Simblee User Guide

V02.05

Note: This is an alpha version of the documentation. Keep visiting Simblee.com for ongoing updates.

 Copyright © 2016 Simblee Corp.

Contents

Section 1: Introducing Simblee 4

What is Simblee? 4

Simblee Hardware 4

Simblee Software 5

The Simblee Ecosystem 6

Getting Started 6

Simblee GPIOs (and Other Pins) 7

3.3V and GND 8

AREF0 and AREF1 9

Digital Inputs 10

Digital Outputs 10

Analog Inputs 11

Pseudo-Analog (PWM) Outputs 11

The Serial (UART) Port 12

The SPI Bus/Port 12

The I2C Bus/Port 13

General-Purpose Interrupts 15

Using Simblee as a Standalone Microcontroller 16

Section 2: SimbleeForMobile 21

Using a Smartphone or Tablet to Control Simblee via Bluetooth 21

Introducing the `ui()` and `ui_event()` functions 21

The Coordinate System 22

Screen Sizes for Common Mobile Platforms 22

Determining the Screen Width and Height of a Mobile Platform 22

Specifying Portrait or Landscape Modes 23

Giving Your Simblee a Name 25

Displaying a Pre-Defined Graphical Object on the Mobile Platform's Screen 26

Detecting and Using Events Occurring on the Mobile Platform's Screen 27

Appendix A: The Arduino IDE 30

Appendix B: Installing the Simblee Library 30

Installing on a Windows Computer 30

Installing on a MAC Machine 30

 Copyright © 2016 Simblee Corp.

Installing on a Linux Machine 30
Appendix C: RFDuino Shields 30
Appendix D: Locating the Simblee's *variant.h* File 31
On a PC 31
On a Mac 31

 Copyright © 2016 Simblee Corp.

Section 1: Introducing Simblee

What is Simblee?

The term "Simblee" refers to a revolutionary new technology that allows anyone to create "things" (devices) that can be connected to the Internet of Things (IoT). These devices can be used to monitor and control the world around them. Of particular interest is the fact that users can quickly and easily create graphical user interfaces (GUIs) to control these devices, and these GUIs can be presented on mobile platforms like iOS (Apple) and Android smartphones and tablet computers.

Simblee Hardware

Let's start with the Simblee module (Figure 1-2), which is 10mm x 7mm x 2.2mm in size (these modules may also be referred to as Simblee chips or Simblee packages). If you are creating commercial or industrial products from the ground up, then you will almost certainly decide to mount the Simblee module directly on your circuit board.

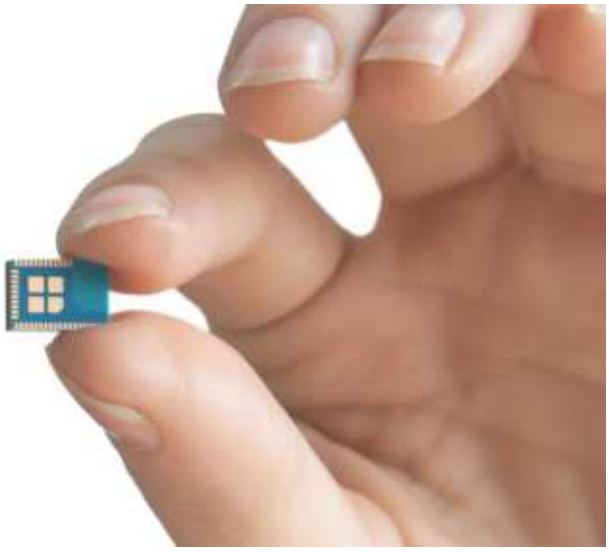


Figure 1-2. The Simblee module is 10mm x 7mm x 2.2mm.

The Simblee module contains two main functions: A 32-bit ARM Cortex-M0 processor (with 128KB of Flash and 24KB of RAM -- no EEPROM -- running at 16MHz) and a Bluetooth Smart Radio.

When it comes to prototyping or hobbyist applications, you may find it more advantageous to use one of the Simblee breakout boards. There are currently two such boards; one offers seven GPIO (general-purpose input/output) pins while the other provides 29 GPIOs (Figure 1-3).

 Copyright © 2016 Simblee Corp.

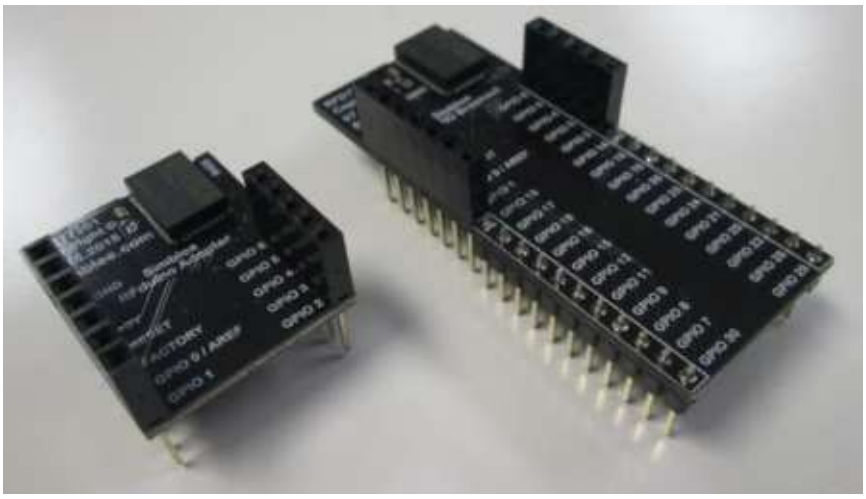


Figure 1-3. The 7 GPIO breakout board (left) and the 29 GPIO board (right).

The part numbers for the 7-GPIO and 29-GPIO breakout boards are RFD77201 and RFD77203, respectively. Each of these breakout boards has a Simblee module (shown in the upper-right-hand corners of the boards in Figure 1-3) along with 0.1" pitch headers on top for connecting shields (see *Appendix C*), and 0.1" pitch pins on the bottom for plugging into breadboards or soldering to printed circuit boards.

Simblee Software

Users create sketches (programs) to run on their Simblees using the Arduino Integrated Development Environment (IDE). These sketches can include user interfaces (UIs), which can be displayed on the user's smartphone or tablet, and which can be used to monitor the Simblee's inputs and control its outputs.

We will be discussing all of these features in more detail shortly. In the context of this introduction, however, we consider the term "Simblee Software" to refer to the free Simblee app that you can download onto your smartphone or tablet computer. If you are using an Apple platform running the iOS operating system, you will download the *Simblee For Mobile* app from the Apple's App Store (support for Android platforms is slated for Q1, 2016).

When you launch the Simblee app on your mobile platform, it will use Bluetooth to announce itself to the surrounding world and ask if any Simblee-enabled devices are in the vicinity. When those devices reply "I'm here," the Simblee app lists their names and descriptions on the screen (from the perspective of the user, this process is essentially instantaneous). Once the user taps the desired item on the screen, the Simblee app sends a message to that device saying "You're up"; the device then uploads its user interface to the Simblee app, which renders it as a graphical user interface (GUI) on the screen. Due to the way in which this works, the Simblee app may also be thought of as being a Simblee Browser or a Simblee Renderer.

As soon as the user returns to the list of Simblee-enabled devices or exits the Simblee app, the user interface associated with the currently selected Simblee-enable device disappears from the mobile platform.

 Copyright © 2016 Simblee Corp.

The Simblee Ecosystem

There are many different facets and usage models associated with Simblee. For example, a Simblee module or breakout board can be programmed using the Arduino IDE and used as a standalone microcontroller. In this case, the Simblee can be visualized as being "Just another flavor of Arduino with a different footprint or form factor."

Of course, the Simblee really comes into its own when it is integrated into a larger ecosystem using its Bluetooth communications capabilities, in which case the following terminologies may apply:

- **SimbleeForMobile:** This encompasses any software, tools, and applications used to control Simblee-enabled devices using a mobile platform such as a smartphone or tablet.
- **SimbleeCOM:** This encompasses any software, tools, and applications used to implement Simblee-to-Simblee communications.
- **SimbleeCloud:** This encompasses any software, tools, and applications used to facilitate communications between Simblee modules and cloud-based databases and tools, including the ability to access data from, and present data to, web pages and sites.

Furthermore, the capabilities of Simblee modules and breakout boards can be augmented with shields. These include Switch/LED, Relay, Servo, microSD, USB, Prototyping, and Power/Battery shields (see Appendix C for more details).

Getting Started

For the purposes of this manual, we are assuming that you are already familiar with the Arduino and the Arduino IDE, and that this IDE is already installed on your computer. If this is not the case, please follow the instructions in *Appendix A*.

Once you have the Arduino IDE installed on your system, the next step is to augment it with the Simblee BSP (board support package) and the Simblee Library. The Simblee BSP allows you to select the Simblee from the selection of supported boards, while the Simblee Library provides a suite of software functions that allow you to create graphical user interfaces on your mobile devices. Instructions for downloading and installing the Simblee BSP and Simblee Library are provided in *Appendix B*.

