

MetaWatch Teardown and Arduino Hookup a learn.sparkfun.com tutorial

Available online at: <http://sfe.io/t113>

Contents

- [Introduction](#)
- [Use Tips](#)
- [Teardown](#)
- [Connecting Arduino \(Hardware\)](#)
- [Connecting Arduino \(Firmware\)](#)
- [Resources and Going Further](#)

Introduction

The MetaWatches and BlueSMiRFs used in this tutorial have been retired from our catalog. This tutorial is available for reference if you still need access to the concepts, methods, hardware connections, and example code.

We do have the [BlueSMiRF V2 with headers](#) and the [BlueSMiRF V2 PTH version](#) available if you are interested in using a BlueSMiRF V2 to stream UART data in your project!

The MetaWatch is a new open-source entry into the latest "Smart Watch" craze. It's a digital watch with a microprocessor and Bluetooth controller built in. It can interface with your smartphone to display stuff like weather forecasts, emails alerts, meeting notifications, or what music is playing (oh, and the time and date too).

SparkFun has a Developer's Kit of the MetaWatch in [black](#) and [white](#), which includes the watch, a programmer/charging clip, and a license to use TI's [Code Composer Studio](#).



The MetaWatch's standard use case is certainly very cool. Having notifications, weather, and music updates visible on your wrist is another step towards the future! But what else can we do with this hackable little device? In this tutorial, we'll go over some basics of the MetaWatch. Then we'll tear it down, and look at its guts! Then we'll put it back together and try to control it with...what else...an Arduino.



Required Materials

The Arduino portion of this tutorial will combine an Arduino board with a Bluetooth module.

- The **Arduino Board** can be any Arduino-compatible board -- [RedBoard](#), [Uno](#), [Pro Mini](#) etc.
- The [BlueSMiRF Silver](#) is used to serve as a bluetooth interface between Arduino and MetaWatch.

- You'll also need to **solder wires or headers** to the BlueSMiRF in order to get it connected to the Arduino.

Of course, you'll also need a MetaWatch in the style/color of your preference ([White FRAME](#) or [Black FRAME](#)).

Suggested Reading

If you're only interested in checking out the MetaWatch use tips, or teardown, please go right ahead. If you want to control it with an Arduino/BlueSMiRF combo, consider reading some of these tutorials before proceeding:

- [What is an Arduino?](#)
- [Serial Communication](#)
- [BlueSMiRF Hookup Guide](#)
- [How to Solder](#)

Use Tips

Before tearing your watch down, or connecting an Arduino to it, we'd recommend you check out what it can do when interfaced to your phone. There are **apps** available for both [Android](#) and [iOS](#) phones.

Play with it. Get a feel for how it functions. Notice how there are four "Idle mode" pages you can cycle through. Check it all out!

Playing with the watch might spawn a new project idea. Whether you want to write your own phone app, customize the watch's firmware, or connect a different Bluetooth device to it, we'd encourage everyone to discover a new, cool way to interact with the watch.

Charging, Connecting the Developer Clip

The Developer Clip is included to serve two purposes: charging and reprogramming. It takes a bit of aiming, but the clip is easy to attach to the watch.



The clip uses a [Spy-Bi-Wire](#) JTAG interface to communicate with the watch's MSP430 microcontroller. If you get really into the embedded firmware development side of the MetaWatch, this'll be what you use to upload and debug a program. For more more info on using the Developer Clip as a JTAG interface, check out the [JTAG Reflashing documentation](#).

Button Labels and Gestures

The watch has six buttons, labeled A-F. A is the top-right button (at two o'clock), and they increment clockwise around the watch.



The standard button uses are:

- A: Press to close out of most views.
- B: Cycle Idle mode pages
- C: Call up settings view. Turn on/off bluetooth radio, backlight, seconds display, invert display, etc.
- D: Show info display. Shows charge level, firmware version, bluetooth address, and other stats.
- E: Music display (phone-dependant)
- F: Backlight

On top of that, you can **reset** the watch by holding down the middle buttons (B and E) down for a few seconds. Alternatively, holding F does the same thing.

