common Operations* subsection. - Added *Serial Printing Sketch* and *External Hardware Interrupt Sketch* examples. - Updated *Arduino UNO* section to include UNO R4 boards and remove UNO WiFi Rev2 board. - Added a *Communication Pins* subsection. - Other minor changes.
1.5	01/18/2023	- Migrated documentation from the legacy IDE 1.X to the new 2.X IDE. - Provided additional information about the Arduino ecosystem. - Created separate Arduino IDE and Sketch Programming sections. - Updated Pin-Out section. - Updated the start-up and shutdown procedures. - Updated URL links. - Other minor changes for clarification and formatting.
1.4	08/18/2021	- Changed *byte* datatype to *uint8_t* in code examples and preferred approach to defining constants. - Added that *D0* and *D1* pins are used for USB communication on other boards as well as the Uno R3. - Added *Notable Pin Differences Between Uno R3 And Uno WiFi Rev2* section. - Updated the classic IDE version. - Other minor changes for clarification.
1.3	01/11/2021	- Added general Arduino platform and IDE descriptions. - Added const versus #define preference. - Updated preprocessor directives example. - Updated bullet formatting. - Updated Woolsey Workshop link in footer. - Other minor changes for clarification.
1.2	10/23/2020	- Changed RedLED constant name in basic Blink sketch example. - Added nonblocking Blink sketch example. - Added language about not using the String datatype. - Updated text coloring and formatting. - Other minor changes.
1.1	02/07/2019	- Added Revisions section. - Removed RX/TX use limitation for Arduino Uno WiFi Rev2 board. - Changed const from `int` to `byte` in LED example. - Added example for selectively compiling source code depending on which board is being used. - Made general wording and formatting changes for clarification.
1.0	12/07/2018	Initial version.