If you are intending to use a mobile platform such as a smartphone or tablet to control your Simblee-enabled devices, then you need to download and install the free Simblee app onto your mobile platform. If you are using an Apple platform running the iOS operating system, you will download the *Simblee For Mobile* app from the Apple's App Store (support for Android platforms is slated for Q1, 2016).

In order to program your Simblee breakout board, you will need an RFDuino USB Shield (Figure 1-4). This will allow you to download your sketches from the Arduino IDE running on your host computer into the Simblee.

> **NOTE:** The RFDuino shield shown in Figure 1-4 is designed to plug directly into the USB port on your host computer. However, it generally makes one's life a lot easier to use a USB-A (male) to USB-A (female) cable, when the male end of the cable is plugged into your host computer and the USB shield is plugged into the female end of the cable.

 Copyright © 2016 Simblee Corp.



Figure 1-4. The 7 GPIO breakout board (left) and an RFDuino USB shield (right).

The part number for USB shield is RFD22121. If you already have an RFDuino USB shield, then you are ready to rock and roll. Otherwise, visit www.rfduino.com or www.simblee.com to purchase such a shield (see also *Appendix C*).

Simblee GPIOs (and Other Pins)

Figure 1-5 shows the general-purpose input/output (GPIO) pins for the 7-GPIO and 29-GPIO breakout boards. Also shown are the Power (+3.3V), Ground (GND), and Reset pins (the functionality associated with the pin sporting the Factory annotation is outside the scope of this manual).

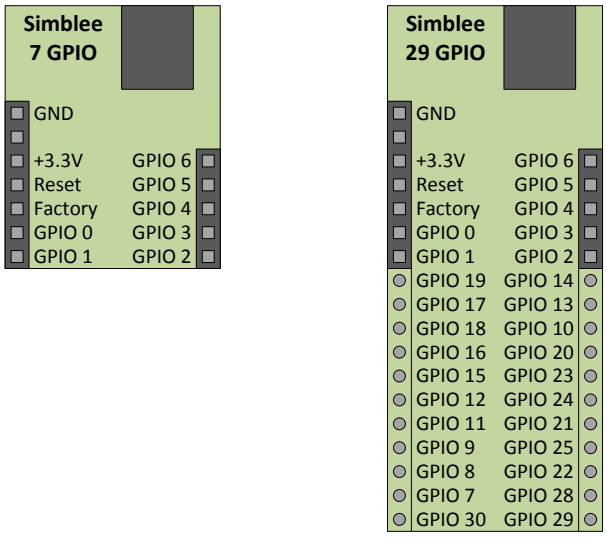


Figure 1-5. Pinouts for the 7 GPIO breakout board (left) and the 29 GPIO board (right).

One point that may be confusing at first is that the 29-GPIO board appears to boast 31 GPIOs numbered 0 through 30. Closer examination, however, reveals that GPIOs 26 and 27 are not exposed (they are used internally), and are therefore not available to the end user. Table 1-1 illustrates the capabilities of the different GPIOs.

 Copyright © 2016 Simblee Corp.

	GPIO#	Analog Reference	Analog Input	Digital In/Out	PWM	Serial (UART)	SPI	I2C
7-GPIO Breakout Board	0	AREF0		Yes	Yes	YES (TX Default)	Yes	Yes
	1		Yes	Yes	Yes	YES (RX Default)	Yes	Yes
	2		Yes	Yes	Yes	Yes	Yes	Yes
	3		Yes	Yes	Yes	Yes	YES (MISO Default)	Yes
	4		Yes	Yes	Yes	Yes	YES (SCK Default)	Yes
	5		Yes	Yes	Yes	Yes	YES (MOSI Default)	YES (SCL Default)
	6	AREF1	Yes	Yes	Yes	Yes	YES (SS Default)	YES (SDA Default)
29-GPIO Breakout Board	7			Yes	Yes	Yes	Yes	Yes
	8			Yes	Yes	Yes	Yes	Yes
	9			Yes	Yes	Yes	Yes	Yes
	10			Yes	Yes	Yes	Yes	Yes
	11			Yes	Yes	Yes	Yes	Yes
	12			Yes	Yes	Yes	Yes	Yes
	13			Yes	Yes	Yes	Yes	Yes
	14			Yes	Yes	Yes	Yes	Yes
	15			Yes	Yes	Yes	Yes	Yes
	16			Yes	Yes	Yes	Yes	Yes
	17			Yes	Yes	Yes	Yes	Yes
	18			Yes	Yes	Yes	Yes	Yes
	19			Yes	Yes	Yes	Yes	Yes
	20			Yes	Yes	Yes	Yes	Yes
	21			Yes	Yes	Yes	Yes	Yes
	22			Yes	Yes	Yes	Yes	Yes
	23			Yes	Yes	Yes	Yes	Yes
	24			Yes	Yes	Yes	Yes	Yes
	25			Yes	Yes	Yes	Yes	Yes
	26	Not exposed to the user						
	27	Not exposed to the user						
	28			Yes	Yes	Yes	Yes	Yes
	29			Yes	Yes	Yes	Yes	Yes
	30			Yes	Yes	Yes	Yes	Yes

Table 1-1. Capabilities of the Simblee's GPIOs.

Additional capabilities and limitations associated with the GPIOs are presented below. Also note that the Simblee supports Serial Wire Debug (SWD), with the SWDIO and SWDCLK signals being presented on the Reset and Factory pins, respectively, but this is outside the scope of this manual.

3.3V and GND

The Simblee requires a 3.3V power supply. If you are using a USB shield, where that shield is plugged into your host computer that is powered up, then this shield will provide the required 3.3V. Alternatively, you may decide to use one of the supported battery shields, whose part numbers are RFD22126, RFD22127, and RFD22128 (see *Appendix C* for more details about the USB and battery shields).

If you are using your Simblee to control another microcontroller that provides 3.3V supply, such as an Arduino Uno, for example, then another alternative is to employ a flying lead to use this supply to power your Simblee.

 Copyright © 2016 Simblee Corp.

NOTE: When you are using an external power supply, and/or when you are connecting any external peripheral, it's also necessary to connect the GND pin on the Simblee with the GND pin(s) on the external device(s).

NOTE: The Simblee's GPIOs output signals between 0V and 3.3V and they expect to see signals between 0V and 3.3V. If you are connecting a 0V to 5V output from a device such as an Arduino Uno to a 0V to 3.3V input on the Simblee, then a level converter is **required**. If you are connecting a 0V to 3.3V output on the Simblee to a 0V to 5V input on an external device, then a level convertor may not be mandatory, but it is **recommended**.

An example device is the 4-bit bidirectional logic level converter from SparkFun (http://bit.ly/1puPKJl). Note that this device has 10KΩ pullup resistors on both sides of each level conversion channel, which means it's also suitable for use with an I2C interface (see also the *I2C Bus/Port* discussions below).

AREF0 and AREF1

When your sketch calls the `analogRead()` function to take an analog reading, it compares the voltage measured on the analog pin against a reference voltage. By default, this reference voltage is the 3.3V supply to the Simblee. The Simblee boasts a 10-bit ADC (analog-to-digital converter), which means it divides the reference voltage into 2^{10} = 1,024 quanta ("slices") numbered from 0 to 1,023. But what if the supply voltage is slightly low (say 3.2V) or high (say 3.4V)? In this case, the accuracy of your analog reading may be compromised.

Another consideration is if whatever is driving the analog pin -- say a sensor -- only provides signals between 0V and 1.55V, for example. In the case of this particular example, we are losing resolution because we are only using half of the available quanta. Also, we may have to scale the values we read in (multiply them by 2X, in the case of this example) if we wish to pretend they map onto a 3.3V range.

The solution to all of these issues is to use an external AREF (**A**nalog **REF**erence) signal. The Simblee allows you to use GPIO 0 or GPIO 6 as external analog references called AREF0 and AREF1, respectively. The AREF inputs can be fed anywhere from 0.83V to 1.3V and you can only use one of these at any particular time.

In order to select the external analog reference, we use the `analogReference()` function as illustrated below:

```
analogReference(EXTERNAL);
```

Similarly, In order to select the internal analog reference, we use the `analogReference()` function as illustrated below:

```
analogReference(INTERNAL);
```

As noted earlier, however, the default is for the Simblee to use its internal reference, so it's only really necessary to use the `INTERNAL` argument if you're previously directed the Simblee to use an external analog reference.

In order to specify AREF1 as being the external analog reference pin, we use the `externalReference()` function prior to using the `analogReference()` function as illustrated below:

```
externalReference(AREF1);
```

 Copyright © 2016 Simblee Corp.

```
analogReference(EXTERNAL);
```

In order to specify AREF0 as being the external analog reference pin, we use the `externalReference()` function as illustrated below:

```
externalReference(AREF0);
```

As noted earlier, however, the default is for the Simblee to use the AREF0 pin as its external reference, so it's only really necessary to use this argument if you're previously directed the Simblee to use its AREF1 pin as the external analog reference.

