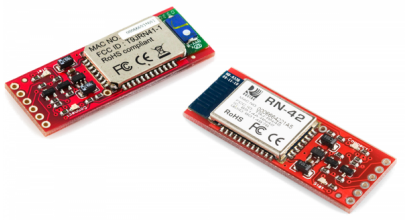
Using the BlueSMiRF

CONTRIBUTORS: *JIMB0*

♡ FAVORITE 5

Introduction

Are you ready to hit the airwaves and add Bluetooth to your project? With the BlueSMiRF and Bluetooth Mate line of products, you're much closer than you think to replacing those pesky, tangled RX and TX wires with 2.4GHz wireless communication.



Each of these modules has a Bluetooth transceiver on it, meaning they're capable of both sending and receiving data. They're perfect for directly replacing a wired asynchronous serial interface. Free of wires, your devices can be up to 100 meters away from each other. On top of those benefits, these modules are also very easy to use. There's no messing with Bluetooth protocols or the stack, just send data over a serial interface, and it's piped through to whatever Bluetooth module to which it's connected.

In this tutorial we'll cover everything you need to know about these Bluetooth modules. We'll begin with an overview of the hardware, and the differences between each device. Then we'll get into hardware hookup and example Arduino code.

Materials and Tools

For starters, you'll need one of the four Bluetooth modems we'll be covering in this tutorial: the Bluetooth Mate Silver, BlueSMiRF Silver, Bluetooth Mate Gold, or BlueSMiRF Gold. The modules all function in the same way, so this tutorial is applicable to all four.

Wireless communication won't do you any good unless you have two devices that can talk to each other! These Bluetooth modems can talk to **any other Bluetooth device that supports SPP**. That (long) list includes other BlueSMiRFs or Bluetooth Mates, or Bluetooth modules embedded into your computer, or even your smartphone. If your computer doesn't already have a Bluetooth module in it, you can plug a Bluetooth USB Module into an available USB slot.

We'll also need something to talk to the Bluetooth modem on the serial end. This will usually be a microcontroller of some sort. In this tutorial we'll be using an Arduino.

Finally, in order to interface to the Bluetooth modem, you'll need to **solder headers or wires** to it. So you'll need a simple soldering iron and solder. This topic is covered further in the Hardware Hookup section.

Suggested Reading

First and foremost, check out the Bluetooth technology tutorial if you want to learn some of the general concepts behind this nifty wireless standard. It'd also be good if you're familiar with these concepts:

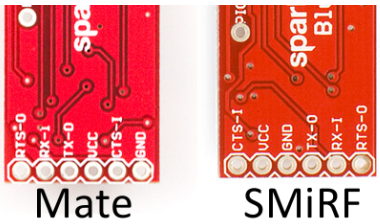
- Serial Communication
- How to Solder
- What is Arduino?
- Serial Terminal Basics

Hardware Overview

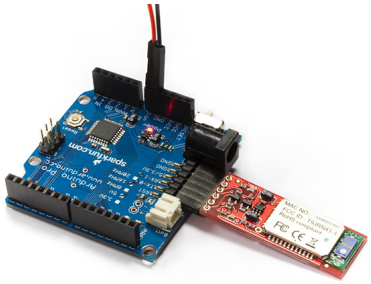
SMiRF? Mate? Silver? Gold? What's the Difference?

The "Silver" and "Gold" designations of these modules indicates whether they use an RN-42 Bluetooth module or an RN-41. The Silvers use the RN-42, and the Gold uses an RN-41. The difference between those two modules? **Range and transmit power.** The RN-41 is a class 1 Bluetooth module, so it can communicate at up to 100 meters, but it also transmits at a higher power (meaning shorter battery life). The RN-42 is class 2, which limits the transmit range to about 10 meters.

The difference between Mate and SMiRF all comes down to the pin-out of the six-pin header. If you flip each of the boards over, and look at the pin labels, this is what you'll see:



The pinout of the Mate matches that of products like the FTDI Basic and the FTDI Cable. It's a "standardized" pinout for a serial interface and power supply combination. This pinout allows the Mate to be plugged directly into the serial header of Arduino Pro's and Pro Minis.

