Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

Electric Imp Breakout Hookup Guide

impRoduction

The Electric Imp is a deviously awesome development platform. Disguised as an every day SD card, the imp is actually a unique combination of microprocessor and WiFi module. The imp makes connecting any device to the Internet a breeze. Looking to catch on with this “Internet of Things” fad? The imp is an excellent place to start.

The Electric Imp card and imp002 Breakout Board

In this tutorial, we’ll be explaining how to use the imp card with one of our Breakout Boards as well as the imp002 breakout board. You will have the choice of which platform to use (the imp card or the imp002).

First, we’ll cover how to hook up the hardware end of the imp and imp002. Following that we’ll head over into the firmware domain, programming the imp to blink LEDs and read analog and digital inputs. The last code example shows off the coolest part of the imp: controlling hardware over the Internet!

Required Materials

You have a choice to make! You can either use the imp card and Breakout Board, or you can use the imp002 Breakout Board.

If you want to use the imp card, you will need an imp card and the Electric Imp Breakout Board.

Electric Imp
WRL-11395
★★★★☆ 3

SparkFun Electric Imp Breakout
BOB-12886

If, on the other hand, you want to use the imp002, you will need the Electric Imp imp002 Breakout Board.

SparkFun Electric Imp imp002 Breakout
BOB-12958
★★★★★ 2

Aside from one of those platforms, we'll use a few common electronics parts you may already have. Here's a wishlist of everything else we'll be using.

NOTE: The 2-pin jumper is only required for the Electric Imp Breakout Board.

Electric Imp Hookup Guide SparkFun Wish List

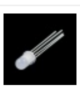

Jumper - 2 Pin
PRT-09044


SparkFun USB Mini-B Cable - 6 Foot
CAB-11301


Breadboard - Translucent Self-Adhesive (Red)
PRT-11317


Rotary Potentiometer - Linear (10k ohm)
COM-09288


Resistor Kit - 1/4W (500 total)
COM-10969


LED - RGB Diffused Common Cathode
COM-09264

LED - Basic Red 5mm
COM-09590


Mini Pushbutton Switch
COM-00097


Jumper Wires Standard 7" M/M Pack of 30
PRT-11026

	Break Away Headers - Straight PRT-00116
	Mini Photocell SEN-09088

In addition to those items, you'll also need the following non-SparkFun materials:

- Wireless network with Internet access
- Electric Imp planner account (sign up is free/easy)
- Electric Imp planner website pulled up in your web browser
- SmartPhone w/ the Electric Imp app (Android or iOS)

Tools

There will be some soldering involved. The Breakout Board does not come with header pins soldered on, which you'll need in order to interface with the imp's I/O pins. You'll need a simple soldering iron and a bit of solder (If you've never soldered before, this is a great place to start! The solder points are easy, through-hole jobs).

Before We Begin

This tutorial builds upon some basic electronics concepts. If you aren't familiar with any of the topics below, consider reading through that tutorial first:

- How to Solder - Through-hole
- How to Power a Project
- Voltage Dividers
- Pulse Width Modulation
- Light-emitting Diodes

Aside from the imp's programming language, Squirrel, there will be a variety of coding languages used in later parts of this tutorial – primarily HTML and Javascript. Don't worry if you're not too familiar with those, as the examples aim to be short, sweet, and easy-to-modify.

Let's start by overviewing the imp hardware itself. It's hard, at first, to wrap your head around the fact that this little, module is actually a powerful WiFi-enabled microcontroller platform.

About the imp Card

It may look like an everyday SD card, but the imp is much, much more. It's a WiFi-enabled microprocessor. It's programmable over the air. It's got GPIOs, UARTS, I^2C and SPI interfaces, pulse-width-modulation, digital-to-analog and analog-to-digital converters. Basically, it's what you'd get if you smushed an ARM microprocessor and a WiFi module down into a tiny SD-card-sized package.

The imp provides an easy, integrated way to connect almost any hardware device to Internet services. It is well suited to be the backbone of your Internet-enabled project, whether you're remotely controlling your electric blanket or triggering an irrigation system via a web browser. Connecting your imp to a wireless network and programming it is a simple, streamlined process.

The Hardware: 6 Wondrous I/Os

The imp is basically made of pure awesome. But, if we lift the hood of awesomeness for a moment, we can talk a bit about the imp's hardware. The platform of the imp is a Cortex-M3 microprocessor. Just like any microprocessor, the imp has a collection of input and output pins, each with unique functions. There are six addressable I/O pins – not as many as an Arduino, but it makes up for it in terms of functionality. The imp has three UARTs, two I^{2}C and SPI interfaces, and two DAC outputs; plus each pin can act as an ADC input and PWM output.

imp pin table from Imp's Pin Mux's Page

Pin #	**UART$_{1289}$**	**UART$_{57}$**	**UART$_{12}$**	**I^{2}C$_{89}$**	**I^{2}C$_{12}$**	**SPI$_{257}$**	**SPI$_{189}$**	**DAC**	**ADC**	**PWM**
1	CTS		TX		SCL		SCLK	Yes	Yes	Yes
2	RTS		RX		SDA	MISO			Yes	Yes
5		TX				SCLK		Yes	Yes	Yes
7		RX				MOSI			Yes	Yes
8	TX			SCL			MOSI		Yes	Yes
9	RX			SDA			MISO		Yes	Yes

Of course, each of those pins can also be used as a simple inputs (with or without pull-up resistors) or outputs, sinking/sourcing up to 4mA each.

Also in that tiny SD package is a WiFi module, an antenna, and a light sensor. We'll find out why the light sensor is critical in the coming pages.

The imp is a **1.8-3.3V** device, supplying it any more voltage than that can be harmful. It can require up to 400mA (worst-case), but it'll usually pull about **80mA** (even 5mA in a power-save mode).

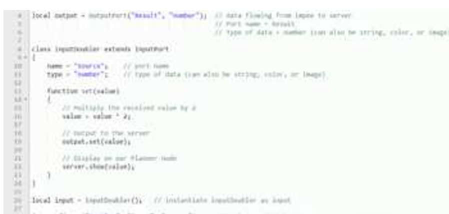
The IDE

All code written for the imp is done online, in a browser-based integrated development environment (IDE). Everyone can (freely) create their own account on the IDE, where both your programs and your imps are kept safe and secure. There are certainly pros and cons to this "always online" approach (though you can write and save every program locally, and upload it when you're ready). Still, it seems like a good solution for this type of platform.