Digital Inputs

All of the GPIOs (0 through 6 on a 7-GPIO breakout board; 0 through 25 and 28 through 30 on a 29-GPIO breakout board) can be used as digital inputs. Two functions are of interest here: `pinMode()`, which is used to declare the GPIO as being a digital input, and `digitalRead()`, which is used to read the value on the pin as illustrated below:

```
pinMode (pin, INPUT);    // Use INPUT_PULLUP if you want
                         // to apply an internal pullup
int i;

i = digitalRead(pin);
```

Where `pin` is an integer constant or variable between 0 and 30 (depending on the breakout board and excluding 26 and 27 as discussed above) and the `digitalRead()` function returns a value of 0 (LOW) or 1 (HIGH).

The `digitalRead()` function may be called multiple times throughout the sketch. By comparison, the `pinMode()` function is typically used only once per digital input, and these function calls typically appear only in the `setup()` function. Having said this, the `pinMode()` function may be invoked multiple times for the same GPIO if it is required to change that GPIO from a digital input to a digital output (or vice versa), or to change a pseudo-analog (PWM) output to a regular digital output, for example (see also the discussions on *Pseudo-Analog (PWM) Outputs* below).

Digital Outputs

All of the GPIOs (0 through 6 on a 7-GPIO breakout board; 0 through 25 and 28 through 30 on a 29-GPIO breakout board) can be used as digital outputs. Two functions are of interest here: `pinMode()`, which is used to declare the GPIO as a digital output, and `digitalWrite()`, which is used to output a value onto the pin as illustrated below:

```
pinMode (pin, OUTPUT);

digitalWrite(pin,value);
```

Where `pin` is an integer constant or variable between 0 and 30 (depending on the breakout board and excluding 26 and 27 as discussed above), and `value` is set to `LOW` (or 0) or `HIGH` (or 1).

 Copyright © 2016 Simblee Corp.

The `digitalWrite()` function may be called multiple times throughout the sketch. By comparison, the `pinMode()` function is typically used only once per digital output, and it typically appears only in the `setup()` function. Having said this, the `pinMode()` function may be invoked multiple times for the same GPIO if it is required to change that GPIO from a digital input to a digital output (or vice versa), or to change a pseudo-analog (PWM) output to a regular digital output, for example (see also the discussions on *Pseudo-Analog (PWM) Outputs* below).

Analog Inputs

GPIOs 1 through 6 can be used as analog inputs by means of the `analogRead()` function as illustrated below:

```
int i;

i = analogRead(pin);
```

Where `pin` is an integer constant or variable between 1 and 6, and the `analogRead()` function returns a value between 0 and 1,023. (It is not necessary to call the `pinMode()` function prior to using the `analogRead()` function.)

Pseudo-Analog (PWM) Outputs

All of the GPIOs (0 through 6 on a 7-GPIO breakout board; 0 through 25 and 28 through 30 on a 29-GPIO breakout board) can be used as pseudo-analog (PWM) outputs by means of the `analogWrite()` function as illustrated below:

```
analogWrite(pin,value);
```

Where `pin` is an integer constant or variable between 0 and 30 (depending on the breakout board and excluding 26 and 27 as discussed above), and `value` is an integer (or byte) constant or variable between 0 and 255. (It is not necessary to call the `pinMode()` function prior to using the `analogWrite()` function).

> **NOTE:** Although all of the GPIOs are capable of being used as PWM outputs, only four PWM outputs can be supported at any particular time.

Although only four PWM outputs can be supported at any particular time, it is possible to change the mode of a GPIO during the course of a sketch as illustrated below:

```
analogWrite(10, 31);  // Set GPIO 10 running as a PWM with a value of  31
analogWrite(11, 63);  // Set GPIO 11 running as a PWM with a value of  63
analogWrite(12,127);  // Set GPIO 12 running as a PWM with a value of 127
analogWrite(13,255);  // Set GPIO 13 running as a PWM with a value of 255

delay(1000);          // Do something (wait for 1 second in this example)

pinMode(12,OUTPUT);   // Reassign pin 12 from PWM to regular digital output
digitalWrite(12,LOW); // Drive pin 12 LOW (0) or HIGH (1)

analogWrite(16,191);  // Set GPIO 13 running as a PWM with a value of 191
```

 Copyright © 2016 Simblee Corp.

In the above example, we start by declaring four GPIOs (10, 11, 12, and 13) as being pseudo-analog (PWM) outputs. This is the maximum number of PWM outputs we can support. Later, however, we change GPIO to be a regular output, and then we declare GPIO 16 as being a new PWM output. Depending on the application, this technique can allow a Simblee to appear to support more than four PWM outputs.

The Serial (UART) Port

The Simblee supports one hardware Serial (UART) port. By default, its TX (Transmit) and RX (Receive) signals are mapped onto GPIOs 0 and 1, respectively. This port cannot be used by the user -- except for communications with the Arduino IDE's Serial Monitor -- whilst the USB shield is attached. When the USB shield is not attached, however, the port can be used to communicate with a wide variety of peripherals.

In order to use this port with the default pins, simply use the `Serial.begin()` function as illustrated below:

```
Serial.begin(speed);

Serial.println("Hello World!");
```

Where `speed` is typically one of the following "standard" values, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. We can also use the `Serial.begin()` function to reassign the Serial port to other GPIOs as illustrated below:

```
Serial.begin(speed,RXpin,TXpin);
```

Where `RXpin` and `TXpin` can be any of the available GPIOs.

NOTE: If the Serial port is reassigned to GPIOs other than 0 and 1, then it will no longer be possible for the Simblee to communicate with the Arduino IDE's Serial Monitor.

NOTE: If GPIO 0 and/or GPIO 1 are assigned to be digital or analog inputs or outputs, or if they are reassigned to the SPI or I2C port as discussed below, or if the external analog reference AREF0 is activated on GPIO 0 as discussed above, then the following will apply:

1) Even if the USB shield is still attached, it will no longer be possible for the Simblee to communicate with the Arduino IDE's Serial Monitor.
2) While the USB shield remains attached, the newly assigned pins (or AREF0) will not function as expected or required.

NOTE: Even if the Serial port is reassigned to GPIOs other than 0 and 1, and/or other functions are assigned to GPIOs 0 and 1, the USB shield can still be used to re-program the Simblee. When the user instructs the Arduino IDE to upload a new sketch into the Simblee, the system first activates the Simblee's `Reset` signal, which causes the Simblee to activate its bootloader. In turn, the bootloader automatically assigns the Serial port to GPIOs 0 and 1.

The SPI Bus/Port

The Serial Peripheral Interface (SPI) bus is a synchronous serial communication interface specification used for short distance communication. You can use this bus to allow your Simblee to communicate with peripheral devices.

 Copyright © 2016 Simblee Corp.

The Simblee supports one hardware SPI port. By default, its SCK (System Clock), SS (Slave Select), MOSI (Master-Out, Slave In), and MISO (Master In, Slave Out) signals are mapped onto GPIOs 4, 6, 5, and 3, respectively.

If required, these signals can be mapped onto any of the Simblee's GPIOs. If you open the *variant.h* file in the *variants**Simblee* folder, you will see the following definitions:

```
#define PIN_SPI_SCK  (4u)
#define PIN_SPI_SS   (6u)
#define PIN_SPI_MOSI (5u)
#define PIN_SPI_MISO (3u)
```

NOTE: Consult *Appendix D* for help in locating the *variant.h* file.

If you are working with a 29-GPIO breakout board and you wish to reassign the SCK, SS, MOSI, and MISO signals to GPIOs 14, 15, 16, and 17, respectively, then you would modify these definitions to read as follows:

```
#define PIN_SPI_SCK  (14u)
#define PIN_SPI_SS   (15u)
#define PIN_SPI_MOSI (16u)
#define PIN_SPI_MISO (16u)
```

NOTE: You should exit the Arduino IDE before modifying the *variant.h* file. If you modify the *variant.h* file while the IDE is open, then you may have to exit the IDE and re-launch it in order for any changes to take effect.

NOTE: Unlike the I2C bus, no pullup resistors are required on the SPI bus.

The SPI library is included with the Arduino-Simblee environment, so there is no need for you to install anything. This library only supports "Simblee as Master" scenarios. The use of the SPI bus is beyond the scope of this manual, so we will limit ourselves to the following minimalist example:

```
#include <SPI.h>

void setup() {
  SPI.begin();
}

void loop() {
  // Put your main code here, to run repeatedly
}
```

The reason for our showing this example is to illustrate the fact that it's only when the `SPI.begin()` function is executed that the GPIOs defined in the *variant.h* file are assigned to the SPI bus.

The I2C Bus/Port

The Inter-Integrated Circuit (I^2C, I2C, or IIC) bus is a multi-master, multi-slave, single-ended, serial bus that is typically used for attaching lower-speed peripheral integrated circuits (ICs) to processors and microcontrollers.

 Copyright © 2016 Simblee Corp.

In the original documentation, I^2C was always written with the '2' as a superscript, so some people pronounce this as "eye-squared-see." However, it's also common to see I2C in which the '2' is presented in the same font and size as the 'I' and the 'C,' so some people pronounce it "eye-two-see." Finally, if the term is written IIC, it may be pronounced "eye-eye-see." (If it doesn't work, it may be pronounced B^%$#@!).