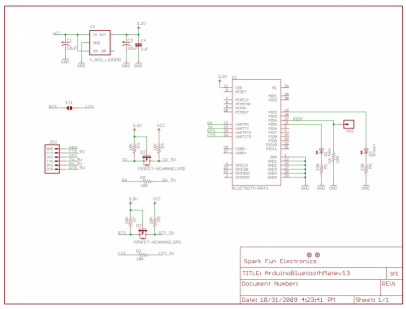


A Bluetooth Mate can be plugged directly into the serial header of an Arduino Pro.

That's all there is to this whole Mate/SMiRF/Silver/Gold debacle: transmit range and pinout. Besides that, everything else on these boards is the *exact same* – schematic, command interface, size, you name it.

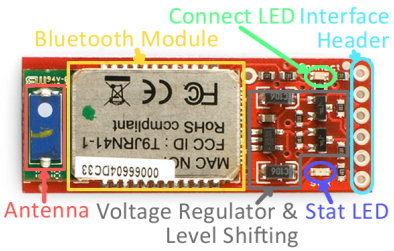
Design Overview

The RN-42 and RN-41 are pin-for-pin compatible, so the schematic for each of these boards is the same. The only difference exists at the connector pin-out for the Mate and SMiRF. Click the image below to see a bigger view of the schematic (or click here to see it in PDF form).



Key to the design are **level shifting circuits** between the RN-41/42's serial pins, and the output header. The maximum operating voltage of the Roving Networks modules is 3.3V, so these enable a device operating at 5V (like an Arduino) to safely communicate with the Bluetooth modems. There is also a linear 3.3V regulator on the board, so a voltage from 3.3V to 6V can be used to supply power to the module.

The boards also include two LEDs. There's a red "Stat" LED, and a green "Connect" LED. These can be used to determine what state the Bluetooth module is in.



Finally, be aware of where the antenna is – give it some room to breathe. Don't place it near any big chunks of metal or enclose it in a Faraday cage, and you should be just fine.

The Pinouts

Each of the four Bluetooth boards breaks out six pins. Four pins are devoted to the serial interface, and the other two are for power.

Pin Label	Pin Function	Input, Output, Power?	Description
RTS-O	Request to send	Output	RTS is used for hardware flow control in some serial interfaces. This output is not critical for simple serial communication.
RX-I	Serial receive	Input	This pin receives serial data from another device. It **should be connected to the TX** of the other device.
TX-O	Serial transmit	Output	This pin sends serial data to another device. It **should be connected to the RX** of the other device.
VCC	Voltage supply	Power In	This voltage supply signal is routed through a 3.3V regulator, then routed to the Bluetooth module. It should range from **3.3V to 6V.**
CTS-I	Clear to send	Input	CTS is another serial flow control signal. Like RTS, it's not required for most, simple serial interfaces.
GND	Ground	Power In	The 0V reference voltage, common to any other device connected to the Bluetooth modem.

Powering the Modules

These Bluetooth devices are designed to work seamlessly in both 3.3V and 5V systems. The **voltage** supplied to the VCC/GND pins can be anywhere **between 3.3V and 6V**. Voltages on the input serial and control signals (RX-I and CTS-I) can be anywhere between 3.3V and 5V. The output signals (TX-O and RTS-O) will range from 0V for a LOW logic level, and VCC for a HIGH. That means if you power them at 6V, the TX and RTS signals will output up to 6V.