Code in the IDE is divided into two halves: the imp device, and the agent. Code in the **device** half is code that actually runs on your imp. The **agent** is a process living on Electric Imp's cloud server. It can communicate with both your imp, and the outside Internet world. We'll dig further into the differences between these two components later.

The Language: Squirrel

Firmware for the imp is written in a language called Squirrel. Squirrel is an object oriented language similar to Javascript, but unlike most embedded system programming languages we've encountered (namely Arduino). Entering imp development from the world of Arduino may be somewhat jarring. There are no `loop()` or `setup()` functions, instead most actions are event or timer-driven.



There are tons of great examples on Electric Imp's wiki page, and if you're truly interested in learning Squirrel, check out the Squirrel homepage. There's also the Electric Imp API to familiarize yourself with. These are functions and libraries used to perform actions with the imp's GPIO pins and other hardware functionality.

About the Breakout

In order to use an imp, two pieces of hardware are required: the imp card and the **impee**. An impee is the piece of hardware that houses the imp. Aside from having a standard **SD socket** for the imp to slide into, the impee also needs to **provide power** to the imp, and do something with the imp's I/O pins. Our impee for this tutorial is as simple as it gets…a breakout board.

Top and bottom views of the imp breakout.

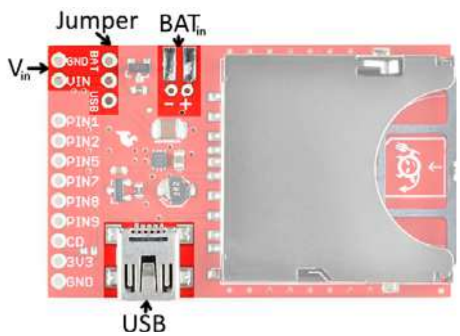
The imp breakout provides the bare minimum you should need to add an imp to your project. There's an SD socket, a step-down voltage regulator, and every I/O pin of the imp is broken out to a 0.1"-spaced header.

Powering the Breakout

A big chunk of the circuitry on the Breakout board is a 3.3V TPS62172 step-down regulator (and the inductor/capacitors supporting it). This regulator allows for input voltages anywhere **between 3.3V and 17V** (voltages in the upper end of that range may produce some heat). It can support up to 500mA of continuous current.

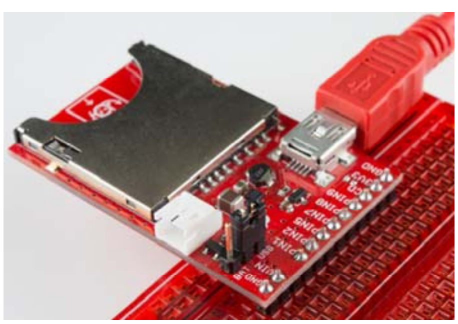
There are three power inputs on the board, all of which, are fed into the on-board 3.3V regulator:

- **"VIN" header** - This standard 0.1" header feeds directly into the 3.3V regulator.
- **Battery input** - These are the pins and pads labeled "+" and "-". The footprint of the two through-hole pins matches up to a PTH 2-pin JST connector, which mates with our LiPo batteries (or AA batteries). This input needs to be selected using the jumper (see below).
- **USB mini-B connector** - This power input should feed a clean, 5V source into the breakout board's regulator. The USB voltage supply can come from either a mini-B cable connected to your computer or a USB wall adapter. This input needs to be selected using the jumper (see below).



Setting the Jumper

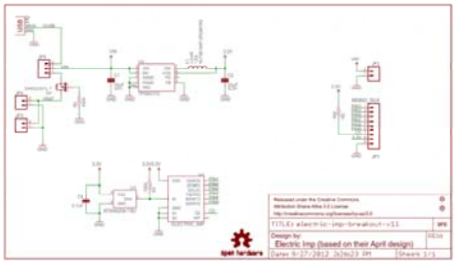
To use either the battery or USB power inputs, a jumper must be set on the board. To use the jumper, first solder a 3-pin male header to the jumper pins. Then use a 2-pin jumper to span from the middle pin, to whichever of the two inputs you'd like to use.



In this image, the jumper is set to apply USB power to the imp breakout. A JST connector was soldered to the battery input pins, in case we want to use a LiPo to power the board.

The Breakout's Schematic

There are three main components to the breakout board: a TPS62172 step-down regulator (U2), the Electric Imp socket (U1), and the ATSHA204 authentication chip (U3).



Electric Imp Breakout Schematic. Click the image to get a larger picture, or click here to view the schematic as a PDF.

Pinout

All of the imp’s GPIO pins are broken out to the 0.1"-spaced header, along with a few related power pins:

- **GND** - Common pin for input voltage
- **VIN** - Input voltage supply fed into regulator
- **PIN1** - imp pin 1 (UART$_{1289}$ CTS, UART$_{12}$ TX, I^{2}C$_{12}$ SCL, SPI$_{189}$ SCLK, DAC, ADC, PWM)
- **PIN2** - imp pin 2 (UART$_{1289}$ RTS, UART$_{12}$ RX, I^{2}C$_{12}$ SDA, SPI$_{257}$ MISO, ADC, PWM)
- **PIN5** - imp pin 5 (UART$_{57}$ TX, SPI$_{257}$ SCLK, DAC, ADC, PWM)
- **PIN7** - imp pin 7 (UART$_{57}$ RX, SPI$_{257}$ MOSI, ADC, PWM)
- **PIN8** - imp pin 8 (UART$_{1289}$ TX, I^{2}C$_{89}$ SCL, SPI$_{189}$ MOSI, ADC, PWM)
- **PIN9** - imp pin 9 (UART$_{1289}$ RX, I^{2}C$_{89}$ SDA, SPI$_{189}$ MISO, ADC, PWM)
- **CD** - Card detect. This signal will connect to GND whenever a card is inserted into the socket.
- **3V3** - 3.3V output from regulator
- **GND** - Common ground