The Simblee supports one hardware I2C port, and -- at the time of this writing -- this port only supports "Simblee as Master" scenarios. The I2C library is included with the Arduino-Simblee environment, so there is no need for you to install anything. The Simblee supports the following standard I2C functions:

```
Wire.begin();
Wire.beginTransmission();
Wire.available();
Wire.requestFrom();
Wire.send();
Wire.receive();
Wire.endTransmission();
```

The use of the I2C bus is beyond the scope of this manual, so we will limit ourselves to the following minimalist example:

```
#include <Wire.h> // The I2C library

void setup() {
  Wire.begin();
}

void loop() {
  // Put your main code here, to run repeatedly
}
```

The reason for our showing this example is to illustrate the fact that it's only when the `Wire.begin()` function is executed that the relevant GPIOs are assigned to the I2C bus.

> **NOTE:** Unlike the SPI bus, pullup resistors are required on the I2C bus. This is because the pins on any of the devices connected to the I2C bus (including the Simblee and any sensors or peripherals) are of a type known as open collector. Ideally, these pullup resistors should be 4.7kΩ in value, and there should be only one pair for the whole bus. In practice, however, some shields come equipped with pullup resistors pre-installed. In this case, it's common for these resistors to be 10kΩ in value, thereby ensuring that if two such shields are connected to the bus, the shared (parallel) resistance will end up being 5kΩ.

By default, the I2C's SCL (Serial Clock Line) and SDA (Serial Data Line) are mapped onto GPIOs 5 and 6 respectively. However, the Simblee also supports the `Wire.beginsOnPins()` function, which allows you to reassign the I2C signals to any of the Simblee's GPIOs. For example, consider the following code snippet:

```
#include <Wire.h> // The I2C library

void setup() {
  Wire.beginOnPins(SCLpin,SDApin);
```

 Copyright © 2016 Simblee Corp.

```
}

void loop() {
  // Put your main code here, to run repeatedly
}
```

Where `SCLpin` and `SDApin` can be any of the available GPIOs.

General-Purpose Interrupts

All of the GPIOs (0 through 6 on a 7-GPIO breakout board; 0 through 25 and 28 through 30 on a 29-GPIO breakout board) can be used as general-purpose interrupts by means of the `attachPinInterrupt()` function (this is equivalent to the Arduino's `attachInterrupt()` function) as illustrated below:

```
attachPinInterrupt(pin, callback, mode);
```

Where `pin` is an integer constant or variable between 0 and 30 (depending on the breakout board and excluding 26 and 27 as discussed above), `callback` is the name of the Interrupt Service Routine (ISR), and `mode` is set to `LOW` (or 0) or `HIGH` (or 1).

In the case of the `LOW` mode, if the interrupt pin is in a `HIGH` (1) state, the interrupt will trigger when the pin is brought `LOW`. By comparison, in the case of the `HIGH` mode, if the interrupt pin is in a `LOW` (0) state, the interrupt will trigger when the pin is brought `HIGH`.

> **NOTE:** There is no limit to the number of GPIOs that you can use as general-purpose interrupts at any particular time.

The in-depth requirement for, and use of, interrupts and timers is beyond the scope of this manual, but it's always worth looking at an example in order to wrap one's brain around things. Let's assume a simple scenario in which we wish to activate a LED attached to GIPO 6 if the signal driving GPIO 7 (which is configured as a digital input) is brought LOW. A sketch to implement this scenario is as illustrated below:

Sketch 1-1. A simple interrupt

```
int ledPin       = 6;
int interruptPin = 7;

void setup() {
  pinMode (ledPin, OUTPUT);
  digitalWrite(ledPin,LOW);

  pinMode (interruptPin, INPUT);
  attachPinInterrupt(interruptPin, letsDoIt, LOW);
}

void loop() {
  // Put your main code here, to run repeatedly
}

int letsDoIt(uint32_t dummyPin) {  // See note below
  digitalWrite(ledPin,HIGH);
```

 Copyright © 2016 Simblee Corp.

```
  return 0;                          // See note below
}
```

NOTE: Interrupt Service Routines (ISRs) should be kept as tight (short and fast) as possible. In the case of the Arduino, ISRs cannot have any parameters, they should be declared as being of type `void`, and they shouldn't return any values. By comparison, the Simblee's ISR functionality is shared with waking up from its Ultra-Low-Power (ULP) Sleep Mode. This means that -- as illustrated in Sketch 1-1 -- even general-purpose ISRs have to be declared as being of type `int` and they have to return a dummy value of 0. Furthermore, they have to be declared with a `dummyPin` parameter of type `uint32_t` (this parameter is not used within the body of the general-purpose ISR).

By default, any interrupts established using the `attachPinInterrupt()` function are enabled (active) in a sketch (they are automatically disabled whenever an ISR is being processed). If you wish to disable an individual interrupt, you can do so using the `detachPinInterrupt()` function (this is equivalent to the Arduino's `detachInterrupt()` function) as illustrated below:

```
detachPinInterrupt(interruptPin);
```

If you subsequently wish to re-enable this interrupt, you can do so by calling the `attachPinInterrupt()` function again. Alternatively, if you wish to disable all of the currently-defined interrupts, you can do so by means of the `noInterrupts()` function as illustrated below:

```
noInterrupts();
```

If you subsequently wish to re-enable all of the currently-defined interrupts, you can do so by means of the `interrupts()` function as illustrated below:

```
interrupts();
```

In this case, there is no need to reuse the `attachPinInterrupt()` function.

NOTE: The Arduino's `RISING` , `FALLING`, and `CHANGE` modes are not supported.

NOTE: The Arduino's `sei()` and `cli()` functions are not supported, but the Simblee's `interrupts()` and `noInterrupts()` functions perform the same tasks, respectively.

See also the discussions on the Simblee's Ultra-Low Power (ULP) mode -- putting the Simblee in ULP mode and waking it from this mode -- elsewhere in this manual.

Using Simblee as a Standalone Microcontroller

In engineering, it's usually a good idea to take things one step at a time and to make sure that any foundation functionalities are working as expected before treading new ground. On this basis, before we leap into using mobile platforms to control our Simblees, it's probably a good idea to prove that our Arduino IDE ↔ Simblee environment and flow functions as expected.

For the purposes of this example, we are going to use a 7-GPIO Simblee breakout board in conjunction with a standard breadboard. We're going to connect a 10KΩ linear potentiometer to one of the Simblee's analog inputs and a LED to one of its pseudo-analog (PWM) outputs as illustrated in Figure 1-6.

Next, we're going to create a sketch that loops around reading the value from the Simblee's analog input and usies this value to control the PWM output, thereby controlling the brightness of the LED. Even if you don't wish to go to the trouble of building this test circuit, it would still be a good idea to create he sketch and download it into your Simblee so as to verify that the Arduino IDE and your Simblee can "talk" to each other.

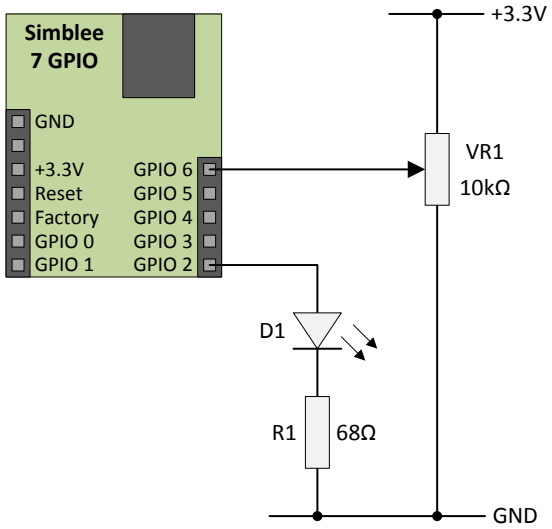


Figure 1-6. Using a potentiometer to control a LED (schematic).

NOTE: The value of R1 in Figure 1-6 is calculated as follows. We are assuming that the LED has a forward (On) voltage drop (V_F) of 2.0V and a forward current (I_F) of 20mA (0.02A). Using Ohm's law of V = I*R, we know that (3.3V - 2.0V) = 0.02A*R. This means R = 1.3V/0.02A = 65Ω. The closest standard 5% tolerance resistor to this value is 68Ω, so that's what we'll use.

Now let's consider a Fritz-style diagram of this circuit as illustrated in Figure 1-7. If we had already loaded the sketch into the Simblee and wished to use the 7-GPIO board on its own, then we would also require external 3.3V (power) and GND (ground) connections. For the purposes of this example, however, we will be using a USB shield (not shown in Figure 1-7), and this will provide the required power and ground connections.

 Copyright © 2016 Simblee Corp.