The **current consumption** of a modem depends on what it's doing at the time. It can be as low as 0.026mA when the device is asleep, and as high as 50mA when data is being transmitted. This table from the datasheet provides some good estimates:

Electrical Characteristics

Parameter	Min	Typ.	Max.	Unit
Supply Voltage (DC)	3.0	3.3	3.6	V
Average power consumption				
Radio ON* (Discovery or Inquiry window time)		40		mA
Connected Idle (No Sniff)		25		mA
Connected Idle (Sniff 100 milli secs)		12		mA
Connected with data transfer	40	45	50	mA
Deep Sleep Idle mode		26		uA

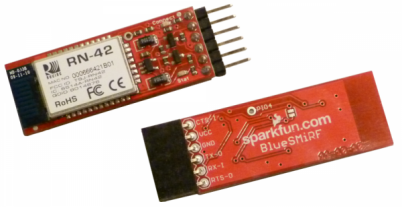
Connecting a device up to the Bluetooth modems is as easy as applying power and wiring up the serial RX and TX pins. What do we send over that serial interface, though? That's where we need to look at the the firmware and the Bluetooth module's operation modes.

Hardware Hookup

Assembly

Happily, most of the assembly on these modules is done for you; you don't need to learn how to solder SMD components just yet. However, before you can begin using these Bluetooth modules, you'll need to solder *something* into the six plated-through-holes to form a solid electrical connection.

What you solder into the holes depends mostly on what you're going to connect the device to. If you've got a Bluetooth Mate, and want to connect it directly to an Arduino Pro, you may want to throw a right-angle female header on there. Another good option, which makes the board breadboard-compatible, is male-headers. A third, ever-reliable option is to solder wires directly to the holes.

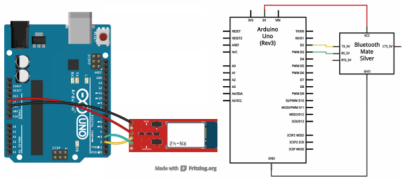


Right-angle male or female headers are good options for assembly. They make the modules breadboard or jumper-wire compatible.

Connecting Everything Together

We need to connect the Bluetooth modems to devices that can send and receive serial signals. These are TTL-level serial signals, make sure you don't confuse that with RS-232! Voltages should be between 3.3V and 5V. There are loads of options here, for this tutorial we'll use an Arduino.

Instead of connecting the Bluetooth modem to the Arduino's lone hardware UART, we'll use SoftwareSerial and connect the modem's RX and TX pins to any of the Arduino's free digital pins. This will help to avoid bus contention and will make sure the Bluetooth modem doesn't receive any spurious data during a sketch upload. Here's the connections we'll make for the example code later in this tutorial:



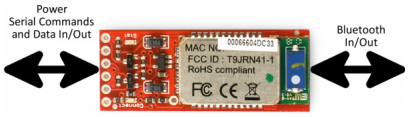
Note that this is a Bluetooth Mate shown in the Fritzing diagram, the BlueSMiRF will have a different pinout.

TX-O is connected to D2 of the Arduino, RX-I is connected to D3, GND goes to GND, and VCC goes to 5V. The CTS-I and RTS-O pins are left floating. The TX-O and RX-I pins could really be connected to any digital pin (besides 0 and 1), so if you need 2 and 3 for something else, feel free to move those around.

Half of the hardware hookup is done. We still need to create a wireless connection to another Bluetooth device. Before we can delve further into that, though, we need to understand more about the Bluetooth modem's firmware.

Firmware Overview

A serial interface is all it takes to control these Bluetooth modules and send data through them. They act, essentially, like a data pipeline. Serial data that goes into the module (from the RX-I pin), is passed out the Bluetooth connection. Data coming in from the Bluetooth side is passed out the serial side (out the TX-O pin).



Establishing this data pipeline is a two step process. First, we need to connect something capable of sending and receiving serial data to the header of the Bluetooth modem. We achieved this in the Hardware Hookup phase by connecting an Arduino to the serial header, but any microcontroller with a UART could work. With the device connected we need to configure the serial port to work at the same baud rate the the modem is configured to – they default to **115200 bps** (8-N-1).

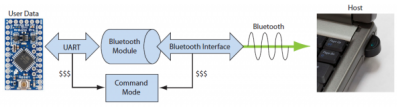
Secondly, on the Bluetooth end of things, we need to establish a wireless connection between the modem and another Bluetooth device. The only stipulation here is the other Bluetooth device must support SPP (which most do). This connection involves a pairing process similar to connecting any other Bluetooth devices together. More on that later. Let's talk a bit more about the serial interface.