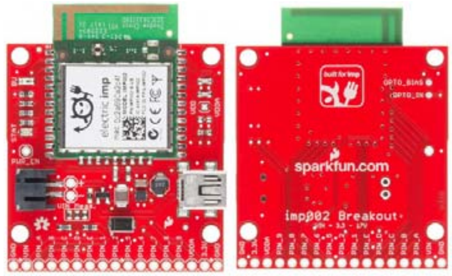
ID Chip

There’s actually one more piece of hardware required of the impee: an ID chip, which provides each impee with a unique identification code. This means that every impee you encounter should include an Atmel ATSHA204 authentication chip. The imp automatically interfaces with this chip every time it boots up, so it can identify which impee it’s plugged into. This actually turns out to be pretty awesome, because the program that an imp runs depends on what impee it’s plugged into. If you had two impees in your house – say controlling an irrigation system and another controlling a coffee machine – one, single imp would run two different programs depending on which machine it was plugged into.

You shouldn’t ever have to fuss with the ID chip. In fact, you can forget we ever said anything about the ATSHA204!

About the imp002 Breakout

The imp002 is a solder-down module version of the original imp card. We have done the hard work of creating a breakout board for you. Now, you just need one board instead of 2 to get started with the electric imp!



We recommend you read the About the imp section to learn what is in the imp, what the Planner is, and a brief overview of the Squirrel language. Like the imp card, the imp002 module contains an embedded ARM Cortex-M3 microprocessor, an onboard WiFi module, and antenna.

The Hardware: 12 Glorious I/Os

We have broken out 12 I/O pins from the imp002 module to standard 0.1" headers. Much like the imp card, these pins can be used for a variety of functions.

imp002 pin table from Imp's Pin Mux's Page

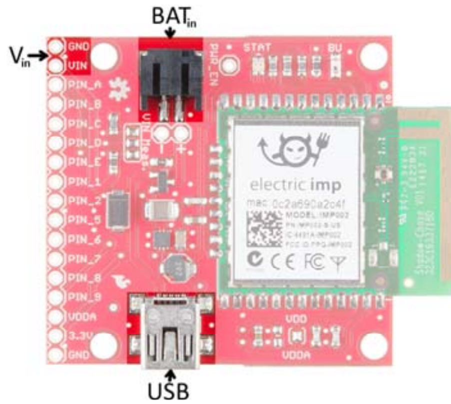
Pin #	UART$_{1289}$	UART$_{57}$	UART$_{12}$	UART$_{6E}$	UART$_{B}$	I^{2}C$_{89}$	I^{2}C$_{12}$	SPI$_{257}$	SPI$_{189}$	DAC	ADC	PWM
A											Yes	
B					RX						Yes	
C												Yes
D												
E				RX								
1	CTS		TX				SCL		SCLK	Yes	Yes	Yes
2	RTS		RX				SDA	MISO			Yes	Yes
5		TX						SCLK		Yes	Yes	Yes
6				TX								
7		RX						MOSI			Yes	Yes
8	TX					SCL			MOSI		Yes	Yes
9	RX					SDA			MISO		Yes	Yes

Powering the imp002 Breakout

The imp002 Breakout Board contains a 3.3V TPS62172 step-down regulator (and the inductor/capacitors supporting it). This regulator allows for input voltages anywhere **between 3.3V and 17V** (voltages in the upper end of that range may produce some heat). It can support up to 500mA of continuous current.

There are three power inputs on the board, all of which, are fed into the on-board 3.3V regulator:

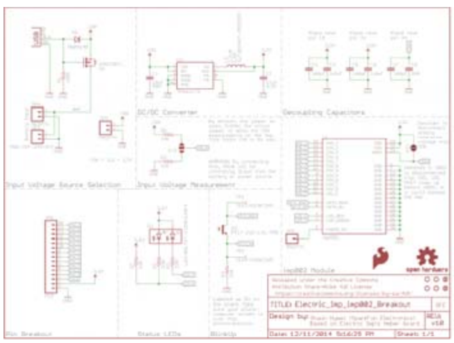
- **"VIN" header** - This standard 0.1" header feeds directly into the 3.3V regulator.
- **Battery input** - These are the pins labeled "+" and "-" as well as the JST connector, which mates with our LiPo batteries (or AA batteries).
- **USB mini-B connector** - This power input should feed a clean, 5V source into the breakout board's regulator. The USB voltage supply can come from either a mini-B cable connected to your computer or a USB wall adapter.



NOTE: There is a voltage selector circuit on the imp002 Breakout Board that will automatically use whichever voltage is higher: battery or USB. Be aware that the circuit does **NOT** charge the battery, it just prevents current flowing back into the source with the lower voltage (i.e. a short).

The imp002 Breakout's Schematic

There are a number of circuits used to support the imp002, all of which can be found on the imp002 Breakout Board.



electric imp imp002 Breakout Schematic. Click the image to get a larger picture, or click here to view the schematic as a PDF.

- **Input Voltage Source Selection** - automatically switches between USB and battery input (whichever voltage is higher)
- **Pin Breakout** - Power and I/O pins from the imp002 module
- **DC/DC converter** - the TPS62172 buck regulator and supporting components
- **Input Voltage Measurement** - the jumper can be soldered to allow VIN measurements on PIN A
- **imp002 module** - the imp module and decoupling capacitors
- **Status LED** - the red/green LED required by the imp to display its status (connecting, error, etc.)
- **BlinkUp** - Light sensor for sending WiFi credentials to the imp002 module

Pinout

All of the imp's GPIO pins are broken out to the 0.1"-spaced header, along with a few related power pins:

- **GND** - Common ground
- **VIN** - Input voltage supply fed into regulator
- **PIN_A** - imp002 pin A (ADC)
- **PIN_B** - imp002 pin B ($UART_B$ RX, ADC)
- **PIN_C** - imp002 pin C (PWM)
- **PIN_D** - imp002 pin D
- **PIN_E** - imp002 pin E ($UART_{6E}$ RX)