Teardown

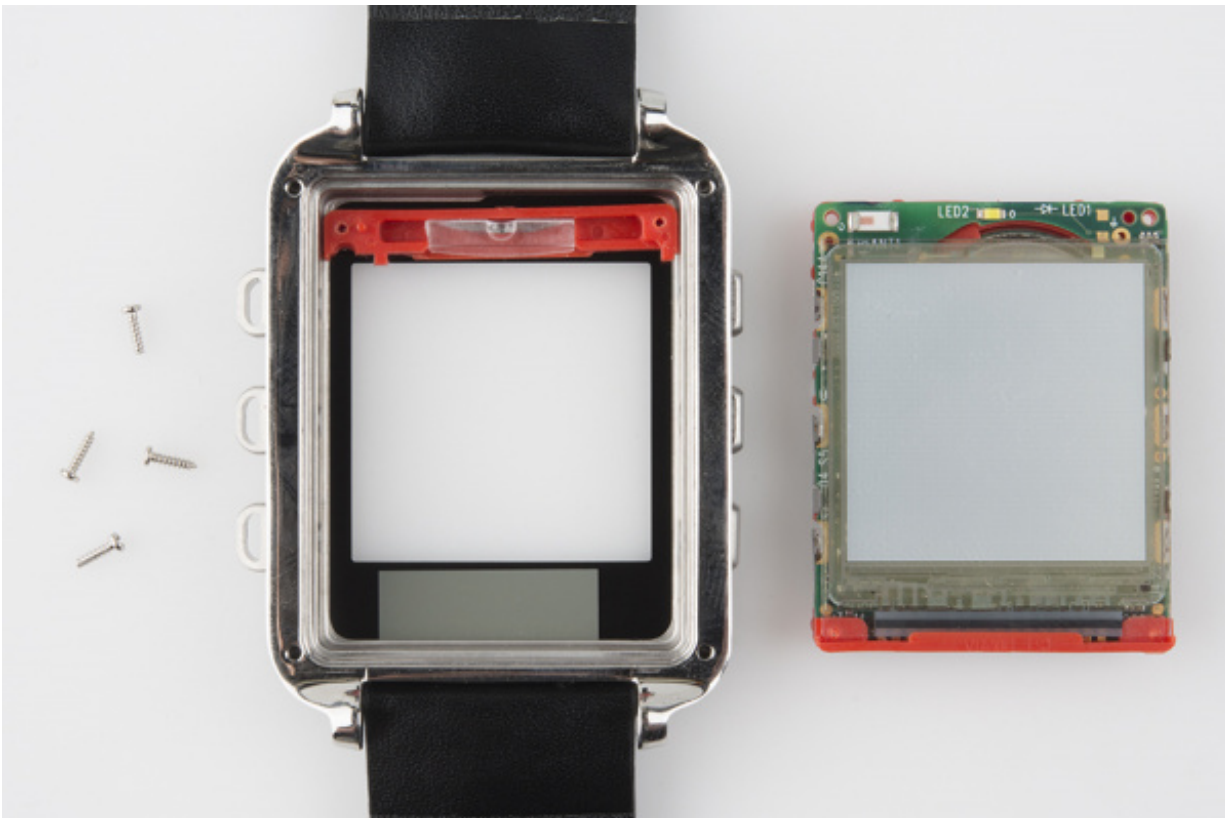
The designers of the MetaWatch made it super-easy to open it up. All you'll need is a small 1.4mm flat-head screwdriver, and an equally small #0-or-so Phillips-head. These screwdrivers should be present in most [screwdriver sets](#). The first set of screws -- flatheads -- are on the four corners of the back of the watch.



Loosening those screws, and removing the back reveals the watch's power source: a 20mm 3.7V **Li-Ion coin cell battery** (a smaller version of [this](#)). The other big, visible, circular component is a **vibration motor**. You can also see four spring connectors for the JTAG interface.