Data and Command Modes

Controlling the Bluetooth module and sending data through it are two very separate operations, but they're both done via the serial interface. To differentiate between these two forms of data, the Bluetooth modules implement two different communication modes.

Command mode is used to configure the Bluetooth module. Characteristics like the device name, baud rate, PIN code, and data rate can be adjusted in command mode. This is also where action commands are sent to the module, which can tell it to connect to a device or scan for other modules.

In **data mode**, the Bluetooth module acts as a transparent data gateway. Any data received over the Bluetooth connection is routed out the module's TX pin. And data sent to the module's RX pin is piped out over the Bluetooth connection.



To enter command mode from data mode, the host controller needs to send a string of three *$* symbols (`$$$`).

Configuration Timer

The configuration timer is the one obstacle to watch out for when entering command mode. The config timer begins counting as soon as the Bluetooth modem is turned on, and once it's done counting, you'll be unable to enter config mode unless you cycle power. By default the config timer is set to 60 seconds, however this can be adjusted, or even turned off (that's the ticket!).

Deciphering the LEDs

There are two LEDs on the Bluetooth modems: a red one labeled "Stat", and a green one labeled "Connect". These help to indicate the status of the module. Never forget the importance of blinkies! The green LED will illuminate when a wireless connection is formed. The "Stat" LED can indicate that the module is in one of three states, depending on how fast it blinks:

Mode	Stat Blink Rate	Notes
Configuration	10 per second	Module is *in* config mode.
Startup/Config Timer	2 per second	Module is not in config mode, but the configuration timer is still counting.
Discoverable/Inquiring/Idle	1 per second	Not in config mode, and the config timer has run out.

If you're having trouble getting the module to enter configuration mode, make sure the timer hasn't run out by checking for a very slow blink rate.

Commanding the Bluetooth Modems

Control of the Bluetooth modems is achieved through a series of AT commands, all of which are documented in the Advanced User's Guide. If you want to get the most out of these modules, make sure you read through that. The commands are split into five categories: set, get, change, action, and GPIO commands. Chapter 2 of the User's Guide covers each of the commands in detail. Appendix B is a quick reference guide – an excellent resource.

In the Example Code section we'll go over a few of the more commonly used commands – naming the device, searching for available modules, and connecting to them.

Example Code: Using Command Mode

With a little ingenuity, we can use the Arduino as a medium between us and the Bluetooth Mate/BlueSMiRF to send and receive commands. Here's a small sketch which relays data between the Arduino Serial Monitor and a Bluetooth modem.

```
/*
  Example Bluetooth Serial Passthrough Sketch
 by: Jim Lindblom
 SparkFun Electronics
 date: February 26, 2013
 license: Public domain

 This example sketch converts an RN-42 bluetooth module to
 communicate at 9600 bps (from 115200), and passes any serial
 data between Serial Monitor and bluetooth module.
 */
#include <SoftwareSerial.h>

int bluetoothTx = 2;  // TX-O pin of bluetooth mate, Arduino D2
int bluetoothRx = 3;  // RX-I pin of bluetooth mate, Arduino D3

SoftwareSerial bluetooth(bluetoothTx, bluetoothRx);

void setup()
{
  Serial.begin(9600);  // Begin the serial monitor at 9600bps

  bluetooth.begin(115200);  // The Bluetooth Mate defaults to 115200bps
  bluetooth.print("$");  // Print three times individually
  bluetooth.print("$");
  bluetooth.print("$");  // Enter command mode
  delay(100);  // Short delay, wait for the Mate to send back CMD
  bluetooth.println("U,9600,N");  // Temporarily Change the baudrate to 9600, no parity
  // 115200 can be too fast at times for NewSoftSerial to relay the data reliably
  bluetooth.begin(9600);  // Start bluetooth serial at 9600
}

void loop()
{
  if(bluetooth.available())  // If the bluetooth sent any characters
  {
    // Send any characters the bluetooth prints to the serial monitor
    Serial.print((char)bluetooth.read());
  }
  if(Serial.available())  // If stuff was typed in the serial monitor
  {
    // Send any characters the Serial monitor prints to the bluetooth
    bluetooth.print((char)Serial.read());
  }
  // and loop forever and ever!
}
```

This sketch makes use of the SoftwareSerial library, which should be included with most of the recent versions of Arduino.

At the beginning of the sketch, the Arduino enters the command mode string and temporarily changes the Bluetooth modem's baud rate to 9600 bps (using the `U,9600,N` command). Remember this is temporary, so when power is cycled, the modem will default back to 115200 bps.

The loop of the sketch simply checks to see if either the Bluetooth modem or the Serial Monitor have sent any data to the Arduino. If so, it'll relay the data sent from one device to the other.

Using the Passthrough Sketch

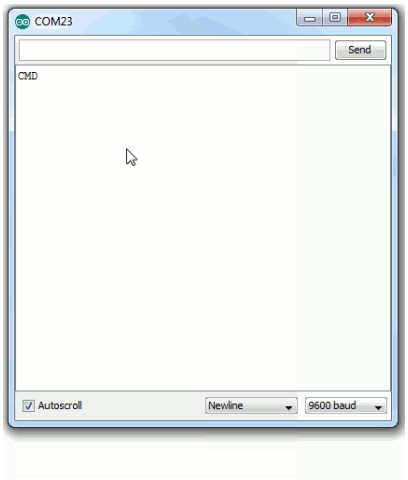
With the code uploaded, and everything hooked up accordingly, open up the Serial Monitor. Make sure the baud rate is set to 9600. Throughout this process you'll have to fudge around with the dropdown menu to the left of the baud rate selection. It should initially be set to **"No line ending"**.

First, let's enter **command mode** by typing `$$$` , and click "Send". You should see the Bluetooth modem respond with `CMD` , and you'll notice the red Stat LED blinking much faster, this all indicates that the device is in command mode.

Once you're in command mode, you'll need to change the line ending dropdown to **"Newline"**. The basis of all this is that the RN-42 module expects a newline character after every command except for the command mode string. Annoying, but we'll deal.

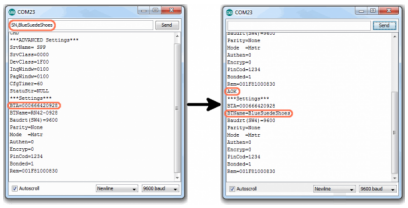
Using GET Commands

The **GET commands** are a good place to start using command mode, they'll display settings, status, or other information that might be helpful. Try sending the **"Display Basic Settings" command by typing "D"**, and pressing "Send". This will trigger a response from the Bluetooth modem that details, among other things, the baud rate settings, the name, and the address (BTA) of the device. The address is something you should take note of, it can either be identified from this command, or by taking a gander at the module's label, next to the "MAC NO". Each Bluetooth moule has a unique address which can't be changed. Try sending the other get commands, and see what information you can retrieve from the modem.



Using SET Commands

After sending the “D” command, you may have noticed your Bluetooth modem has it’s own name, in addition to the address. Unlike the address, this name can be changed to whatever you’d like. By default it’ll be RN42-XXXX, where XXXX is the last four digits of the address. Let’s give a SET command a whirl. The `SN,<name>` command is used to set the name, where <name> is any collection of up to 20 characters. Think up a unique name, and assign it to your device. After sending the SN command, the modem should respond with an “AOK”. Now if you send the D command, you should see your new name listed next to the “BTName” setting.

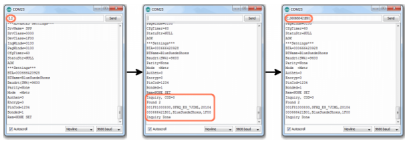


Be careful with the SET commands, only change something if you’re sure it won’t negatively affect the modem, or your ability to communicate with it. If you change something you don’t think you should have, send the `SF, 1` command to reset *everything* back to its factory default value. Another handy command, if you’re lazy like me, is `ST,0`, which turns the config timer off. Remember that any setting you modify will be saved to the Bluetooth modem’s memory, and will be retained upon loss of power.

ACTION Commands

Finally, it’s time for some action. Among other things the Bluetooth modem’s ACTION commands can be used to find other Bluetooth devices, connect to them, and disconnect from them.

Begin by sending the **inquiry scan command** – `I,<value>` – to search for other Bluetooth modules in range. The <value> parameter defines the number of seconds the modem will take to look for other modules. It’ll default to 10 if not defined. If you just type “I” and click send, the device should respond with a “Inquiry, COD=0”, and then after ten seconds it’ll respond with any Bluetooth modules it found. It will print their information as “BT address, BT name, COD” (COD is class of device).



If the modem finds any modules, you can try sending the **connect command** – `C,<address>` – to connect to one of them. The modem in the example above found two devices in range, by sending the `C,000666421B01` command, we can attempt to connect to one of them.

After sending the connect command, the device will respond with “TRYING”, which will be followed by either “CONNECT failed” (the meaning of which should be pretty apparent) or the connection will be successful! After a successful connection we immediately enter data mode, and the modem becomes a pipeline. Any characters sent from one Bluetooth device will be sent to the other, and vice-versa. To disconnect, you’ll need to re-enter command mode (don’t forget to set to “No new line”), and send the “K,” command (*with* Newline selected, bleh).

There are a lot of other commands to explore! Thumb through the User’s Manual and familiarize yourself with all of the power at your Bluetooth modems’s fingertips!

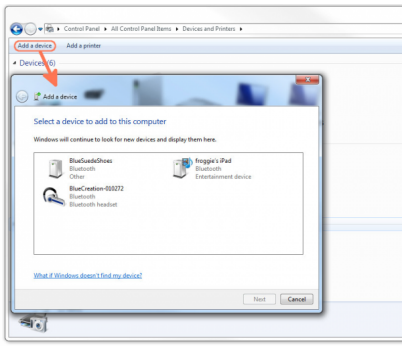
Connecting From Another Device

In the example code section we attempted to connect to a device from the Bluetooth modem, but what if you wanted to initiate the connection from another Bluetooth device? This process varies by operating system and device, but most of the steps involved are pretty similar.

If your device (computer, phone, etc.) doesn’t already have an Bluetooth modem, you’ll need to connect an external module to it. The Bluetooth USB Module works for any computer with an available USB slot.

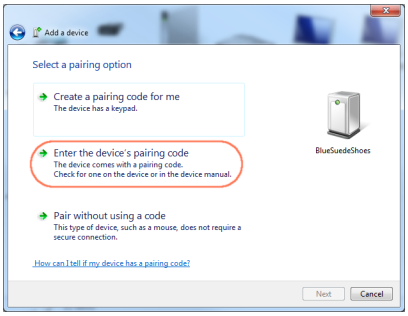
Connecting to the Modem in Windows

Go to the **Control Panel** and navigate to the **Devices and Printers** window. In the top-left section of that window, there should be an **Add a device** button. Click that.

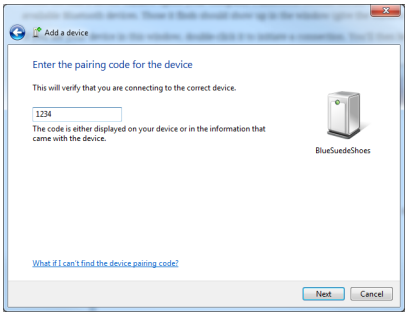


When the *Add a device* window opens your computer's Bluetooth module should automatically search for any in-range, available Bluetooth devices. Those it finds should show up in the window (give the window a few seconds to search).

If you see your device in this window, double-click it to initiate a connection. You'll then be presented with the **Select a pairing option** window. Since the modems don't have an attached keypad, select the **Enter the device's pairing code** option.

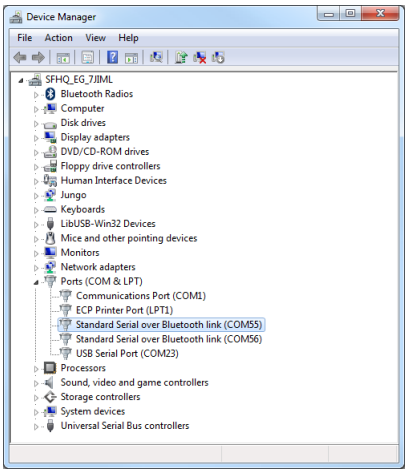


On the next window, enter **1234** as the PIN code. This is the default PIN value for every RN-42 and RN-41.



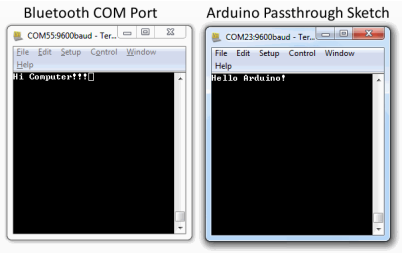
Windows will take a few moments to install drivers for your device. Once it's done, it'll pop up a notification to let you know that your device is ready to use!

But how do you actually *use* it? You'll need to open up a terminal emulator (check out our Serial Terminal Basics tutorial for help!). When Windows installed drivers for your new Bluetooth device, it created a new COM port for it. Opening up your **device manager**, and looking in the "Ports (COM & LPT)" tree, you'll find a new port named "Standard Serial over Bluetooth link (COM##)" (there may be two of them).



To open up a connection between the Bluetooth devices, open up a terminal to that COM port at 9600 bps (8-N-1). (If you see two ports, try the lower number first). When the terminal opens up, your Bluetooth modem's green connect LED should light up. Connection successful!

If you have the sketch from the last example (the serial passthrough) still loaded up on your Arduino, you can open up a second terminal window to communicate between devices.



If you're within the config timer window (cycle power on the modem if you're not), you can even *remotely* enter command mode by sending the "$$$" string. Now you can remotely alter the settings of your Bluetooth modem. Nifty!

If your using a Mac, Linux, or even a smartphone, pairing and connecting should involve a similar process. If authentication is required, you'll want to use the PIN-code option, and enter the default PIN of "1234". Open up a serial terminal emulator – *Terminal* or *CoolTerm* on Mac OSX, a variety of apps are available for smartphones – to initiate the connection and start passing data.

Resources and Going Further

Hopefully this tutorial has prepared you for an exciting foray into the world of wireless communication. Now that you have a good idea of how to command these Bluetooth modems, and connect them to other devices, the rest is up to you. How are you going to make use of your pleasant lack of wire? Go hit the airwaves!

Resources

- RN-42 and RN-41 Command Reference and User's Guide
- RN-42 Datasheet – For "Silver" versions.
- RN-41 Datasheet – For "Gold" versions.
- Bluetooth Mate Schematic
- BlueSMiRF Schematic

Going Further

If you're interested in checking out other Bluetooth-related tutorials, check these links out:

- RN-52 Hookup Guide – The RN-52 is a Bluetooth v3.0 module, which (on top of SPP) supports the Bluetooth audio profile A2DP. Using this module you could make a wireless boom box, or a Bluetooth-enabled MP3 player!
- MetaWatch Teardown and Hookup – The MetaWatch is a smartwatch with an embedded Bluetooth module. In this tutorial we use a Bluetooth Mate to communicate between the watch and an Arduino.

Here are some other tutorials which features wireless communication:

- IR Communication - If you're only really looking to transmit wireless data short distances, infrared may be a good (cheap!) option.
- ATmega128RFA1 Dev Board Hookup Guide – The ATmega128RFA1 sports an RF module which operates on the same frequency as Bluetooth (2.4 GHz). If you want to dig down into the nitty, gritty area of RF communication, check out this board.
- Electric Imp Hookup Guide – The Electric Imp makes connecting to WiFi incredibly easy. Follow along with this tutorial, and you'll have an embedded module able to interact with web pages!