- **PIN_1** - imp002 pin 1 (DAC, $UART_{1289}$ CTS, $UART_{12}$ TX, I^2C_{12} SCL, SPI_{189} SCLK, DAC, ADC, PWM)
- **PIN_2** - imp002 pin 2 ($UART_{1289}$ RTS, $UART_{12}$ RX, I^2C_{12} SDA, SPI_{257} MISO, ADC, PWM)
- **PIN_5** - imp002 pin 5 ($UART_{57}$ TX, SPI_{257} SCLK, DAC, ADC, PWM)
- **PIN_6** - imp002 pin 6 ($UART_{6E}$ TX)
- **PIN_7** - imp002 pin 7 ($UART_{57}$ RX, SPI_{257} MOSI, ADC, PWM)
- **PIN_8** - imp002 pin 8 ($UART_{1289}$ TX, I^2C_{89} SCL, SPI_{189} MOSI, ADC, PWM)
- **PIN_9** - imp002 pin 9 ($UART_{1289}$ RX, I^2C_{89} SDA, SPI_{189} MISO, ADC, PWM)
- **VDDA** - ADC reference voltage. Connected to 3.3V by default.
- **3.3V** - 3.3V output from regulator
- **GND** - Common ground

> **IMPORTANT:** If you disconnect the VDD/VDDA jumper, you **MUST** bring up the VDD (3.3V) power before bringing up the VDDA reference voltage. Additionally, if VDDA is greater than VDD (3.3V), it might cause damage to the imp002 module.

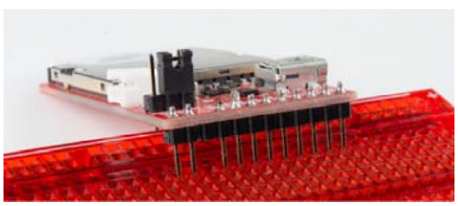
Hardware Hookup

The hardware hookup approach in this guide is just one of many ways to use the board. The breakout is made to be a versatile extension of the imp. You can connect whatever you want to the imp pins, and power the board however your project requires.

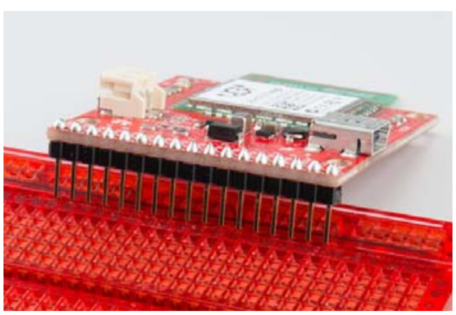
Solder Headers

In order to do much with the input/output capability of the imp, you'll need to solder to the broken out pins. If you want to use the imp Breakout with a breadboard or perfboard, 0.1" male headers make for a good choice. Depending on your application, you could swap the headers with wire, female headers, screw terminals, or a variety of other connectors.

We're going to solder male headers into the board, so we can use it with a breadboard later on.



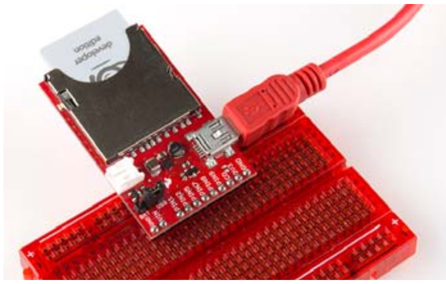
Pins soldered onto the imp Breakout Board



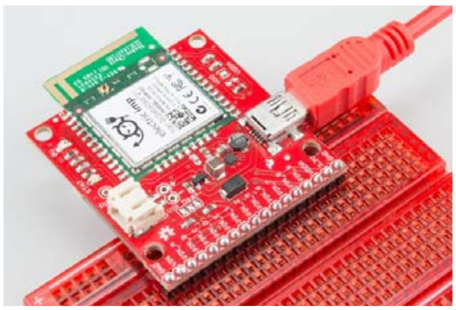
Pins soldered onto the imp002 Breakout Board

Apply Power

Depending on what you want to use for your power source there are a few options here. You could use the on-board USB connector. Or you could solder down a 2-pin JST connector, and plug battery (LiPo or AA) into the board to make it mobile. If you go with either of those options on the imp card Breakout, you'll also need to set the jumper (the imp002 Breakout will automatically select the higher voltage).



Note how the jumper is set. We're using USB to power the imp in this picture.



No need to set a jumper on the imp002 Breakout Board! We're using USB to power the imp002 here.

Alternatively, you can apply power straight to the headers labeled "VIN" and "GND". This pin bypasses the jumper and goes straight to the regulator.

Plug in the imp!

If you have the original imp card, plug the imp card in so the suspicious little imp logo is facing up. If you've got power to the board, once plugged in, the imp should start blinking orange. If there's no blinking on the card, it's probably not getting any power. Double-check that the jumper is set correctly.

If you have the imp002, the status LED should start blinking orange as soon as you apply power.

What's all that blinking signify? How do we get the imp connected to our wireless network? Read on!

BlinkUp

Blink Codes

The imp has an internal red/green LED, which is used to tell the world what state it's currently in. If you've just plugged the imp in, and haven't told it how to get on your WiFi network, it should be blinking orange (red/green simultaneously). Here are the rest of the codes to look out for:

imp blink codes (from the imp blinkup guide)

Color	Speed	imp State
Orange	1 Hz	No WiFi settings

Green	Single Pulse	Successfully received configuration via Blinkup.
Red	Triple-pulse	Failed to receive configuration via Blinkup.
Red	1 Hz	Attempting to connect to WiFi.
Red, Orange, Off	1 Hz	Getting IP address (via DHCP).
Orange, Red, Off	1 Hz	Got IP address, connecting to server.
Green	0.5 Hz	Connected to cloud (turns off after 60 seconds).
Red	2 Hz	Connection lost, attempting to reconnect.
None		Normal operation

Let’s make that LED blink green! Time to send a BlinkUp.

