**Electric Imp Tutorial – Getting Started with IoT.**

**1. Introduction.**

The electricimp is one of the many devices that can be used to provide computing power to the sensor layer of an IoT framework. It is arguably much simpler to use than the Arduino or Raspberry Pi although not as flexible.

The Electric Imp is a micro-processor board that has 6 configurable input or output channels allowing control of several devices. These are connected through the Electric Imp and on-board WiFi to the company's server (in California) which in turn can connect to your web service and then onto your mobile phone.

The programming of the electric imp is carried out online through a simple web based IDE using the Squirrel programming language. Whilst you may not have seen Squirrel previously it looks like C and Java so is easy to programme. For further details of how the Electric Imp works look at electricimp.com.

The Electric Imp is designed to be built into custom made hardware and in the process the electronics will be minimised. What we will be using is a development environment designed for prototyping and testing applications rather than developing production systems. The hardware consists of two parts, the electric imp that looks like an SD card but IS NOT and the April development board.

Along with the electric imp itself there is a unique agent, running on the electric imp servers in California. Communications to and from the Internet are channelled through this Agent. The Agent can be thought of as the key to the IoT. The device provides the sensor layer but the agent provides the communication layer interface.

These notes are not complete and can be supplemented from the electricimp.com web site.

Click on “Dev Centre” and then on “Getting Started” to see their introduction.

**1. Getting Started with the electric imp.**

**1.1 Register as a Developer.**

To use the electric imp system you will need an account on the electricimp.com site. Register for an account and login. You will be placed in the software development environment.

**1.2. Get the Electric Imp Hardware online.**

Plug the electric imp into the April board and connect to the power supply. The device is now on but not connected to the Internet. To do this you need to provide the ElectricImp with the WiFi credentials. This is done using the Electric Imp’s unique “blink-up” process through your smart phone.

**Blink-Up**

This is the stage to get the Electric Imp online. You will need an App on your phone for this. Download the App from Google or Apple. Start the App and enter the details for the WiFi system. You cannot use the standard “eduroam” network due to the security features on this network. In room 2021 there is a special network that you should use with the electric imp.

The SSID is ***RM2021*** and password **testnet*2021.***

The APP transfers the data to the device by flashing the screen. It is best to set the screen as bright as possible and to do this in a shaded or dark area. DO NOT LOOK AT THE SCREEN DURING THIS BLINK\_UP PROCESS. Once finished look at the device and a green light on the imp will flash and then go to solid green. That is the device connected to the WiFi and thus the Internet.

You are working in a shared environment – you have to share the electric imp with others so :

* Make sure you clear device settings prior to “blink up”
* Make a note of the device number once connected
* Try to use the same Electric Imp each week.

You will see on your IDE that you have an online device. Note the device number and the associated URL of the Agent.

**1.4 Exercise – Hello World**

Whenever you write your first program in a strange language you must start with a Hello World program – it is the law.

Programmes are called models in electric imp terms. You need to create a new model associated with your device and call it HelloWorld. Copy the code below into the code section of the development environment, compile and build. You will see the results in the device log.

|  |
| --- |
| // Hello World example code    // Display our important message  server.log("Hello, World!");    // End of code. |

So first programme written – now for some real work.

**2. Binary Output - Programmable Red and Green LEDs.**

**2.1 Simple Flashing Led.**

Connect a RED led to pin 5 and a GREEN led to pin 7. You can do this using wires, a breadboard or using the special Abertay board. Using the code below inside a new model you will see that the led flashes every 5 seconds.

|  |
| --- |
| // define led to be the pin in question (if a different pin this is the only change)  led5 <- hardware.pin5 ;  // define the state of the led  state <- 0;  // configure the pins as digital output pins  led5.configure(DIGITAL\_OUT);  // Define the loop flash function  function flash() {  // code to swap state  state = 1-state ; // reverse the state  led5.write(state) ;  server.log("Flash : " + state);  imp.wakeup(5.0, flash);  }    // Start the flash loop  flash(); |
|  |

Now change this to flash the green led, then both together and finally for the adventurous flash alternatively.

**2.2 Switch RED led from a URL**

In the exercise above you used the electric imp as a stand alone device. What makes the electric imp device exceptional, is that it is paired to a unique agent on the Electric Imp cloud. You can send data to and from the Imp via this agent. Look at the development environment and you will see the unique URL for your device.