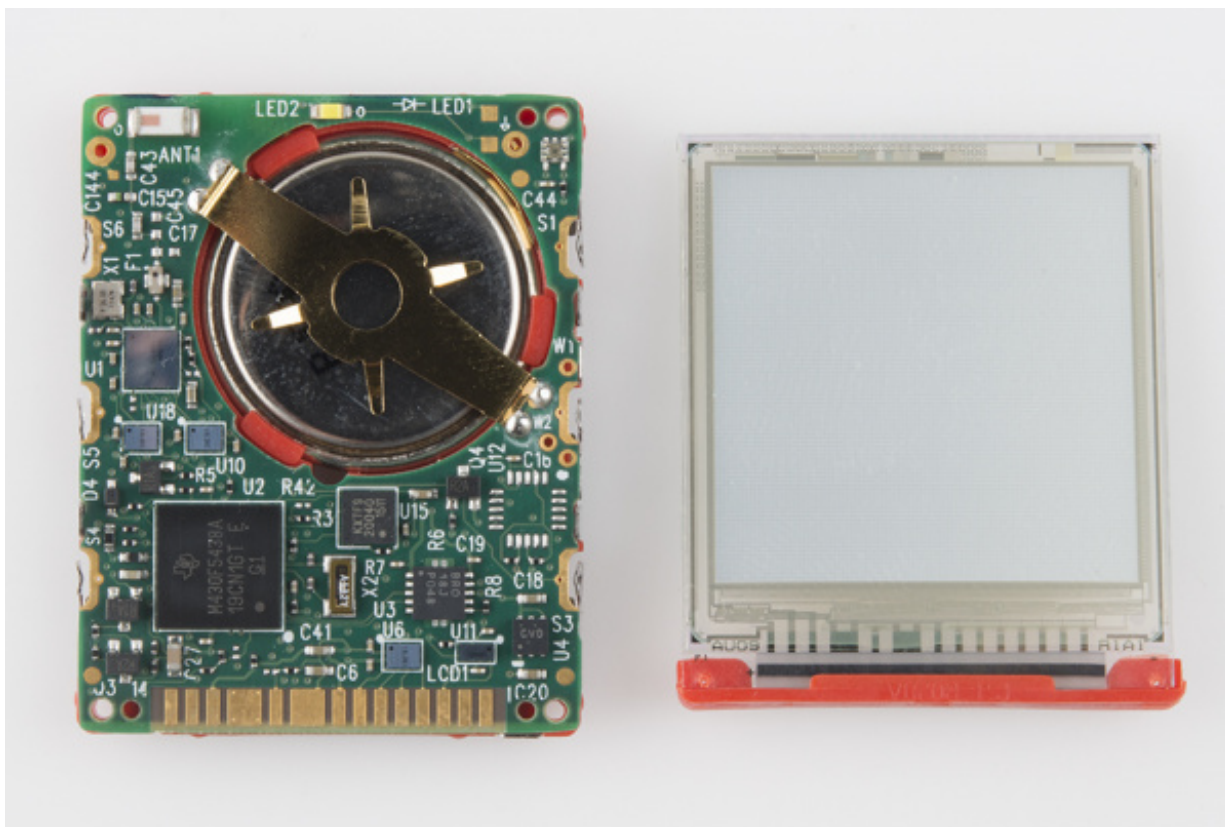


The guts of the watch are tied to the watch frame by four small Phillips screws. Loosening those screws allows you to (gently) remove the entire LCD/PCB assembly from the frame (you may have to pry it up with the flathead):



Examining the Guts

The MetaWatch's LCD is a 96x96 pixel, reflective, always-on display. We didn't get a picture of the back, but the part number is printed there. It's a [Sharp LS013B4DN01](#). Looks like it's got a simple [SPI](#)-like interface. Neat!