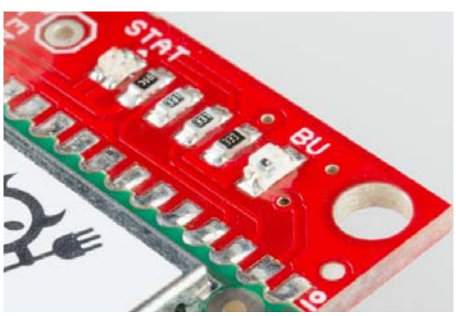
BlinkUp

To get your imp connected to your WiFi network as well as the online imp servers, you need to go through the process Electric Imp calls **commissioning**. There’s a great write-up on the commissioning process over on Electric Imp’s Getting Started page. Here’s the gist of it, as well as a few tips.

Before you begin, you’ll need to **make an Electric Imp account** by visiting the IDE page.

Updating the imp with your WiFi credentials is a unique process. The imp card has a built-in light sensor, looking out of the little window on the short, flat edge of the imp. The imp002 has an external light sensor built into the breakout board. The light-sensor can be used to process small amounts of precisely modulated data in the form of a blinking light.

The light sensor is just behind the translucent window on the edge of the imp card.



The light sensor on the imp002 is located to the left of the module, with the lable “BU” (for BlinkUp).

To generate this blinking light, you need the Electric Imp app installed on your smartphone (iOS or android). Go download that app if you haven’t already!

Follow the directions in the app, and prepare to update the imp with your WiFi network. Then, when your settings all look correct, hit the **Send BlinkUp** button. Quickly place the screen of the phone as close to the imp’s light sensor as possible.

Avert your eyes! Unless you enjoy staring into bright, white strobing lights.

Similar warnings about white strobing lights.

If all goes well, there should be a very short green blip of the LED, followed by a few blinks of red and orange. When the imp starts blinking green once a second, you know you've got your imp commissioned yay!

Troubleshooting

If you're imp isn't yet in the blinky green phase, use the LED blink codes to find out where it's failing. Here are some recommended steps, depending on the failure point:

- Connecting to the server (orange, red, off) - Make sure there's no firewall blocking the imp's way to the Internet (and make sure your WiFi network has an Internet connection in the first place).
- Getting IP address via DHCP (red, orange, off) - Double check your WiFi password.
- Attempting WiFI connection (red) - Double check your WiFi network name (SSID).

If all of the above are set correctly, try sending the BlinkUp one more time. We've found that it helps to close out all other app, or even try **resetting your phone** if it continues to fail.

More troubleshooting information can be found on Electric Imp's site.

Example 0: Hello World

Now that your imp is commissioned, it's time to upload your first bit of code!

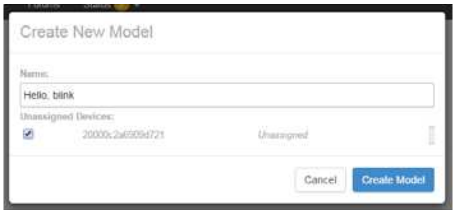
As with any new development platform, our first goal is to make sure we can make an LED blink. If you can make an LED blink, you're well on your way to spinning motors or communicating with sensors.

Using the IDE

To begin, go to Electric Imp IDE, and log in if you haven't already.

If your Electric Imp was successfully commissioned, you should see your imp device appear under *Unassigned Devices* on the left-hand side.

Click the *Create New Model* button.



In the name field, type "Hello, blink" for the name of our model. Check the box next to our device under *Unassigned Devices*. Click *Create Model*.

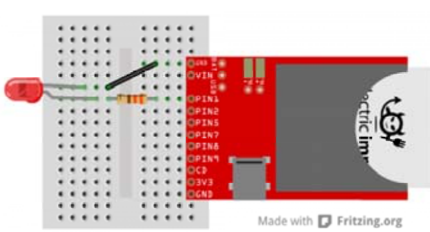
Now, on the left side, you should see a new tab called *Hello, blink*. Select than, then click your imp name. This is the standard view of the imp IDE. It's split into three sections:

1. **Agent** – This is code that runs external to your imp, in the **cloud**. You can offload server tasks, like HTTP requests, here. There are built in functions to aid in communication between imp and agent.
2. **Device** – This is the code that your **imp** runs. This is where you do all of your hardware control, like writing pins high and low, or reading inputs.
3. **Log** – This is where messages and errors are printed (using the `server.log()` function).

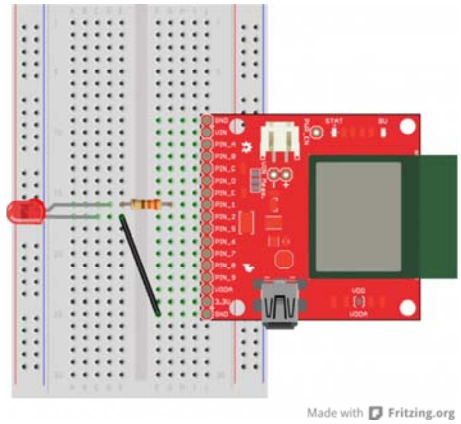
Now we're ready to load some code and blink some LEDs!

The Circuit

The circuit for this example is very simple. We only need to connect an LED to pin 1. Don't forget your current-limiting resistor (330 Ω)!



imp circuit



imp002 circuit

Any of the imp's I/O pins would work for this example. After working with the code, see if you can modify it to blink on other pins (or all of them!).

Hello, blink Code

We'll only be working with the *Device* portion of the IDE right now. Copy and paste the code below into the middle section of your window.

```
/* Hello, Blink
   by: Jim Lindblom
   SparkFun Electronics
   date: October 31, 2013
   license: Beerware. Use, reuse, and modify this code howeve
r you see fit.
   If you find it useful, buy me a beer some day!

   This is an Electric Imp hello, world blink sketch. It'll bl
ink an LED
   connected to pin 1, once every second.
*/

////////////////////////////////////////
// Global Variables                   //
////////////////////////////////////////
ledState <- 0;

////////////////////////////////////////
// Function definitions               //
////////////////////////////////////////

// Loop constantly updates the LED. If ledState is 1, we'll tu
rn the LED on and
// set ledState to 0. Vice-versa is ledState is 0 coming in. T
his function
// schedules a wakeup in 1 second, and calls itself again.
function loop()
{
    if (ledState)
    {
        hardware.pin1.write(1); // Write pin 1 high
        ledState = 0; // Flip ledState
    }
    else
    {
        hardware.pin1.write(0); // Write pin 1 low
        ledState = 1; // Flip ledState
    }

    // This must be called at the end. This'll call loop() aga
in in 1s, that way
    // it'll actually loop!
    imp.wakeup(1.00, loop);
}

////////////////////////////////////////
// Setup Stuff: Runs first at startup //
////////////////////////////////////////
hardware.pin1.configure(DIGITAL_OUT);   // Configure Pin 1 as
digital output

loop(); // Call loop, and let the program go!
```

Then hit the ***>Build and Run*** button up top, and enjoy the blinks.