To switch the red led (pin 5) on and off you need 2 pieces of code; one for the device and one for the agent. Paste the code below into a new model

|  |
| --- |
| // define led to be the pin in question (if a different pin this is the only change)  led5 <- hardware.pin5 ;  // configure the pins as digital output pins  led5.configure(DIGITAL\_OUT);    // function to turn LED on or off  function setLed(state) {  led5.write(state) ;  server.log("Light : " + state);  }    // register a handler for "led" messages from the agent  agent.on("led", setLed); |

|  |
| --- |
| // define the http handler  function requestHandler(request, response) {    local state = request.query["state"].tointeger() ;  device.send("led", state);  response.send(200, "OK");  }  // register the http handler  http.onrequest(requestHandler); |

Now copy and paste the agent code into the agent window in the IDE. Build and run. Now all is set up. But nothing will happen because we need an external stimulus to switch the light. You will note a URL of the agent within the IDE.

Write a short, very short, web page containing two lines of HTML

<a href=” [https://agent.electricimp.com/N72wa71gJ?state=1”>Switch](https://agent.electricimp.com/N72wa71gJ?state=1) on</a>

<a href=” [https://agent.electricimp.com/N72wa71gJ?state=0”>Switch](https://agent.electricimp.com/N72wa71gJ?state=0) off</a>

Replacing the “N72wa71gJ” with the code of your Agent otherwise you will switch my led.

**2.3 Exercise Switch either Red or Green LED**

The next stage is to switch either the Red or Green leds. One way is to add a second parameter to the URL and process this in a similar way. Another way is to use the led parameter and code so that :

* 0 – switch red led off
* 1 – switch red led on
* 2 – switch green led off
* 3 – switch green led on

When you come to pass the data from the agent to the device you can only use a single parameter so this coding makes sense.

Now test to ensure you can switch red and green lights.

**2.4 Mobile Web App.**

The final stage today is to write a mobile web page that can switch the lights. Make your page better than a list of 4 URLs !!

Use your app to switch the lights. Whilst you are near to the leds, think of the journey that the data travels.

* From your phone to the local mast to Vodafone (or EE or Three or O2) switching centre in London(?) then onto the electric imp site in California then back to the imp in Abertay to switch the light. And all this is such a short time with little delay.

**3. Binary Input – press a switch.**

**3.1 Device Code.**

Connect a switch to pin 1 of the electric imp. The code below shows how this is processed. Load this into a model and see that when you press the switch a message appears on the console.

|  |
| --- |
| // 4 parallel switches  button1 <- hardware.pin1 ;  function buttonPress1() {  local state = button1.read();  local code = 1 \* 2 + state ; // 1 is pin number and I am coding the input  server.log ("switch " + code ) ;  // this is a simple debounce  imp.sleep(0.5) ;  }    button1.configure(DIGITAL\_IN\_PULLDOWN, buttonPress1); |

**3.2 System Code**

Pressing a simple button and this fact displaying on a console is not very interesting. The fact that the switch is pressed will now be sent to the agent and then to your web site. When the switch is pressed this data has to be moved through the electric imp system so it can be seen on the mobile. Using the agent.send( .. ) and device.on( .. ) code to send the fact that the switching is sent to the agent.

|  |
| --- |
| // 4 parallel switches  button1 <- hardware.pin1 ;  function buttonPress1() {  local state = button1.read();  local code = 1 \* 2 + state ; // 1 is pin number  server.log ("switch " + code ) ;  agent.send ( "swdata", code ) ;  // this is a simple debounce  imp.sleep(0.5) ;  }    button1.configure(DIGITAL\_IN\_PULLDOWN, buttonPress1); |

|  |
| --- |
| // define the URL of the server code  const url = "http://mayar.abertay.ac.uk/~15999999/button/controller/button.php" ;  function log (code) {  local headers = { "Content-Type" : "application/json"} ;  local state = code % 2 ;  local pin = (code - state) / 2 ;  server.log ("state"+state+" pin"+pin) ;  local body = { "pin" : pin , "state" : state } ;  local jsonBody = http.jsonencode(body) ;    // POST the values  local request = http.post(url, headers, jsonBody);  local response = request.sendsync();  server.log(response.statuscode + ": " + response.body);  }  device.on("swdata", log) ; |

In the agent code there is a URL where the data is sent. You need to change this URL to your script. So you will need to write a script that will take the data and store this in your database. The code below will store the data in my database through the createSwitchState() method. You will need to write your own.

|  |
| --- |
| <?php  include("../model/api.php") ;  // decode the json body from the request  $jsonbody = file\_get\_contents('php://input') ;  $jsonobj = json\_decode($jsonbody) ;  $state = $jsonobj -> state ;  $pin = $jsonobj -> pin ;  $result = createSwitchState($state, $pin) ;  ?> |

**3.3 Accessing the state on the phone – burglar alarm.**

From the code above you will see the latest state of the switch in your database. Add another switch to pin2 and amend the code above so that it records the status of switch on pin2 as well as that on pin1. This is the makings of a burglar alarm. In addition, you need to be able to set the burglar alarm and if it is set and either of the switches are released then you have a burglar. Add to your mobile script to make a reasonable burglar alarm system.