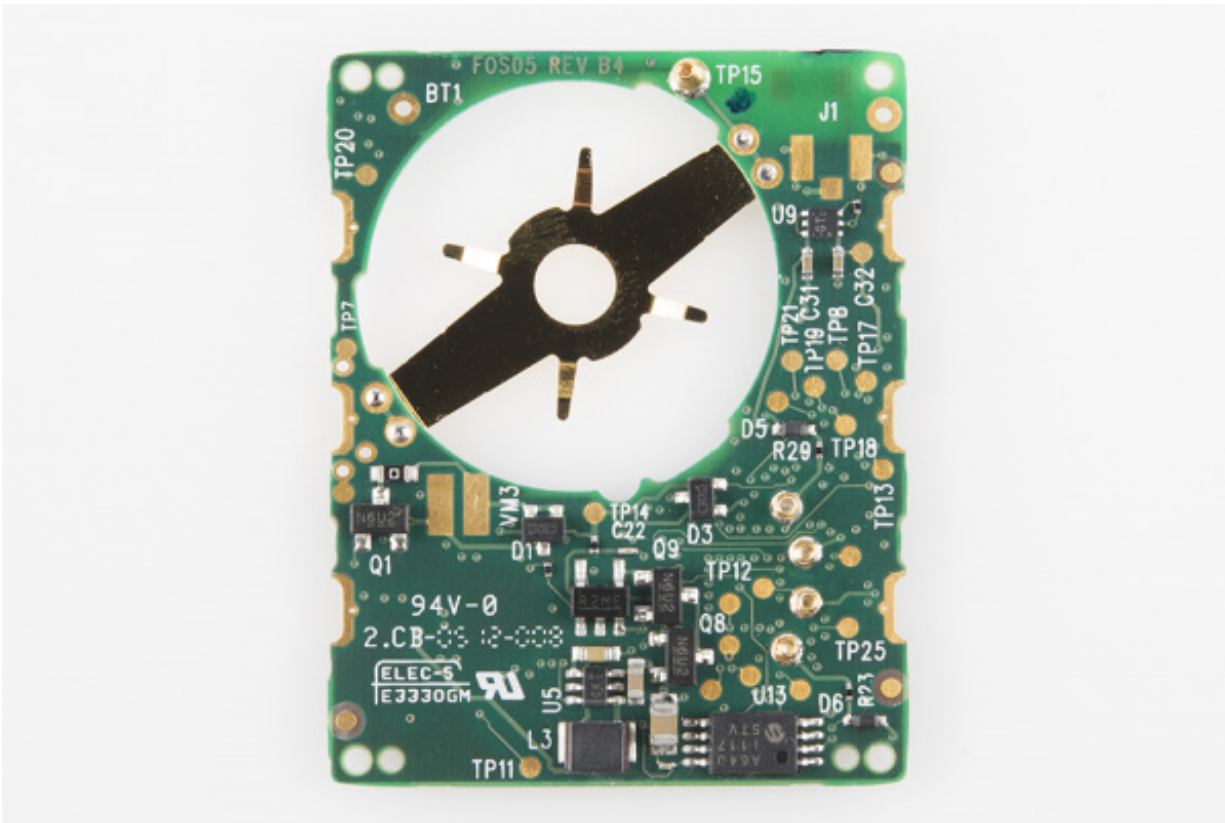


You can gently lift the display off the main PCB. The display and PCB are connected together using a [zebra connector](#), so they just have to be lined up and pressed together to interface. The PCB is very thin -- 0.6mm.



The main processor is the BGA-packaged version of the [MSP430F5438A](#). Ultra-low power, 256KB flash, 16KB RAM, 87 I/O. Sweet chip! Low power too. The small chip labeled KXTF9 to the right of the MSP430 is a [Kionix accelerometer](#). The blue-ish, reflective chip above the MSP430 is a [CC2560 Bluetooth controller](#). Those are the most heavy-lifting ICs on the watch. There's also a chip antenna (top-left), backlight LED (top-middle), and light sensor (top-right). We also notice an unpopulated IC package -- intriguing.

Let's flip it over:



Looks like most of the interesting stuff was on the top side. On the back, there's what looks like a Microchip [Serial EEPROM](#), maybe some voltage regulator circuitry, and a wealth of test points. Meh.

And that's about it. The watch is really easy to take apart, and put back together (and it still works!). It seems very hackable, and it shows off a few chips which may be useful for other projects.



Next we'll have a look at controlling the watch via Bluetooth, using its published API.

Connecting Arduino (Hardware)

I wanted to take the watch's published bluetooth API for a test drive. But instead of writing a phone app for it, I wanted to use something a little more familiar. Arduino! I'm not sure how useful it'll be in the long run (part of the watch's allure is it's connected to a phone, which is connected to the Internet), but it's a fun exercise in controlling a cool, consumer-product-looking device with an Arduino.