Shortcut heads up! If you're a neurotic **CTRL+S** saver, the standard save shortcut *does* **save**, but it also attempts to **build and run your code**. If successful, it'll upload the code and immediately start running on your imp. If there's an error, you'll start hearing about it in the log window.

Into the Code

If you're only used to working with Arduino sketches, this code may make very little sense, Electric Imp programs have a very different "flow" to them. Begin by looking at the **2 lines of code at the bottom** (under the "Setup Stuff" header). This is actually where our imp starts when it begins to run it's program. Everything above is simply a function or variable definition.

The majority of this code deals with the imp's pin class, which handles all of the I/O control. If you're used to using Arduino GPIO's, the imp's API isn't too different. You have to set the pin up as either an input or output, analog or digital. Then write or read to the pin accordingly.

At the end of the setup, we make a call to a `loop()` function, which is defined above. `loop()` is simple, it checks a global variable named `ledState`. If `ledState` is 1 we turn the LED on, if it's 0 we turn the LED off.

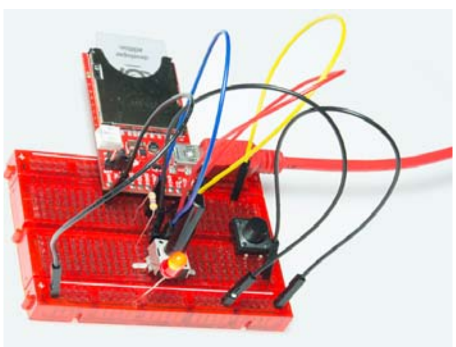
To write a pin high or low, we call the `hardware.pin1.write([0:1])` function. You can probably extrapolate from that how to control the other five pins.

The special sauce making loop *actually* loop is the last line of code in the function: `imp.wakeup(1.00, loop)`. The `imp.wakeup` function puts the imp to sleep, but sets a timer. When the timer goes off, the requested function (`loop` in this case) function is called from its beginning. In this case we set the timer to 1.00 seconds, so `loop()` should run once a second. This is really the only way to make the imp "loop" like an Arduino might.

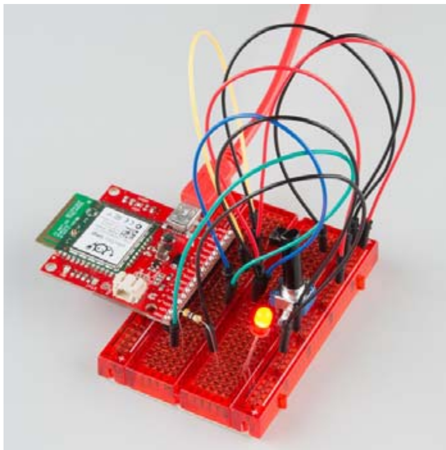
Check out the comments in the code for a more in-depth overview of each function call. Or, for more information, check out Electric Imp's API reference.

Example 1: I/O Control

The imp can do most anything an Arduino or similar microcontroller can. It's got analog-to-digital converters, PWM, SPI, I²C, UARTs, and it even has digital-to-analog converters. In this snippet of example code, we'll dig further into the imp's I/O control delving into digital and analog input/output.



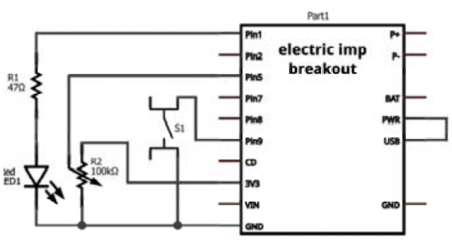
imp and Breakout Board connected the Example 1 circuit



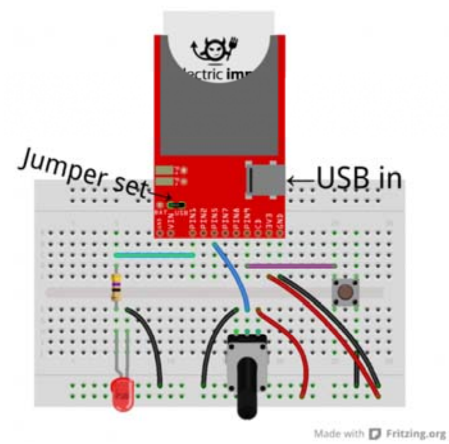
imp002 connected the Example 1 circuit

The Circuit

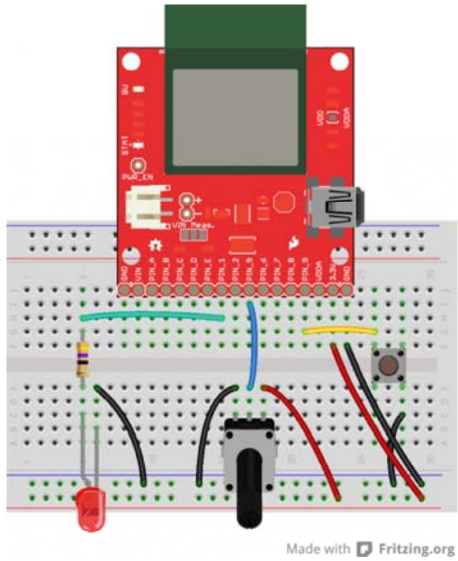
The setup for this example code requires three unique components: an LED, potentiometer, and a button (plus a current-limiting resistor for the LED). Here's a fritzing diagram and schematic (click to see it bigger) for our circuit:



imp schematic



imp circuit

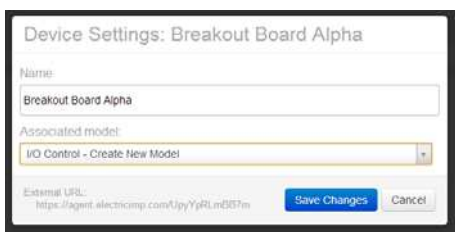


imp002 circuit

Make sure the imp is getting power. USB is usually the quickest/easiest way to apply power to the breakout board, but you'll need to set the jumper accordingly on the breakout.