When the burglar alarm is activated and a switch is released it means you have a burglar and so an important message needs to be sent to the webpage. This is properly carried out using Notifications or indeed sending SMS message to the user. Alternately you can do some polling of the server via AJAX. Just for now a refresh button on your page will do.

----------

Note that there is a problem in that the electricimp needs to execute a script on the mayar.abertay.ac.uk server. There is a firewall in the way. So you will have to use your own server to do this. Other options are to explore using UbiDots to achieve the same effect.

**4. Numerical Input - Temperature Readings.**

The next stage is to use the temperature gauge that you can attach to pins 8 and 9, from the secondary board. Sample code below will show the temperature recorded by the 2 gauges on the server log.

|  |
| --- |
| // Read the internal and external temperature from a thermister.  // B57861S0103F040 NTC Thermistor with matched resistor  id <- hardware.getdeviceid();  // Configure Pin  // thermistor connected to pin8(external) and pin9(internal)  external <- hardware.pin8 ;  external.configure(ANALOG\_IN);  internal <- hardware.pin9 ;  internal.configure(ANALOG\_IN);    // Define the relevant constants for this thermister  const aconst = 65535.0 ;  const bconst = 3988;  const t0const = 298.15;  vconst <- hardware.voltage() ;    // function to read the voltage and convert to degrees Centigrade  function getTemp() {  // read the value  local v8 = external.read() ;  local v9 = internal.read() ;    // convert the voltage to temperature in centigrade.  v8 = v8 \* vconst / aconst ;  v9 = v9 \* vconst / aconst ;  local r8 = 10000.0 / ( (vconst / v8) - 1);  local r9 = 10000.0 / ( (vconst / v9) - 1);  local ln8 = math.log(10000.0 / r8);  local ln9 = math.log(10000.0 / r9);  local t8 = (t0const \* bconst) / (bconst - t0const \* ln8) ;  local t9 = (t0const \* bconst) / (bconst - t0const \* ln9) ;  local t8 = t8 - 273.15 ;  local t9 = t9 - 273.15 ;    // send the value to the server log  local c8Str = format("%.01f", t8) ;  local c9Str = format("%.01f", t9) ;    // send the values to the agent  local message = {"device" : id ,  "external" : c8Str ,  "internal" : c9Str } ;  server.log(message) ;  // get the imp to sleep and wake up every 20 s  imp.wakeup(20, getTemp);  }    getTemp() ; |

This is not very clever, what we need is to send the data to a server. Use the agent code below :

|  |
| --- |
| // define the URL of the server code  const url = "http://mayar.abertay.ac.uk/~17999999/temp2/controller/temp2.php" ;  function log (message) {  local headers = { "Content-Type" : "application/json"} ;  local jsonBody = http.jsonencode(message) ;    // POST the values  local request = http.post(url, headers, jsonBody);  local response = request.sendsync();  server.log(response.statuscode + ": " + response.body);  }  device.on("chtemp", log) ; |

AND change the “server.log” line in the device code with :

agent.send ("chtemp", message) ;

This POSTs the message to a URL on a server. You need to write the PHP code on the server to store the temperature values. You will also need to write a simple web page that allows you to view the temperature values on the screen.

**5. Putting this all together.**

You now have all 6 lines to the electric imp connected:

* Pin 1 – switch 1
* Pin 2 – switch 2
* Pin 5 – Red Led
* Pin 7 – Green Led
* Pin 8 – External Temperature
* Pin 9 – Internal Temperature

These form the basis of a home control system. The data for this needs to be stored in a database. I would store the data as a single JSON string. Devise a JSON string to store the data and put the first dummy state into your database.

Put together the device, and then agent code, to allow all 6 of the inputs or outputs as defined above. The examples above have all the code needed.

* When a switch (pin 1 or 2) is pressed, the agent needs to call a PHP script on your server to update the database.
* When you wish to switch a light (pin 5 or 7) press a button (link) on a web page, store the new state in the database and then call the agent to switch the light.
* Every 20 secs, take the temperature readings and deliver the new readings to the database.

You should have on your web page the current state of the “system”. You can have a refresh button to update the app screen.

Dr Geoffrey Lund.