Simblee™
Connecting Everyone and Everything

1601 Pacific Coast Hwy • Suite 290 • Hermosa Beach • CA • 90254
www.simblee.com • Tel: 949.610.0008 • contact@simblee.com

Getting Started with SimbleeForMobile

SimbleeForMobile is a revolutionary new technology that allows you to create and interact with an iOS or Android(in development) interface all from Simblee. Using only one app, SimbleeForMobile enables interaction with any Simblee device, no matter the application.

Overview

You can easily dive into SimbleeForMobile by checking out the examples available. In the Arduino IDE, select

File > Examples > SimbleeForMobile

to see a series of examples on SimbleeForMobile in action. The best way to get started with SimbleeForMobile is by using the examples and through experimentation.

A simple SimbleeForMobile sketch contains 4 functions:

```
void setup() {

}

void loop() {

}

void ui() {

}

void event(event_t &event) {

}
```

Adding the SimbleeForMobile library into your sketch

Add the following line of code to the very beginning of your sketch:

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

void setup()

This is the standard sketch setup() function. You can setup different variables and settings inside of this function.

```
void setup() {

        // insert Setup code here

        SimbleeForMobile.begin(); // starts SimbleeForMobile

}
```

Other extensions will be covered later.

void loop()

This is the standard sketch loop() function.

```
void loop() {

        SimbleeForMobile.process(); // process must be called in the loop for SimbleeForMobile

}
```

void ui()

The ui() function is where the majority of the SimbleeForMobile user interface is constructed.

The ui() function is laid out as follows:

```
void ui() {

        SimbleeForMobile.beginScreen();

        // here is where you place your code to begin building your interface

        SimbleeForMobile.endScreen();

  }
```

void event(event_t &event)

The event callback is where SimbleeForMobile communicates user interface updates back to the sketch.

Before we go through the drawing functions, there are two important concepts we need to cover.

z-order

The order in which things are drawn on the screen can be important. If you draw a red rectangle first, and then draw a green rectangle that partially overlaps it, the green rectangle will appear to be on top of the red rectangle. If you reverse the order of the statements, the red rectangle will appear on top of the green rectangle. This gives the user interface a three dimensional aspect, which is where the term "z-order" comes from (the z axis being the third dimension, or distance away from the user).

opacity / transparency / alpha channel

The z-order is also important when transparency is used. Objects that are transparent, allow objects behind it to show through the transparent parts. Some image formats allow areas to be painted that are transparent (e.g.: png). Additionally, objects can be drawn with a transparent or semi-transparent color.

Color can be specified with the rgb() function, where the red, green and blue color components are represented as a value between 0 (black) and 255 (full intensity color).

Colors can also include an "alpha" channel using the rgba() function. The alpha channel is a number between 0 (which is opaque – meaning no transparency at all) to 255 (which is transparent – meaning no color at all). This allows the objects behind to show through.

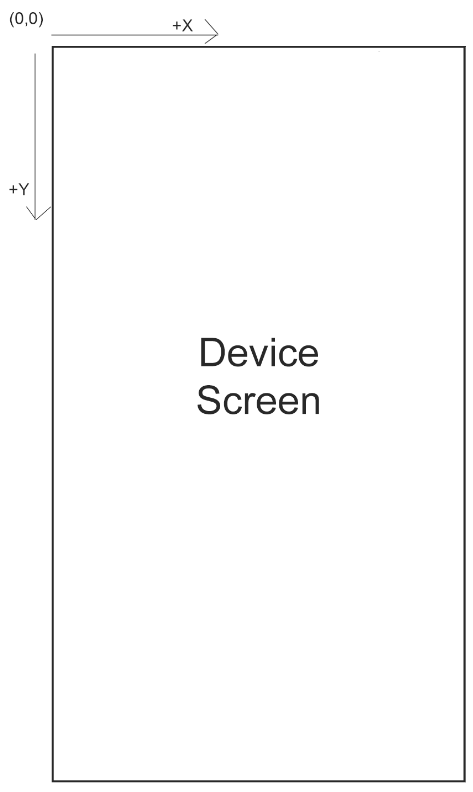
In addition, SimbleeForMobile defines the following color constants:

RED, GREEN, BLUE, YELLOW, MAGENTA, CYAN, WHITE, GRAY and CLEAR (transparent)

Draw Functions

Coordinate System

Simblee For Mobile uses an XY coordinate system that begins from the top left of your devices screen, X increasing to the right, Y increasing down.



For example, the iPhone 5 screen is 320 units wide, and 570 units tall.

You can use SimbleeForMobile.screenWidth and SimbleeForMobile.screenHeight to find the actual width and height of your display. These values are set once you are connected with a device using SimbleeForMobile. Simply print them out to your Serial Monitor, or display them on the screen.

Rectangles

Solid Color Rectangle Function

```
uint8_t drawRect(uint16_t x, uint16_t y, uint16_t w, uint16_t h, color_t color);
```

Gradient Color Rectangle Function

```
uint8_t drawRect(uint16_t x, uint16_t y, uint16_t w, uint16_t h, color_t color, color_t color2);
```

The first variant draws a rectangle with a solid color. The second variant draws a rectangle that is gradient filled from “color” at the top of the rectangle, to “color2” at the bottom of the rectangle.

Example:

To draw a rectangle with starting coordinates at x = 5 and y = 10. The rectangle will be 200 units wide and 400 units tall, and will be colored a gradient from orange to red, top to bottom:

```
SimbleeForMobile.drawRect(5, 10, 200, 400, ORANGE, RED);
```

Text

There are two groups of similar functions for drawing text, depending on whether you want to populate them with numbers or strings:

Text String Function

```
uint8_t drawText(uint16_t x, uint16_t y, const char *title);

uint8_t drawText(uint16_t x, uint16_t y, const char *title, color_t color);

uint8_t drawText(uint16_t x, uint16_t y, const char *title, color_t color, uint8_t size);
```

Text Number Function

```
uint8_t drawText(uint16_t x, uint16_t y, int16_t value);

uint8_t drawText(uint16_t x, uint16_t y, int16_t value, color_t color);

uint8_t drawText(uint16_t x, uint16_t y, int16_t value, color_t color, uint8_t size);
```

The height of the text is automatically calculated from the contents. The text can contain “\n”, allowing multiple lines of text in a single call.

Images

The functions to draw images are broken into two parts:

Image Source Function

```
uint8_t imageSource(int8_t image_num, uint8_t image_type, const uint8_t *image, uint16_t len);
```

The imageSource is the actual contents of the image file. You will need to convert your image into a header file with the extension ".h".

One utility that converts images to a header file is called "xxd" which is available on OSX and Linux by default. You can download a program online for Windows. These utilities generated code necessary to allocate a block of memory containing every byte in the original image file. The block of memory and the length are named after the original image file

For example, "dot_png" is the pointer to the memory block, and "dot_png_len" is the length

IMPORTANT NOTE: ***By default, the "xxd" utility declares the block as "unsigned char" which will be loaded into RAM. You can load the block into FLASH instead, so you do not run out of RAM, by adding CONST to the front of "unsigned char" of the header file.***

You can upload up to 31 images files to the iPhone, numbered 1 to 31.

So, assuming that we have run xxd on dot.png, and dot_png.h has been moved to our sketch directory, and #include "dot_png.h" at the top of our sketch, we can upload dot.png from the Simblee to the iPhone as imageSource number 1 as follows:

Example:

```
SimbleeForMobile.imageSource(1, dot_png, dot_png_len);
```

Now the image has been uploaded, image 1 can be drawn using the drawImage functions.

Draw Image Function

```
uint8_t drawImage(int8_t image_num, uint16_t x, uint16_t y);

uint8_t drawImage(int8_t num, uint16_t x, uint16_t y, uint16_t repeatx, uint16_t repeaty);
```

You can draw the image as many times as you like. In addition, the image may be repeated horizontally, or vertically, or both using a single call.

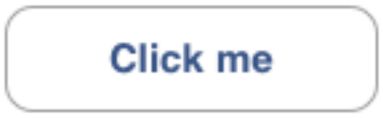
imageSources are cached on the iPhone to avoid uploading them every time (see cache management – covered later).

In addition, imageSources are uploaded in the order they appear in the ui() function. Since the drawImage for an image number can appear before the imageSource for it, this gives you control to when the imageSources are uploaded, separately from the z-order in which the images are presented.

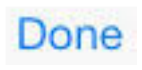
The iPhone application also contains some "Shipped Resources", which are images that are on the iPhone that don't need to be uploaded. These images are identified using negative numbers. Currently the only shipped resource is the ColorWheel image, which is assigned the image number -1 (or you can use the constant named COLOR_WHEEL).

An image does not support scaling by design – as shrinking an image means that more data was uploaded to the iPhone than necessary, and enlarging an image reduces the quality of the image. Images should be resized using tools like Photoshop prior to uploading.

Buttons



box type button



text_type button

Button Functions

```
uint8_t drawButton(uint16_t x, uint16_t y, uint16_t w, const char *title);

uint8_t drawButton(uint16_t x, uint16_t y, uint16_t w, const char *title, color_t color);

uint8_t drawButton(uint16_t x, uint16_t y, uint16_t w, const char *title, color_t color, uint8_t type);
```

Buttons comes in two styles:

The boxed button style (BOX_TYPE) that appeared in iOS 6

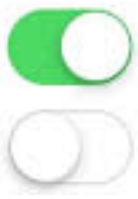
The text button style (TEXT_TYPE) that appeared in iOS 7 as the replacement for BOX_TYPE

The BOX_TYPE is the default button style, as the text style is often difficult to tell from colored text.

The button width can be specified, to aid in alignment. The button height is automatically calculated.

Buttons have a transparent background so that screen background color is always correct.

Switches

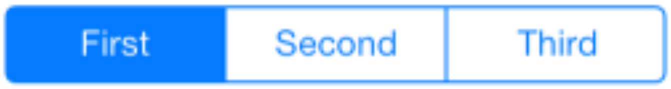


Switch Functions

```
uint8_t drawSwitch(uint16_t x, uint16_t y);

uint8_t drawSwitch(uint16_t x, uint16_t y, color_t color);
```

Switches only have two states – on or off.

Segment

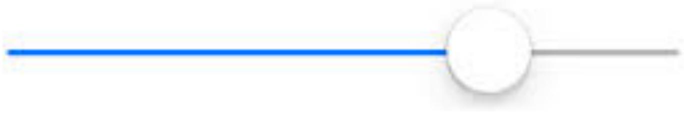


Segment Functions

```
uint8_t drawSegment(uint16_t x, uint16_t y, uint16_t w, const char * const *title, uint8_t count);

uint8_t drawSegment(uint16_t x, uint16_t y, uint16_t w, const char * const *title, uint8_t count, color_t color);
```

Segments allow one of a number of items to be selected.   A title can be specified for each one of the segments.

Sliders

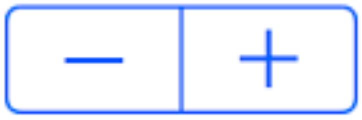


Slider Functions

```
uint8_t drawSlider(uint16_t x, uint16_t y, uint16_t w, int16_t min, int16_t max);

uint8_t drawSlider(uint16_t x, uint16_t y, uint16_t w, int16_t min, int16_t max, color_t color);
```

Sliders allow a range of values (from min to max) to be represented on a bar of a specified width.

Steppers



Stepper Functions

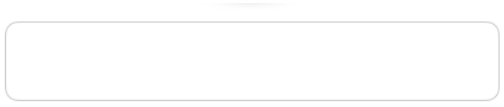
```
uint8_t drawStepper(uint16_t x, uint16_t y, uint16_t w, int16_t min, int16_t max);

uint8_t drawStepper(uint16_t x, uint16_t y, uint16_t w, int16_t min, int16_t max, color_t color);
```

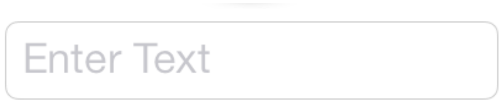
Steppers allow a value to be incremented or decremented from min to max using a "+" or "-" button.

The user can hold their finger on the "+" or "-" button to continuously increment or decrement the value.

Steppers are commonly used with TextFields to display the actual value, since steppers don't have a way to display the value directly.

TextFields


Regular TextField



TextField with placeholder

There are two groups of similar functions for TextFields, depending on whether you want to populate them with numbers or strings:

Text Field Functions (string)

```
uint8_t drawTextField(uint16_t x, uint16_t y, uint16_t w, const char *text);

uint8_t drawTextField(uint16_t x, uint16_t y, uint16_t w, const char *text, const char *placeholder);

uint8_t drawTextField(uint16_t x, uint16_t y, uint16_t w, const char *text, const char *placeholder, color_t
color);

uint8_t drawTextField(uint16_t x, uint16_t y, uint16_t w, const char *text, const char *placeholder, color_t
color, color_t background);
```

Text Field Functions (value)

```
uint8_t drawTextField(uint16_t x, uint16_t y, uint16_t w, int16_t value);

uint8_t drawTextField(uint16_t x, uint16_t y, uint16_t w, int16_t value, const char *placeholder);

uint8_t drawTextField(uint16_t x, uint16_t y, uint16_t w, int16_t value, const char *placeholder, color_t
color);

uint8_t drawTextField(uint16_t x, uint16_t y, uint16_t w, int16_t value, const char *placeholder, color_t color,
color_t background);
```

TextFields solicit user input via a touch keyboard (the value group uses a numeric keyboard; the text group uses the text keyboard).

A placeholder can be specified, which is a text string that appears in a blank text field informing the user of what type of text should be entered (so you don't need to add a label next to the field for the same purpose). Placeholders allow more area for text entry, and make the display less cluttered, but may not work depending on the user interface.

TextFields allow a background color to be specified, as its often difficult to see the contents if the TextField is presented on top of an image. The background color can be specified as CLEAR, if needed.

Update Functions

All the draw functions above create objects than can be updated after they are drawn.

Each draw function returns a uint8_t (an 8-bit unsigned number) identifier. That identifier can be used with the update functions below, to move the object, resize the object or change its color:

```
void updateX(uint8_t id, uint16_t x);

void updateY(uint8_t id, uint16_t y);

void updateW(uint8_t id, uint16_t w);

void updateH(uint8_t id, uint16_t h);

void updateOrigin(uint8_t id, uint16_t x, uint16_t y);

void updateSize(uint8_t id, uint16_t w, uint16_t h);

void updateRect(uint8_t id, uint16_t x, uint16_t y, uint16_t w, uint16_t h);

void updateColor(uint8_t id, color_t color);

void updateColor2(uint8_t id, color_t color2);
```

The identifier can also be used to update the objects contents:

```
void updateValue(uint8_t id, int16_t value);

void updateText(uint8_t id, const char *text);
```

Each object has the concept of a value:

For a button, its value is 1 if pressed, and 0 if not

For a switch, it is 0 for off, and 1 for on

For a segment, it is the 0 based index of the segment selected

For a slider, it's the value between min and max selected

For a stepper, it's the current value between min and max

A TextField can either contain a value or text (depending on whether the TextField is numeric or not, and whether updateValue or updateText is used.)

All other objects act like buttons (more on that later – see Events).

In addition, all objects can be hidden or shown using:

```
void setVisible(uint8_t id, bool visible);
```

Since you can't update an object that hasn't been created, the SimbleeForMobile.updatable flag can be used to determine whether its ok to update an object or not. The SimbleeForMobile.connected flag can be used to determine if the Simblee is even connected to the SimbleeForMobile iPhone app.

void ui_event(event_t &event)

A user interface is relatively limited if it can't interact with the user. The ui_event() function is the function called for all user interface events.

The event structure contains the following information:

event.id – the identifier of the object that generated the event

event.type – the type of event (EVENT_PRESS, EVENT_RELEASE, or EVENT_DRAG)

event.value – the current value of the object

event.text – the current text of the object (non numeric TextFields only)

event.x, event.y – the position inside the object the PRESS/RELEASE/DRAG occurred

event.red, blue, green, alpha – the color of the pixel touched (image only)

Events are generated when the objects value is changed, and only contains the updated value.   This reduces the amount of unnecessary information sent from the iPhone to the Simblee over BLE.

The setEvents function below is used to control which interactions generate an event, and what information is returned:

```
void setEvents(uint8_t id, uint8_t events);
```

… where the "events" parameter contains bit flags or'ed together from the following constants:

EVENT_PRESS, EVENT_DRAG, EVENT_RELEASE

and

EVENT_POSITION, EVENT_COLOR

Example:

So, given a button, if we want an event when the button is pressed, and when its released, we would call:

```
SimbleeForMobile.setEvents(button, PRESS_EVENT | RELEASE_EVENT);
```

If we want the color pressed in an image, we would call

```
SimbleeForMobile.setEvents(image, EVENT_COLOR);
```

Because images act like buttons by default, unless specified by setEvents, we don't need to specify PRESS_EVENT.

If we want the x and y coordinates when a user drags their finger across an image (or allow the image to be moved on the user interface), we would call:

```
SimbleeForMobile.setEvents(image, PRESS_EVENT | DRAG_EVENT | RELEASE_EVENT | POSITION_EVENT);
```

As mentioned above, all objects support events (press/release/drag – although the iPhone doesn't support all events on all objects). Objects like rectangles, text and images return values like buttons by default.

Back to setup()

SimbleeForMobile is built on top of SimbleeBLE. As such, the following are available:

SimbleeForMobile.deviceName

SimbleeForMobile.advertisementData

SimbleeForMobile.advertisementInterval

SimbleeForMobile.txPowerLevel

SimbleeForMobile.radioActive

SimbleeForMobile_onAdvertisement(bool start)

SimbleeForMobile_onConnect()

SimbleeForMobile_onRSSI(int rssi)

SimbleeForMobile_onDisconnect()

The SimbleeForMobile.radioActive flag can also be used to determine when the BLE radio is active (or about to become active).

NOTE: SimbleeBLE functions do not work with SimbleeForMobile functions.

Cache Control

SimbleeForMobile uses sophisticated cache control techniques to cache data on the iPhone, while ensuring that data delivered from the cache is not stale. f you simply ignore cache control, it will perform as expected.

The default behavior of SimbleeForMobile is to share a default common cache over all sketches. As the use of SimbleeForMobile grows world wide, you may want to manage the cache at a domain name level to prevent your sketch from being overwritten unnecessarily.

```
SimbleeForMobile.domainName
```

This variable is used to subdivide the cache by organizational domain name (think of it as a folder in the file system). An organization can share a cache between multiple sketches (this would be useful if they all shared the same images), or the domain name can be subdivided into multiple subdomains if a cache per application is required.

```
SimbleeForMobile.baseline
```

This variable is a date/time stamp in string format. The baseline variable is used to control the contents of the cache. Once a baseline date is established, any sketch using the same domain and baseline timestamp, has a known cache content state. If the contents of the cache must be invalidated (say because the company gets a new logo), then the image can be added and the baseline advanced to insure that all copies of the sketch utilize the new logo.

The default baseline for a sketch, is the date and time it was compiled. If you upload a large image, you may want to establish a baseline during development to prevent the image upload each time as follows:

```
SimbleeForMobile.baseline = “Oct 23 2014 13:02:04”;
```

If later, you update the image in Photoshop, simply advance the baseline as follows, to force a cache update:

```
SimbleeForMobile.baseline = “Oct 23 2014 14:28”;
```

(the seconds, minutes and even hours can be omitted if not needed)

SimbleeForMobile uses advanced compression techniques to reduce the data transferred over the BLE connection to construct the user interface. To further reduce the content transferred, each user interface for each sketch in a domain are also cached. During development, the user interface cache is invalidated each time you compile. Once development stops and deployment begins, the user interface is generated from the cache. Only the function calls between begin Screen() and endScreen() are cached, so you can control what contents are cached, and what contents are dynamically generated each time.

While the cache controls allow you to control the cache contents for an organizations domain(s), ultimately the user is in control of the maximum size of the cache. Once the users maximum cache size is exceeded, domains are discarded from the cache (oldest used first) until the users desired cache size threshold is met.

Multiple Screen User Interfaces

SimbleeForMobile user interfaces can support multiple screens. Each screen is given a screen number, with screen number 1 being shown first. Below are the relevant modifications necessary to create a 2 screen user interface, with a button press on screen 1 causing a transition to screen 2:

```
void screen1() {
    .. draw functions for screen 1
}
void screen2() {
    .. draw functions for screen 2
}
void ui() {
    if (SimbleeForMobile.screen == 1)
        screen1();
    else
      screen2();
}
void ui_event(event_t &event) {
    if (SimbleeForMobile.screen == 1)
        if (event.id == button)
            SimbleeForMobile.showScreen(2);
}
```