The IDE

To create a new piece of code, we need to create a new "model" and associate it with our Breakout Board impee. To do this, **hover over your impee** and **click the "settings" gear**. The familiar *Device settings window should pop up. Under the* Associated model: *box, create a new model named* I/O Control*. Then click* Save Changes_.



This will create a new tab on the left side labeled *I/O Control*. If you expand that tab, you'll see that the Breakout Board impee has be reassigned there.

The Code

Once again, we'll only be using the *Device* portion of the IDE. Copy and paste everything from the below box, into your *Device* window and click *Build and Run* up top.

```
/* Digital Input, Analog Input, PWM Output Example
   by: Jim Lindblom
   SparkFun Electronics
   date: July 15, 2013
   license: Beerware. Use, reuse, and modify this code howeve
r you see fit.
   If you find it useful, buy me a beer some day!

   This is a simple piece of code which uses an LED, potentiom
eter, and button.
   The LED connects to pin 1 through a 47 ohm resistor. The ca
thode of the LED should connect to ground.
        This means writing pin 1 will turn the LED on, and wri
ting it to 0 turns the LED off.
   The button connects on one side to pin 9, and the other pi
n of the button goes to ground.
        We'll use the internal pull-up resistors on pin 9 to b
ias the button high.
        When the button is pressed, pin 9 should read as low.
   The wiper of the potentiometer is connected to pin 5. The o
ther two pins of the pot should be
        connected to +3.3V and GND. This'll make the voltage a
t pin 5 adjustable from 0-3.3V.
*/

////////////////////////////////////////
// Function definitions               //
////////////////////////////////////////

local ledState = 1; // Says local, but think of this as a glob
al var. Start with LED on

// function pin9Changed() will be called whenever pin 9 goes f
rom high->low or low->high
function pin9changed()
{
    local buttonState = hardware.pin9.read(); // Read from th
e button pin

    if (buttonState == 0) // Button will read low if pressed
    {
        ledState = ledState ? 0 : 1;    // Flip flop ledState
        server.log("Button pressed!");
    }
    else // Otherwise button was released, no action
    {
        server.log("Button released");
    }
}

// Loop constantly updates the LED. If ledState is 1, we'll re
ad the pot, and set the LED brightness accordingly.
// If ledState is 0, we'll just turn the LED off. ledState is
updated in the pin9Changed() function.
function loop()
{
    if (ledState == 1)
    {
        local rawValue = hardware.pin5.read();  // Read from t
he potentiometer. Returns a value between 0 and 65535.
        rawValue /= 65535.0;  // Make rawValue a % (and a floa
t). The pin write function requires a value between 0 and 1.
        hardware.pin1.write(rawValue);  // Pin 1 is already co
nfigured as PWM, write potentiometer value
```

```
    }
    else
    {
        hardware.pin1.write(0);  // Write pin 1 low -- LED off
    }

    // This must be called at the end. This'll call loop() aga
in in 10ms, that way it'll actually loop!
    imp.wakeup(0.01, loop);
}

////////////////////////////////////////
// Setup Stuff: Runs first at startup //
////////////////////////////////////////
hardware.pin1.configure(PWM_OUT, 0.0005, 0.0);   // Configure
Pin 1 as PWM output, 5ms period, 0% high (off)
hardware.pin5.configure(ANALOG_IN); // Configure pin 5 as anal
og input
hardware.pin9.configure(DIGITAL_IN_PULLUP, pin9changed); // Co
nfigure pin 9 as digital input (with pull-up enabled). On chan
ge it'll call function pin9changed().
imp.configure("LED Trigger Wiper", [], []);

loop(); // Call loop, and let the program go!
```

The code creates an adjustable-brightness LED controller. The brightness of the LED is adjusted by turning the potentiometer. Pressing the button will turn the LED on and off.

Explaining the Code

The skeleton of this code acts a lot like that of *Hello, blink*. The function definitions are up top, the setup stuff runs at the bottom, and `loop()` is called at the beginning. `loop()` continually calls itself, using the `imp.wakeup(0.01, loop)` function call, every 10 ms.

The `loop()` function again relies on an `ledState` variable. If `ledState` is 1, we read the potentiometer voltage, and adjust the brightness of our LED accordingly.

The `ledState` variable is flip-flopped in the `pin9changed()` function. This is like an **interrupt**. It's called whenever the state of pin 9 changes – if it goes from high to low, or low to high. When setting up pin 9 as a digital input, we added this function as the one to be called when the state change occurred.

Check out the comments in the code for a more in-depth overview of each function call. Or, for more information, check out Electric Imp's API reference.

Enough hardware stuff! The next two examples will make use of the imp's greatest feature…it's web connectivity.

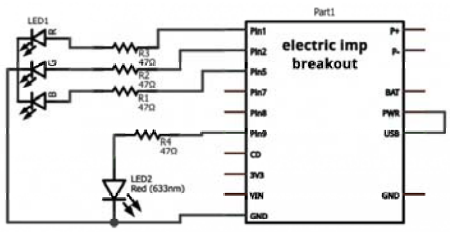
Example 2: Web Control (Request)

Some of the most fun you can have with the imp is connecting it to the Internet, and interfacing it with web pages. In this example, we'll use a simple HTML/Javascript web page to control some LEDs connected to the imp.

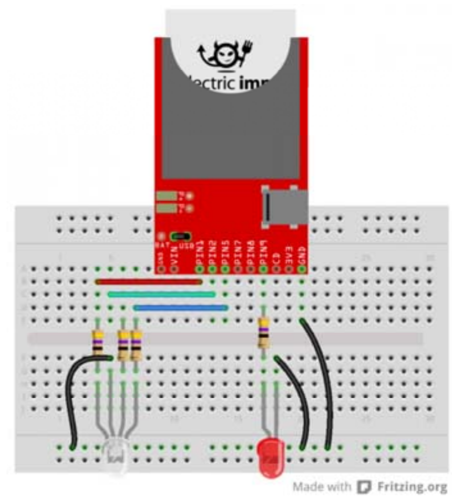
This time, we'll not only be writing code for the imp, but the **agent** as well. This example code will show how to pass data from the imp to the agent, and how to write a simple web page to interact with the agent half of the code.