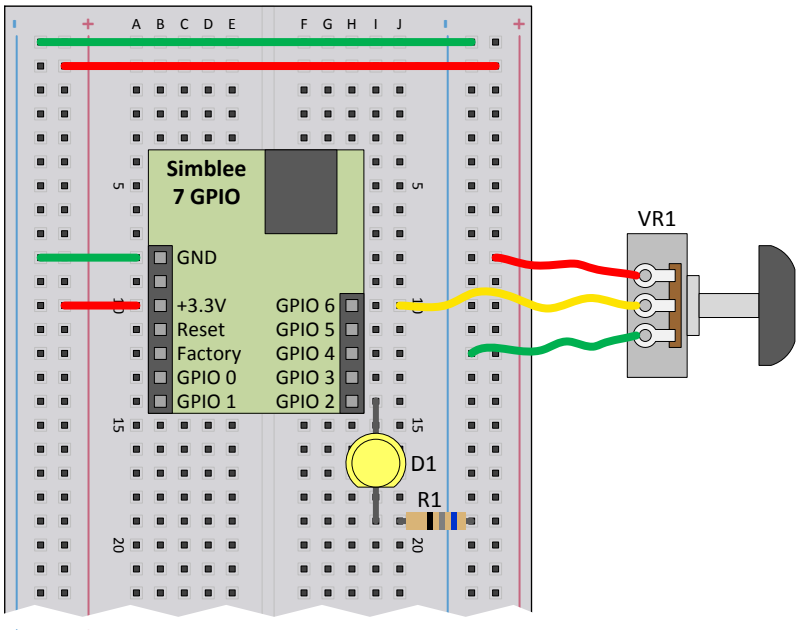


Figure 1-7. Using a potentiometer to control a LED (Fritz-style diagram).

The actual breadboard realization of this circuit is shown in Figure 1-8. The 7-GPIO breakout board is hidden underneath the USB shield, which is itself connected to the host computer via a USB-A (male) to USB-A (female) cable.

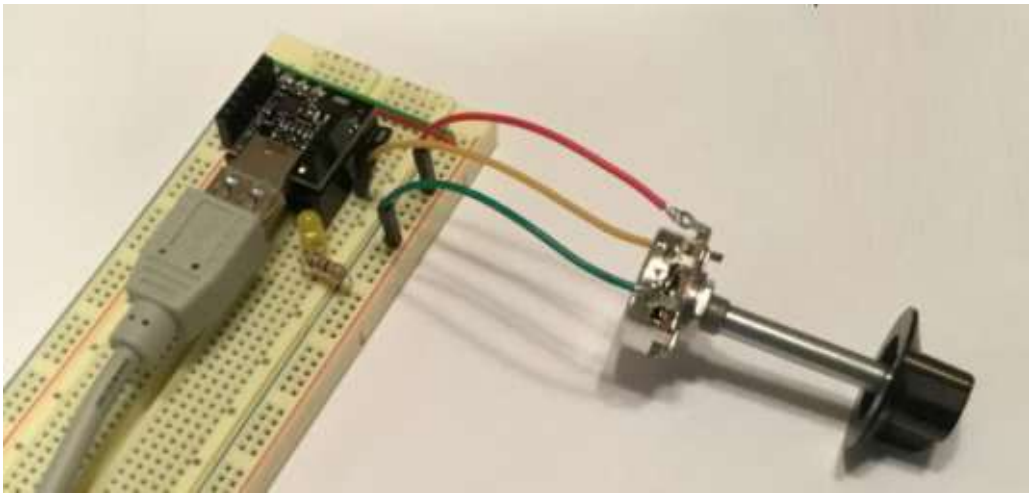


Figure 1-8. Using a potentiometer to control a LED (breadboard implementation).

OK, so now let's use the Arduino IDE to capture our sketch. This assumes that you've already got the IDE installed along with the Simblee BSP (board support package) and the Simblee Library (see also *Getting Started* earlier in this manual).

Launch the Arduino IDE and use the **Tools → Board** pulldown menu to select the Simblee board. Now, plug the cable connected to the USB shield into your host computer.

> **NOTE:** If your host computer is a PC, when you first plug in the USB cable, wait for the "bing" sound that informs you that the computer sees this device, and then use the **Tools → Port** pulldown menu to select whichever COM port is driving the Simblee. By comparison, if your

 Copyright © 2016 Simblee Corp.

host computer is a Mac, do *not* wait for a "bing" sound because there won't be one. Also, the port on a Mac is not a COM port; instead, it's a "/dev/cu/usbserial-xxxxxxxx" port.

NOTE: The "Simblee" entry in the **Tools → Board** pulldown menu is used for both the 7-GPIO and 29-GPIO breakout boards. This is because the boards are functionally identical apart from the number of GPIOs they support.

Now it's time to enter Sketch 1-2 into the IDE. This is obviously a very simple program, but it serves to demonstrate a number of things, not the least that -- in standalone mode -- the Simblee acts like "just another flavor" of the Arduino microprocessor development boards we've all grown to know and love.

For example, we see that the `Serial.begin()`, `Serial.print()`, and `Serial.println()` functions work in the same way as for the regular Arduino, facilitating communication between the Simblee and the IDE's Serial Monitor window.

Sketch 1-2. Using the a potentiometer to control a LED (Simblee acting as standalone microcontroller)

```
int potPin = 6;       // Analog input from potentiometer
int ledPin = 2;       // PWM output driving LED

void setup() {
  Serial.begin(9600); // Initialize serial communications
}

void loop() {

  int potValue;

  potValue = analogRead(potPin); // Read analog input
  potValue = potValue / 4;       // Scale value for PWM output

  Serial.print("PWM Value = ");  // Display PWM value on Serial Monitor
  Serial.println(potValue);

  analogWrite(ledPin,potValue);  // Use the PWM value to control the LED

  delay(100);
}
```

The Simblee boasts a 10-bit ADC (analog-to-digital converter), which means the `analogRead()` function returns values between 0 to 1,023 in decimal. Meanwhile, the Simblee's four 8-bit PWMs accept values between 0 to 255 in decimal. This explains why we scale the `potValue` acquired from the potentiometer by dividing it by four before passing it to the PWM. We could, of course, have achieved the same effect by shifting `potValue` two bits to the right, or by using the standard `map()` function.

Once you've entered this sketch, save it (always save things), verify that it compiles, download it into your Simblee, and open the IDE's Serial Monitor. Rotating the potentiometer will cause the brightness of the LED to vary and the values displayed in the Serial Monitor to look something like those shown in Figure 1-9.

 Copyright © 2016 Simblee Corp.

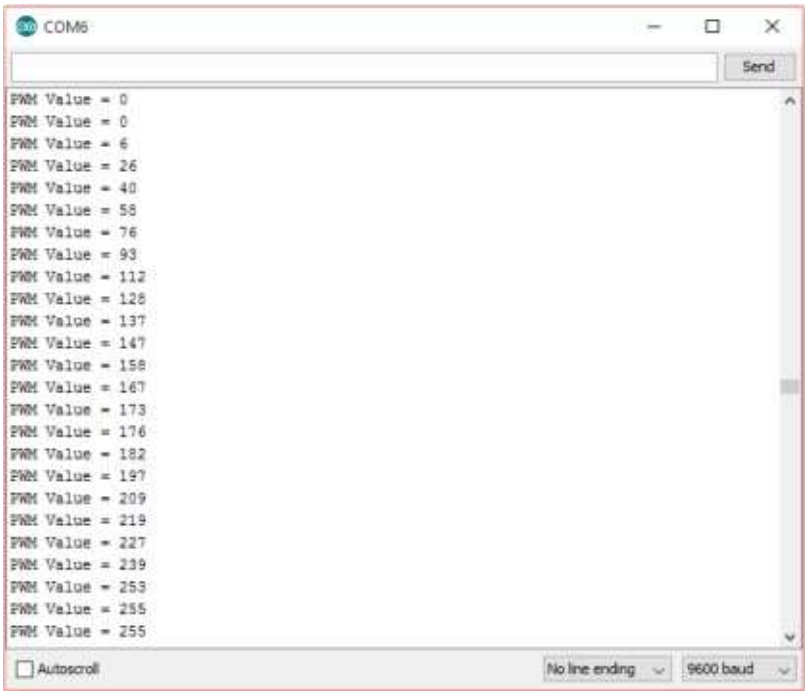


Figure 1-9. Using a potentiometer to control a LED (breadboard).

OK, now we're really ready to rock and roll. Hold onto your hat because we're about to embark on a rollicking roller coaster of discovery that will make you squeal in delight and -- if you don’t take appropriate precautions -- blow your socks off (so, before reading further, make sure you're wearing the elasticated kind)!

 Copyright © 2016 Simblee Corp.

Section 2: SimbleeForMobile

Using a Smartphone or Tablet to Control Simblee via Bluetooth

Introducing the `ui()` and `ui_event()` functions

A simple Arduino sketch includes two functions called `setup()` and `loop()` as illustrated in Sketch 2-1. Of course, users can create their own functions in addition to `setup()` and `loop()`.

Sketch 2-1. A simple Arduino sketch

```
void setup() {
  // Put your setup code here, to run once
}

void loop() {
  // Put your main code here, to run repeatedly
}
```

Similarly, a simple SimbleeForMobile sketch contains four functions. In addition to the traditional `setup()` and `loop()` functions, we also have the `ui()` and `ui_event()` functions as illustrated in Sketch 2-2.

Sketch 2-2. A simple SimbleeForMobile sketch

```
void setup() {
  // Put your setup code here, to run once
}

void loop() {
  // Put your main code here, to run repeatedly
}

void ui() {
  // Put your user interface code here
}

void ui_event() {
  // Put your event-related code here
}
```

The `ui()` function is where the majority of the SimbleeForMobile User Interface (UI) is defined in textual format. This is the interface that will be uploaded to the mobile platform, where it will be rendered (presented) as a Graphical User Interface (GUI). The `ui_event()` function provides the callback mechanism by which the GUI on the mobile platform can communicate any actions back into the body of the sketch.

 Copyright © 2016 Simblee Corp.

The Coordinate System

SimbleeForMobile uses an XY coordinate system whose origin is located in the upper-left-hand corner of your mobile platform's screen, with X values increasing to the right and Y values increasing downward.

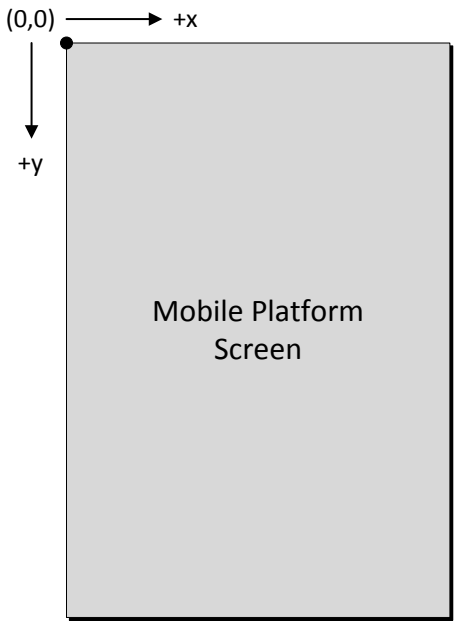


Figure 2-1. The coordinate system used for mobile device screens.

For the purposes of this manual, X and Y values are specified in terms of "units" instead of "pixels" because different devices can have varying definitions as to what actually constitutes a pixel.

Screen Sizes for Common Mobile Platforms

Table 2-1 shows the screen sizes (specified in "units" as discussed above) for a variety of common platforms.

Device	Width (Units)	Height (Units)
iPhone 6	320	568

Table 2-1. Screen sizes for common mobile platfrms.

Determining the Screen Width and Height of a Mobile Platform

If your mobile platform isn't represented in Table 2-1, you can use the `SimbleeForMobile.screenWidth` and `SimbleeForMobile.screenHeight` values to determine the actual width and height of your display. These values are automatically set once you are connected with a device using SimbleeForMobile, at which point you can display them in the Arduino IDE's Serial Monitor window using Sketch 2-3.

Sketch 2-3. Determining the screen width and height of a mobile platform

```
#include <SimbleeForMobile.h>

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

 Copyright © 2016 Simblee Corp.

```
  SimbleeForMobile.begin();
}

void loop() {
  SimbleeForMobile.process();
  Serial.print("Screen Width: ");
  Serial.print(SimbleeForMobile.screenWidth);
  Serial.print(" Screen Height: ");
  Serial.println(SimbleeForMobile.screenHeight);
}

void ui() {
  // Put your user interface code here
}

void ui_event() {
  // Put your event-related code here
}
```

The `SimbleeForMobile.begin()` function initializes SimbleeForMobile and sets everything running. The `SimbleeForMobile.process()` function does something really interesting, but I can't remember what that something is at the moment.

When you upload Sketch 2-3 into your Simblee and open the Serial Monitor window in the Arduino IDE (with the USB cable still connecting the Simblee to your host computer), you will see something like the following:

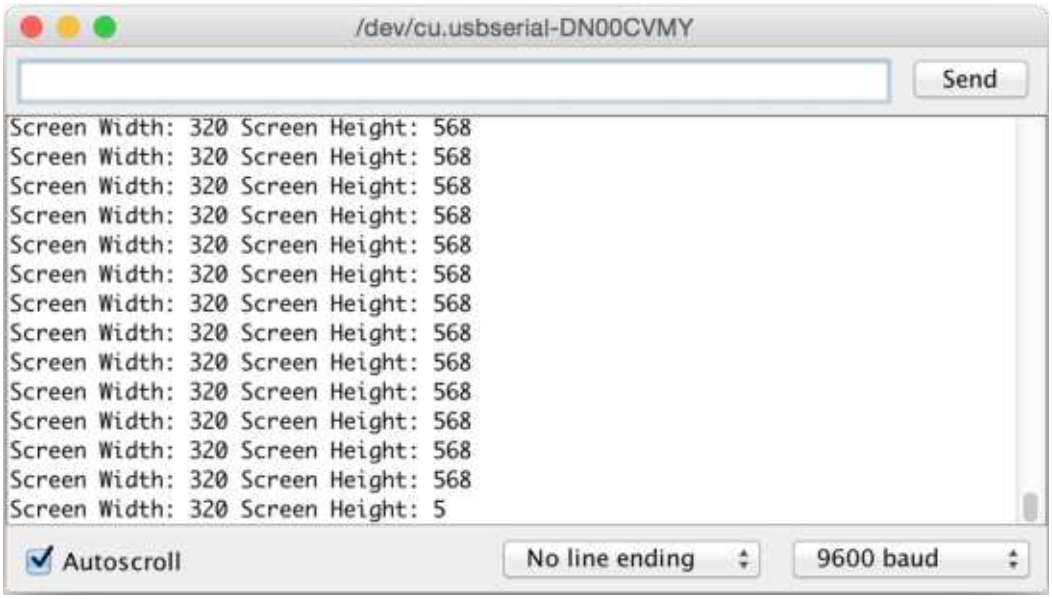
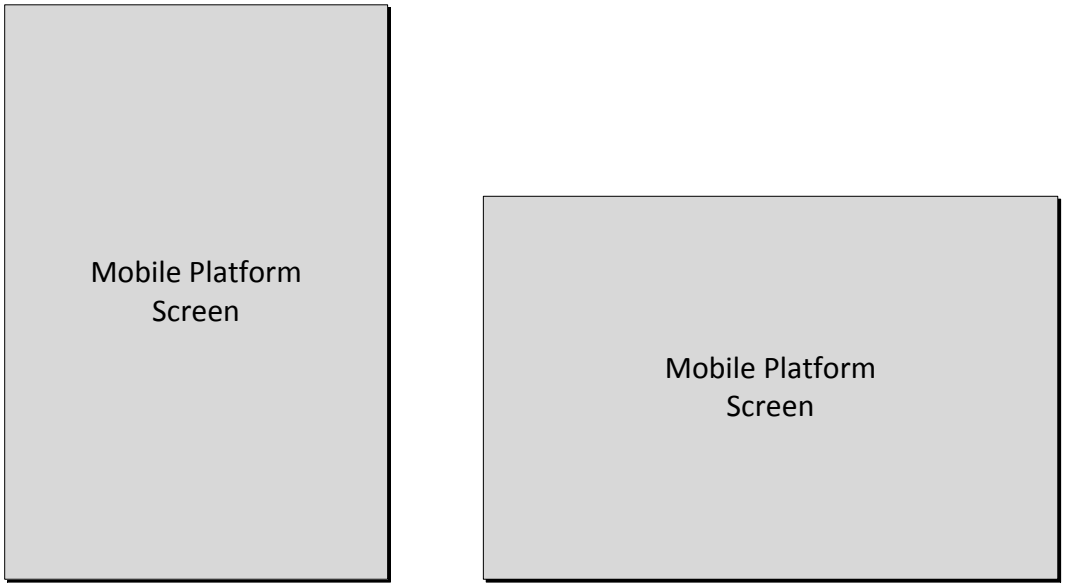


Figure 2-2. Displaying the width and height of the mobile platform's screen.

Specifying Portrait or Landscape Modes

The way in which a rectangular display is presented for viewing is referred to as its orientation. For historical reasons, based on the way in which painters typically presented images of people and landscapes, the two most common types of orientation are referred to as *portrait* and *landscape* as illustrated in Figure 2-3.



(a) Portrait orientation (b) Landscape orientation

Figure 2-3. Portrait versus landscape modes.

SimbleeForMobile can display your UI in either portrait or landscape mode. You can specify this explicitly using the `SimbleeForMobile.beginScreen()` function as illustrated in Sketches 2-4 and 2-5 (if no mode is specified, the portrait mode will be used by default).

Sketch 2-4. Setting the screen to portrait mode

```
#include <SimbleeForMobile.h>

void setup() {
  SimbleeForMobile.begin();
}

void loop() {
  SimbleeForMobile.process();
}

void ui(){
  SimbleeForMobile.beginScreen(WHITE, PORTRAIT);
  SimbleeForMobile.endScreen();
}

void ui_event() {
  // Put your event-related code here
}
```

Sketch 2-5. Setting the screen to landscape mode

```
#include <SimbleeForMobile.h>

void setup() {
  SimbleeForMobile.begin();
}
```

 Copyright © 2016 Simblee Corp.

```
void loop() {
  SimbleeForMobile.process();
}

void ui(){
  SimbleeForMobile.beginScreen(WHITE, LANDSCAPE);
  SimbleeForMobile.endScreen();
}

void ui_event() {
  // Put your event-related code here
}
```

The `SimbleeForMobile.beginScreen()` and `SimbleeForMobile.endScreen()` functions are used to encapsulate all of the definitions for the objects that are to form part of the user interface.

With regard to the `WHITE` parameter used in Sketches 2-4 and 2-5, SimbleeForMobile defines the following color constants: `RED`, `GREEN`, `BLUE`, `YELLOW`, `MAGENTA`, `CYAN`, `WHITE`, `GRAY`, and `CLEAR` (transparent). Additional colors can be defined as required (this is discussed elsewhere in this document).

Giving Your Simblee a Name

You can specify the name of your Simblee (i.e., the name that will appear in the list of "Found Simblees" on your mobile platform's screen) by assigning a text string to the `SimbleeForMobile.deviceName` parameter as illustrated in Sketch 2-6. Also observe that you can add a short advert (or description) by assigning a text string to the `SimbleeForMobile.advertisementData` parameter.

Sketch 2-6. Giving your Simblee a name

```
#include <SimbleeForMobile.h>

void setup() {
  SimbleeForMobile.deviceName = "Hello";         // Device name
  SimbleeForMobile.advertisementData = "World"; // Advert or Description
  SimbleeForMobile.begin();
}

void loop() {
  SimbleeForMobile.process();
}

void ui(){
  SimbleeForMobile.beginScreen(WHITE, PORTRAIT);
  SimbleeForMobile.endScreen();
}

void ui_event() {
  // Put your event-related code here
}
```

NOTE: You can specify up to 15 characters between the device name and advertising data.

 Copyright © 2016 Simblee Corp.

NOTE: The `deviceName` and `advertisementData` values must be assigned *before* the `SimbleeForMobile.begin()` function is executed.

Observe the resulting "Hello" and "World" text strings as they appear in the "Found Simblees" list on an iPhone 6 as illustrated in Figure 2-4 (note that only one Simblee is present in this example).

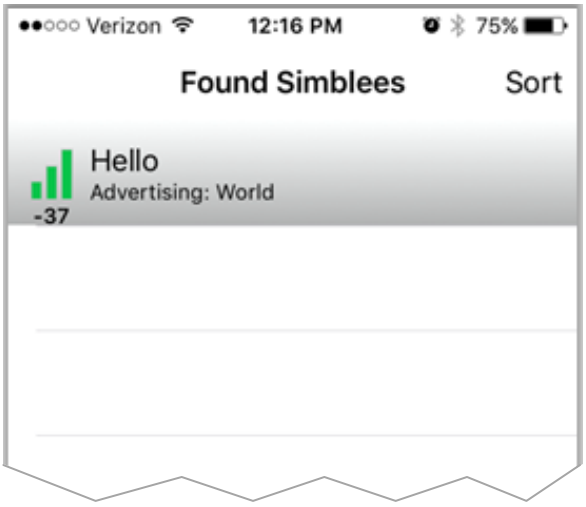


Figure 2-4. Device name and advertisement data in the "Found Simblees" list.

Displaying a Pre-Defined Graphical Object on the Mobile Platform's Screen

For the purposes of this simple example, we are going to create a sketch that displays a switch on your mobile platform's screen. (Note that all we are going to do here is draw a switch and play with it a bit; we will discuss things like adding annotations to the switch and changing its "On" color later in this manual.) As is illustrated in Figure 2-5, the origin of the switch graphic is in its top left-hand corner, and the width and height values of the switch object are 51 and 31 units respectively.

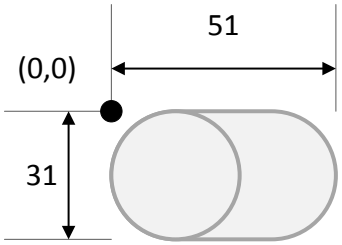


Figure 2-5. The graphical switch object.

In order to add the switch to our user interface (UI), we call the `SimbleeForMobile.drawSwitch()` function as illustrated in Sketch 2-7.

Sketch 2-7. Drawing a switch

```
#include <SimbleeForMobile.h>
```

 Copyright © 2016 Simblee Corp.

```
void setup() {
  SimbleeForMobile.deviceName = "Switch"; // Name of Simblee
  SimbleeForMobile.begin();
}

void loop() {
  SimbleeForMobile.process();
}

void ui() {
  SimbleeForMobile.beginScreen(WHITE,PORTRAIT);
  SimbleeForMobile.drawSwitch(135,100); // X= 135 units; Y = 100 units
  SimbleeForMobile.endScreen();
}

void ui_event() {
  // Put your event-related code here
}
```

In this example, we are drawing the switch at XY location (135,100), where these values are measured from the origin of the screen to the origin of the switch, as illustrated in Figure 2-6.

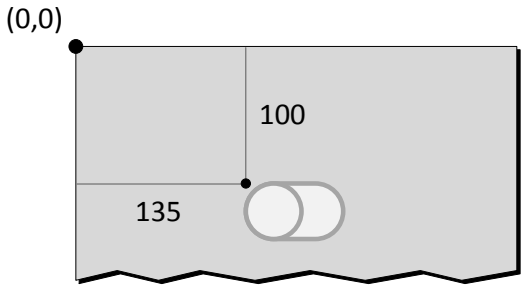


Figure 2-6. The location of the switch on the screen.

In Figure 2-7 we see actual screen shots of the way in which our switch will appear on the screen of an iPhone 6 after we've uploaded our sketch into a Simblee and then selected this device from the "Found Simblees" list.

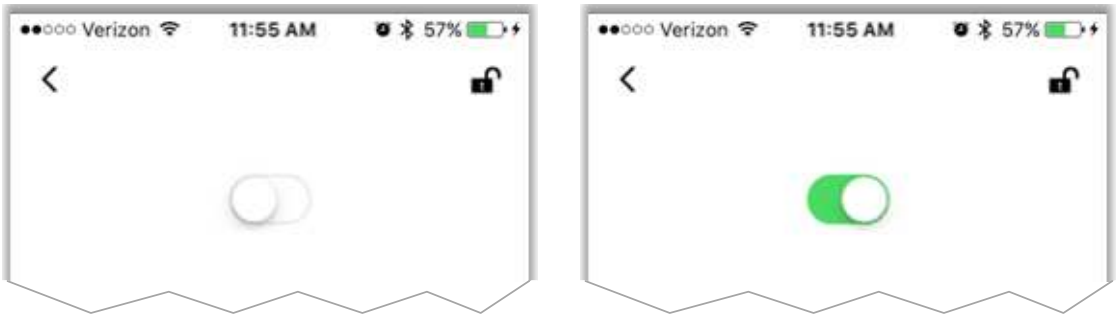


Figure 2-7. The way the switch graphic appears on an iPhone 6 screen.

Detecting and Using Events Occurring on the Mobile Platform's Screen

Observe that we can manipulate the switch on the mobile platform's screen in the previous example by dragging it to the left or right with our finger. By default, the "On" color associated with the switch is `GREEN` (we'll discuss

 Copyright © 2016 Simblee Corp.

how to change this later). However, manipulating the switch in the screen won't cause anything to happen in our sketch, because we have not yet provided any mechanism for it to do so. This is where the `ui_event()` function comes into play as illustrated in Sketch 2-8.

Sketch 2-8. Detecing and using switch events

```
#include <SimbleeForMobile.h>

uint8_t Switch; // 8-bit field in which to store switch ID number

void setup() {
  Serial.begin(9600);
  SimbleeForMobile.deviceName = "Switch";
  SimbleeForMobile.begin();
}

void loop() {
  SimbleeForMobile.process();
}

void ui() {
  SimbleeForMobile.beginScreen(WHITE,PORTRAIT);
  Switch = SimbleeForMobile.drawSwitch(135,100);
  SimbleeForMobile.endScreen();
}

void ui_event(event_t &event){
  if (event.id == Switch){
    Serial.println(event.value);
  }
}
```

Let's take this step-by-step. Each graphical object we instantiate is automatically assigned its own unique identifier (ID), whether we use it or not. The first thing we do is declare an unsigned 8-bit value called `Switch`. We use this value to store the switch's ID, which is returned by the `SimbleeForMobile.drawSwitch()` function.

Now consider the `ui_event()` function. This supports a single parameter of type `event_t`, which is defined in the Simblee Library. The parameter itself is the address of a structure, which contains the following information:

id	The identifier of the object that generated the event.
type	The type of event that occurred (`EVENT_PRESS`, `EVENT_RELEASE`, or `EVENT_DRAG`).
value	The current value of the object.
text	The current text associated with the object (non-numeric text fields only).
x	The X coordinate in the object where the press-release-drag action occurred.
y	The Y coordinate in the object where the press-release-drag action occurred.
red	The R value of the pixel touched (image only).
green	The G value of the pixel touched (image only).
blue	The B value of the pixel touched (image only).

 Copyright © 2016 Simblee Corp.

alpha The A value of the pixel touched (image only).

When the user performs some action on the mobile platform's screen, the system communicates this information back to a function in the Simblee Library. In turn, this calls the `ui_event()` function in our sketch.

In the case of our simple example, all we do is to check if the event is associated with our switch and, if so, we use a `Serial.println()` function call to display the current `value` associated with the switch on the Arduino IDE's Serial Monitor as illustrated in Figure 2-8.

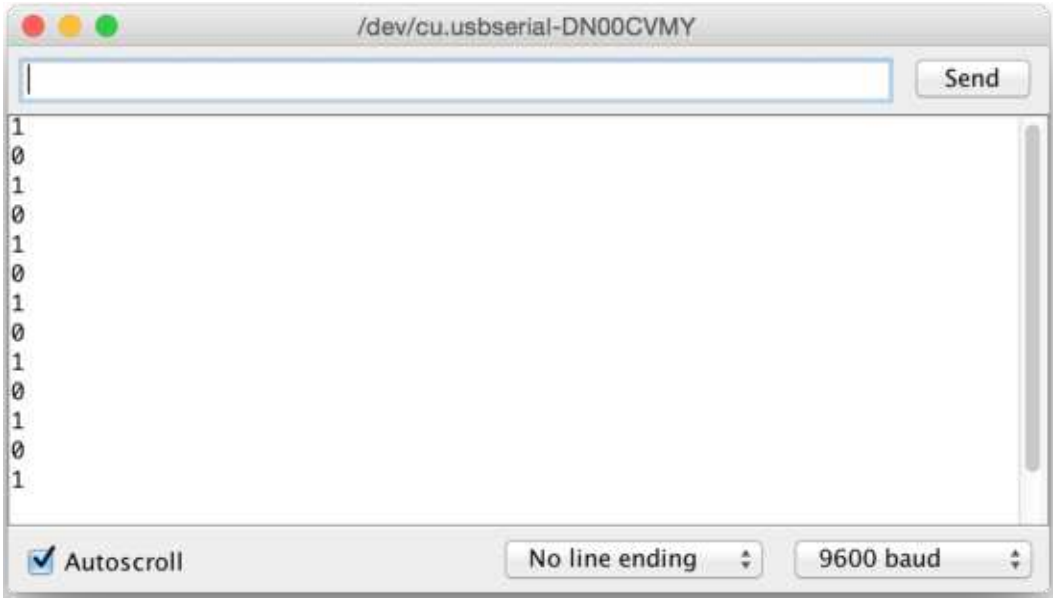


Figure 2-8. Displaying value changes on the switch in the Serial Monitor.

It's easy to see how we can use the `ui_event()` callback mechanism to interface with the rest of the sketch and with the outside world by doing things like (a) calling modifying the values associated with variables that can subsequently be accessed elsewhere in the sketch, (b) by calling other functions, and/or (c) by controlling the state of the Simblee's GPIOs.

NOTE: Have you noticed anything interesting about the `ui()` and `ui_event()` functions shown in the previous examples? The point is that we don't actually call these functions ourselves; instead, they are invoked by other functions that are hidden in the Simblee library.

 Copyright © 2016 Simblee Corp.

Appendix A: The Arduino IDE

The Arduino is an open-source computer hardware and software company, project, and user community that designs and manufactures microcontroller-based kits for building interactive devices that can sense and control objects in the physical world.

The Arduino IDE is an integrated development environment that is used to create the user programs (called "sketches") that run on Arduino microcontroller boards.

You can download the Arduino IDE from the main Arduino website (https://www.arduino.cc). This site provides comprehensive instructions for downloading and installing the Arduino IDE.

Other useful Arduino-related websites are as follows:

- http://www.adafruit.com/
- https://www.sparkfun.com/

Additional recommended books and resources are as follows:

- *Programming Arduino: Getting Started with Sketches* by Simon Monk (ISBN: 978-0071784221)
- *Programming Arduino Next Steps: Going Further with Sketches* by Simon Monk (ISBN: 978-0071830256)
- *Arduino Cookbook* by Michael Margolis (ISBN: 978-1449313876)
- *Arduino Workshop* by John Boxall (ISBN: 860-1200651553)
- *Understanding and Using C Pointers* by Richard Reese (ISBN: 978-1449344184)
- *Making Things Talk* by Tom Igo (ISBN: 978-1449392437)
- *Making Things Move* by Dustyn Roberts (ISBN: 978-0071741675)
- *Make an Arduino-Controller Robot* by Michael Margolis (ISBN: 978-1449344375)
- *The Maker's Guide to the Zombie Apocalypse* by Simon Monk (ISBN: 978-1593276676)

Appendix B: Installing the Simblee Library

Installing on a Windows Computer

Coming soon

Installing on a MAC Machine

Coming Soon

Installing on a Linux Machine

Coming Soon

Appendix C: RFDuino Shields

Coming soon

 Copyright © 2016 Simblee Corp.

Appendix D: Locating the Simblee's *variant.h* File

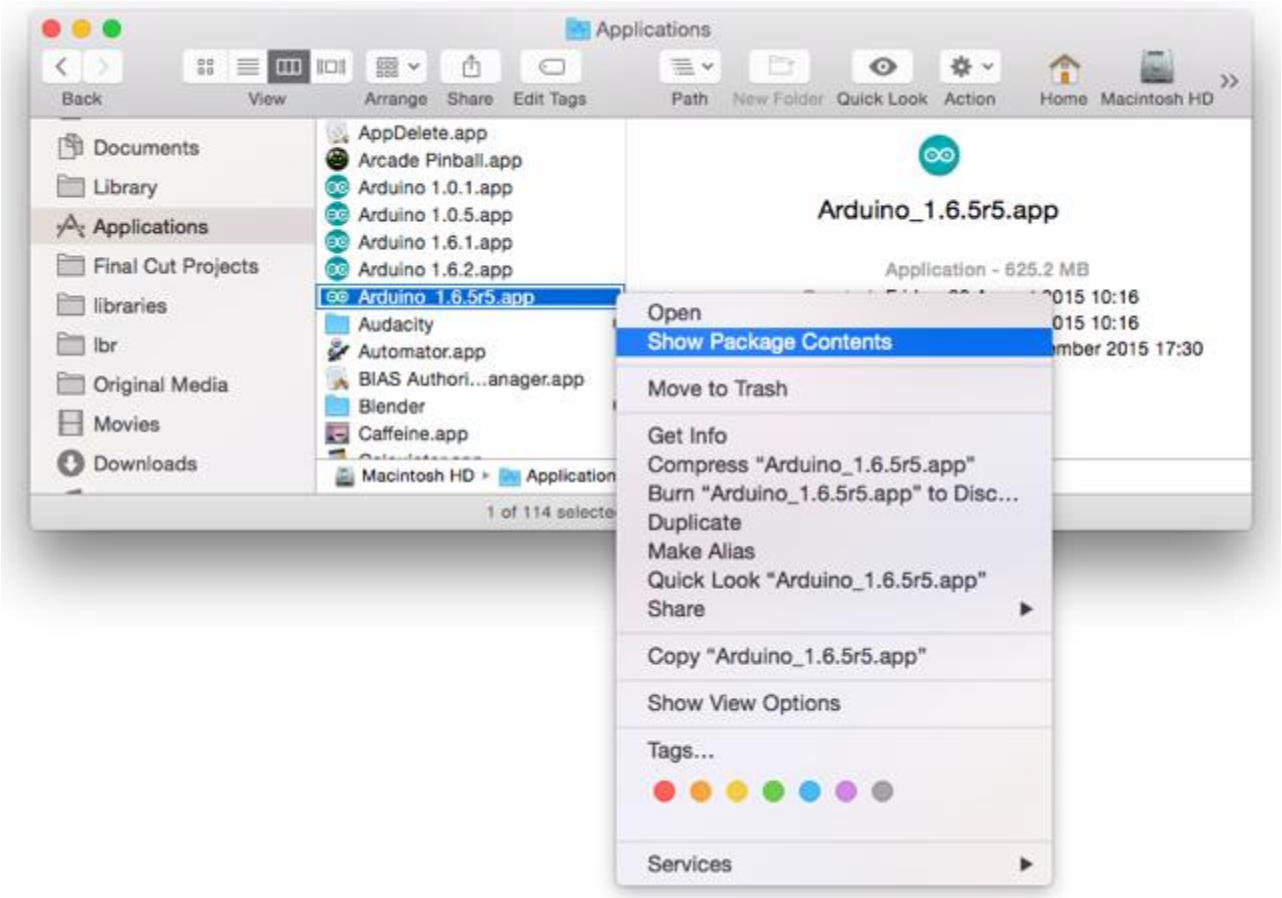
These instructions are intended to help you locate the Simblee's *variant.h* file on your host computer. One reason for accessing this file is to reassign the SPI port's GPIO pins from their default values.

On a PC

Coming soon

On a Mac

Locate the Arduino app in the *Applications* folder (*Macintosh HD/ApplicationsArduino_1.6.5r5.app*, in this example), then right-mouse-click on the app and select the **Show Package Contents** option as illustrated in Figure D-1.



Now navigate to the following path:

Contents/Java/portable/packages/Simblee/hardware/Simblee/1.0.0/variants/variant.h

 Copyright © 2016 Simblee Corp.