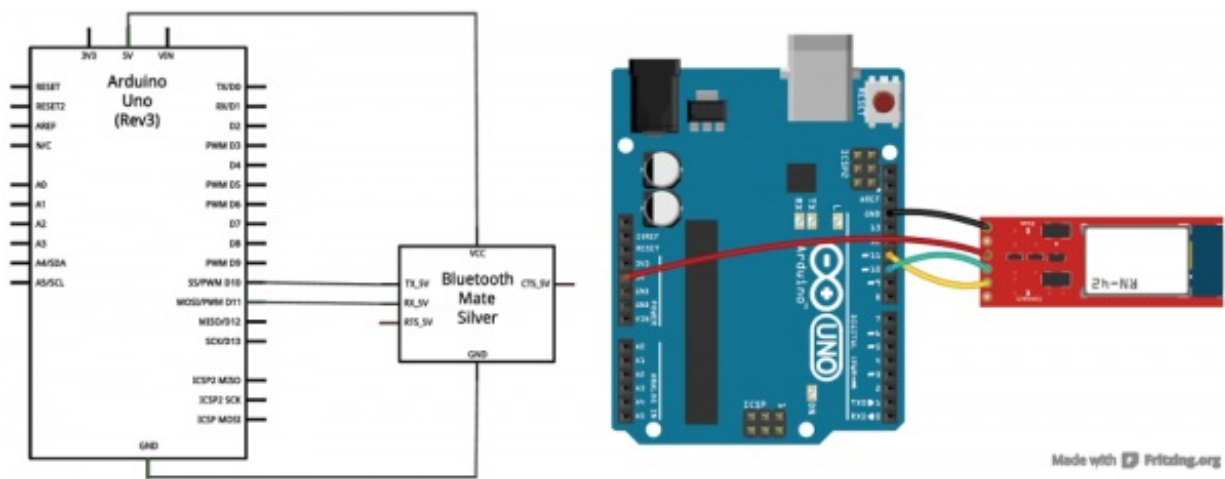
Connecting Bluetooth

You'll need a Bluetooth module to interface between the Arduino and MetaWatch. The MetaWatch's CC2560 Bluetooth module is a dual-mode, so it supports serial port profile (SPP) as well as Bluetooth 4.0 (BLE). We'll stick to using our go-to SPP Bluetooth module, the [BlueSMiRF Silver](#) (the [Mate](#) should also work).



The BlueSMiRF interfaces to an Arduino via [serial interface](#). On the next page, in the Firmware section, we'll use the [Software Serial Arduino library](#) to set up our serial interface (keeping the hardware interface free for Serial Monitor debugging).

Connecting the BlueSMiRF to the Arduino is about as simple as it gets: four wires. Two wires for power -- 5V and GND -- and two wires are for the two serial lines.



Note: If pins 10 and 11 don't work for your project, you can swap those pins to any SoftwareSerial-enabled pin. You'll need to change a couple values in the library if you do so.

Connecting Arduino (Firmware)

Download the SFE_MetaWatch Library

We wrote a simple library to interface from Arduino to BlueSMiRF to MetaWatch. Click [here](#) to download the library (or visit the [GitHub repo](#) to help contribute!). If you need help installing it, check out our [How to Install an Arduino Library tutorial](#).

The library includes a couple pieces of example code, we'll be discussing the **SFE_MetaWatch_Menu.ino** example in this tutorial. You can go to *File > Examples > SFE_MetaWatch > SFE_MetaWatch_Menu* within Arduino to open it.

A few things on the library:

- It assumes you have a BlueSMiRF Silver (based on the [RN-42 bluetooth module](#)) connected as shown in the Hardware portion of this guide (Arduino pins 10 and 11).
- It defines a class called `SFE_MetaWatch`, which has a range of member functions you can call to interact with the watch.
- It does its best to automatically connect to the watch, but sometimes this'll need to be done manually from the Serial Monitor (see the directions below).

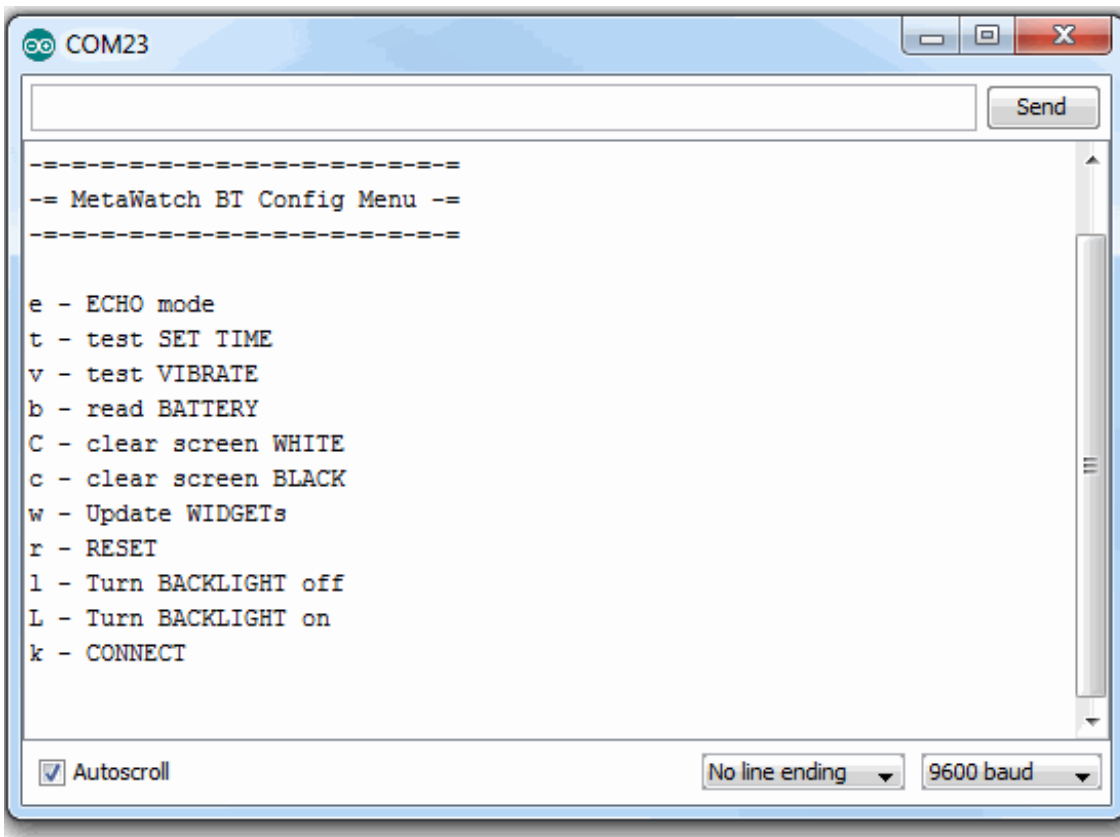
Connecting BlueSMiRF to MetaWatch

The toughest part of all of this is getting the BlueSMiRF connected to the MetaWatch. Before uploading, make sure you set the `btBaudRate` variable near the top of the sketch to the baud rate of your BlueSMiRF (115200 is the module's default). Then set the `metaWatchAddress` variable to the 12-digit HEX address of your watch (press button D (bottom-left), the address is the xxxx-xxxx-xxxx formatted number at the bottom).

After uploading the code, **open up the serial monitor** (set at 9600 bps). When the code first starts, you can enter *k* (case-sensitive) to attempt to connect between BlueSMiRF and watch. If it succeeds, the BlueSMiRF's green, "Connect" LED should turn on.

If the connect fails, you'll enter into echo mode. Here, anything you send to the Serial Monitor will be relayed to the BlueSMiRF. If you're in echo mode, follow these steps to connect:

1. Make sure **No line ending** is selected in the Serial Monitor.
2. Enter command mode by sending `$$$`. The BlueSMiRF's stat LED should blink very fast to show that it's in command mode. If it doesn't double-check that the `btBaudRate` variable is correctly set.
3. Switch the line-ending drop down in the Serial Monitor to **Newline**.
4. Type *C*, and click "Send". should be the 12-digit HEX string (0-9, A-F) matching the watch's BT address.
5. The BlueSMiRF should respond "TRYING", and in a few seconds the BlueSMiRF's green connected LED should illuminate.



If it still doesn't connect after you see "TRYING", double-check the `metaWatchAddress` variable. Also make sure both devices are in range of each other. If all else fails, restoring factory default values to the BlueSMiRF may be a last resort.

Using the Library

Once connected, you can play around with the menu choices to adjust stuff on the watch. You should definitely try out setting up the clock widgets (send `w`), setting the time (send `t`, then `HHMMSS`), vibrating (`v`), and controlling the backlight (`l` for off, `L` for on). For more information on what's going on, check out the comments in the code.

There are a few parts of the code to highlight:

Include the Library and Create a Watch Variable

These two pieces of code are required. Include the library near the top of your sketch. Then, sometime before `setup()` create an `SFE_MetaWatch` instance (we called it `watch`), which you'll refer to through the rest of the sketch. The two parameters for this constructor are the Watch's BT address, and the baud rate of your BlueSMiRF module.

```
language:c
```

```
#include <SFE_MetaWatch.h>
```

```
...
```

```
SFE_MetaWatch watch(metaWatchAddress, btBaudRate);
```

Begin and Connect

The `begin()` function should be called before you do anything else with the watch.

The `connect()` function will attempt to connect the BlueSMiRF to the MetaWatch. If successful, it should return a 1. If it fails it will either return a negative number.

```
language:c
watch.begin();
...
if (watch.connect() < 0) // If connect fails, enter echo mode to manually connect
{
  Serial.println("Connect failure. Putting you into echo mode.");
  Serial.println("Type $$$ (no line ending) to get into command mode, should respond CMD");
  Serial.println("Type C,<BT address> (newline) to connect to watch. No dashes in address.");
  Serial.println("Type ~~~ to exit echo mode");
  watch.echoMode(); // echo mode, will loop forever until it receives ~~~
}
```

If you're having trouble connecting, the `echoMode()` function is a handy tool to try to communicate directly between the Serial Monitor and BlueSMiRF.

Useful Watch Control Functions

The library can be used to control the watch's vibration motor, backlight, and the display. You can also read from the watch's light sensor, accelerometer, and battery gauge. And of course, since it is a watch, you can set the real-time clock's (RTC) hours, minutes, seconds, and date variables.

Here are a few handy functions to interface with the watch.

Set the RTC

The `setTime(year, month, date, day, hours, minutes, seconds)` function should be used to set the watch's clock. Each parameter is required. `year` and `month` are pretty self-explanatory, as are `hours`, `minutes` and `seconds`. `Date` is the date of the month (e.g. August**15**). `day` is the day of the week, and it should be a value between 0 and 6 (Sunday is 0, Saturday is 6).

Draw on the Display

The watch has four different idle pages you can play with, and cycle through by pressing the B button (middle-right). Eventually, we'll get to writing a sprite drawing function on the display, but for now here are some functions you can play with:

The display can be **cleared** completely black or white using the `clear(x)` function. If `x` is 0 it'll make the screen white, and if it's 1 the screen will black-out.

Pre-defined **widgets** can be drawn on the screen using the `setWidget(msgTotal, msgIndex, widgetIDSet[],`

numWidg) function. Most of these widgets are clock-related. You can draw full-screen clocks, quarter-screen and half-screen. The widgets aren't super-well documented in the API, so it takes some guess-and-check to find out which widget ID does what. The example code throws all four of the full-screen clock widgets on the screen. Check out the code and comments to see how it does that.

Before sending the `setWidget()` function, make sure to send the `watch.fullScreen(0)` command. That tells the watch that the Arduino will be drawing on the full screen of the watch.

After either the `clear()` or `setWidget()` commands, you need to send the `watch.update()` message, which tells the watch to draw what we've been putting in its buffer.

```
language:c
watch.fullScreen(1); // Two options here are 0 for 2/3rd screen or 1 for full-screen
watch.setWidget(1, 0, fullClockWidget, 4);
watch.update(0, 0, 96, 0, 1, 0); // Update display to make it happen.
Serial.println("Widgets updated");
```

Those four lines, for example, will get your watch to draw four clock full-page clock widgets on four different pages.

Vibrate

You can control the watch's vibration motor using the `vibrate(timeOn, timeOff, numCycles)` function. `timeOn` determines how many milliseconds the watch vibrates, `timeOff` determines how many milliseconds to *not* vibrate, and `numCycles` tells the watch how many times to repeat that process.

The example, `vibrate(100, 100, 5)` will vibrate 100ms, then stop for 100ms, and repeat five times.

Reset

If your watch goes crazy, or gets into an unknown state, the `reset()` function will start you over. After sending `reset()` you'll need to re-connect to the watch.

If you'd like to contribute to the SFE_MetaWatch library, or add some example code of your own, check out the [GitHub repository](#).

Resources and Going Further

Because MetaWatch is open-source, there's a wealth of information out there:

- [Developer Site](#)
- [Source Code on GitHub](#)
- [Remote API](#) or [PDF](#)
- [Developer Forums](#)
- [JTAG Reflashing Documentation](#) (PDF)

If you've got some MSP430-development chops, try customizing the watch's firmware. We've found

one such project [here](#), which looks pretty neat.

Going Further

- [Leap Motion Teardown](#) -- If you're still on a teardowns kick, check out what we found inside the Leap Motion.
- [Using the OpenSegment](#) -- If the Watch has you inspired to make one of your own, the OpenSegment might be all you need to control a 4-digit 7-segment display.
- [Serial 7-Segment Hookup Guide](#) -- The S7S is a smaller version of the OpenSegment.
- [Connecting Arduino to Processing](#) -- Instead of using the clunky Serial Monitor interface, you could create a nice GUI to interface with your Arduino from Processing!

learn.sparkfun.com | [CC BY-SA 3.0](https://creativecommons.org/licenses/by-sa/3.0/) | SparkFun Electronics | Niwot, Colorado