The Circuit

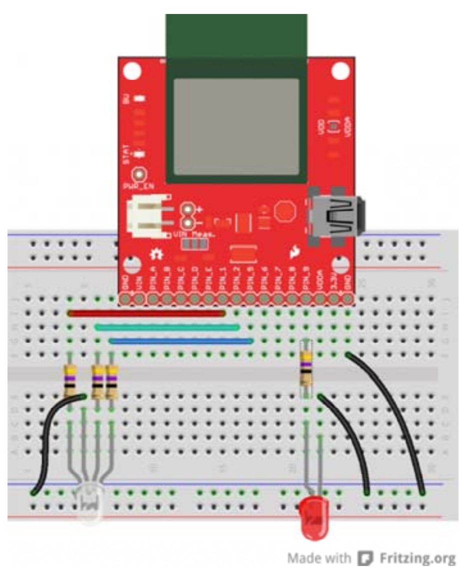
The circuit for this example is very simple: a common-cathode RGB LED is connected to the imp's pins 1, 2, and 5 (red, green, and blue anodes respectively), and another basic red LED is connected to pin 9 of the imp. Don't forget to add some current limiting resistors (in the range of 50-100Ω)!



imp schematic



imp circuit



imp002 circuit

The imp (Device) Code

Create a new model, as you did in the last example. We'll call this one *LED Web Control*. Copy and paste the code below into the ***Device*** section of the IDE.

```
/* Electric Imp Web-controlled LEDs
    by: Jim Lindblom
    SparkFun Electronics
    date: November 1, 2013
    license: Beerware. Please use, reuse, and modify this cod
e.
    If you find it useful, buy me a beer some day!

    This is a simple Electric Imp example, which shows how to
interface
    the imp with an agent and webpage. This example code goes
hand-in-hand with
    an HTML webpage. Check out this page for more informatio
n:
    https://learn.sparkfun.com/tutorials/electric-imp-breakout
-hookup-guide/example-2-web-control
    This will show how you can use html color, text, and radi
o form inputs
    to control LEDs on/off, PWM them, and set a timer to turn
them off.

    Circuit:
    A common cathode RGB LED is connected to the imp's pins
1, 2, and 5.
    The red anode connects to 1 through a 47 Ohm resistor, gre
en 2, and blue 5.
    The cathode of the LED connects to ground.
    Another simple, red LED is connected to the imp to imp pi
n 9, through
    another 47 Ohm resistor. The cathode of the LED is grounde
d.
*/

imp.configure("LED Web Control", [], []); // Configure the imp

///////////////
// Pin Setup //
///////////////
// Setup reference variables for our pins:
redPin <- hardware.pin1;   // R of RGB
greenPin <- hardware.pin2; // G of RGB
bluePin <- hardware.pin5;  // B of RGB
ledPin <- hardware.pin9;   // Lonely red LED

// Configure our pins:
ledPin.configure(DIGITAL_OUT);        // Simple digital output
redPin.configure(PWM_OUT, 0.01, 0);   // PWM output 10ms cloc
k, off
greenPin.configure(PWM_OUT, 0.01, 0); // PWM output 10ms cloc
k, off
bluePin.configure(PWM_OUT, 0.01, 0);  // PWM output 10ms cloc
k, off

/////////////////////////////////
// Agent Function Declarations //
/////////////////////////////////
// setLed will turn the lonely red LED on or off.
// This function will be called by the agent.
function setLed(ledState)
{
    ledPin.write(ledState);
}

// setRGB will take a table input, and set the RGB LED accordi
```

```
ngly.
// the table input should have parameters 'r', 'g', and 'b'.
// This function will be called by the agent.
function setRGB(rgbValue)
{
    bluePin.write(rgbValue.b/255.0);
    redPin.write(rgbValue.r/255.0);
    greenPin.write(rgbValue.g/255.0);
}

// setUser will print out to the log the name of the LED chang
er
// This function will be called by the agent.
function setUser(suspect)
{
    server.log(suspect + " set the LEDs.");
}

// setTimer will turn the LEDs off after a specified number o
f seconds
// This function will be called by the agent.
function setTimer(time)
{
    if (time != 0)
        imp.wakeup(time, ledsOff); // Call ledsOff in 'time' s
econds.
}

///////////////////////////////////
// Important Agent Handler Stuff //
///////////////////////////////////
// Each object that the agent can send us needs a handler, whi
ch we define with
// the agent.on function.  The first parameter in agent.on is
an identifier
// string which must be matched by the sending agent. The seco
nd parameter is
// the name of a function to be called. These functions are al
ready defined up
// above.
agent.on("led", setLed);
agent.on("rgb", setRGB);
agent.on("user", setUser);
agent.on("timer", setTimer);

//////////////////////
// Helper Functions //
//////////////////////

// ledsOff just turns all LEDs off.
function ledsOff()
{
    ledPin.write(0);
    redPin.write(0);
    greenPin.write(0);
    bluePin.write(0);
}
```

The key bit of new code in this example is the `agent.on` function call. Run during the setup portion of the code, these function calls set up a **handler function** to be called whenever the agent sends a specific string to the imp. For example, the `agent.on("led", setLed);` functions says that whenever a message tagged with an “led” string is received from the agent, call the `setLed()` function.

How do we send messages from the agent to the imp? Looks like it's time to start using the other half of the IDE window…

The Agent Code

The agent is a piece of squirrel code living and running in the Electric Imp cloud. While the imp is managing all of its hardware pins, the agent can be off mingling with other servers and dealing with Internet traffic. There are built in functions which allow the imp to send data to the agent, and vice-versa.

In this example, we'll set the agent up to listen for HTTP requests. Upon receiving a request, the agent will parse the query, and relay the important information back to the imp.

Copy and paste this code into the ***Agent*** half of your *LED Web Control* model: