

Institiúid Teicneolaíochta Cheatharlach



At the Heart of South Leinster

Utility Watch

Research Manual

Student Name: Ray Shannon
Student ID: C00079959
Student e-mail: c00079959@itcarlow.ie
Supervisor: Paul Barry

BSc (Honours) in Software Development (4th Year)
Institute Of Technology Carlow,
Kilkenny Road,
Carlow.
Date: 17/04/2015

Department of Computing and Networking,
Institute Of Technology Carlow

Abstract

The purpose of this document is to detail the completed work and effort carried out in researching each element of the project. This document will contain all researched material, some of which may not be used in the final project, but may have been researched for comparative purpose. The document format may not be in chronological order, rather grouping relevant and linked material with the aspect of the project being discussed.

Table of Contents

1. Initial Investigation.....	4
1.1. What similar products exist	4
2. Galileo Board.....	6
2.1. Board features	7
2.2. Initial connectivity.....	7
2.3. Supported Operating Systems	8
3. Network Configurations.....	8
3.1. Local configuration.....	8
3.2. Global Configuration	9
4. 3 rd Party Hardware	11
4.1. Network analysis for data packets.....	13
4.2. Network analysis for control packets	15
5. Required Software and Technologies	16
5.1. Python	16
5.2. Flask	16
5.3. Jinja2	16
5.4. JQuery	17
5.5. AJAX.....	17
5.6. SQLite	17
5.7. Matplotlib vs HighCharts.....	18
5.8. Responsive Web Design	19
5.9. Git.....	19
6. Appendix	19
Galileo Gen1 Win 7 Setup	20
Linux image on Galileo SD card setup and testing.....	23
6. References	28

1. Initial Investigation

1.1. What similar products exist

Once the final project submission was accepted, I started my research by looking at what similar products existed on the market. I wanted to see what the basic specification was being offered and also if any product stood out by offering something different which set them apart for other competitors. The following is a review of what I discovered and what I felt I would like to include and maybe improve in my project.

Company: Thermtec Energy Limited

Market: Domestic and Commercial

Domestic Products: OWL – Electricity, heating, water heating.

Commercial Products: Energy survey, Site analysis, audits, Billing Analysis, thermal imaging.

Link: www.theowl.com¹

Thermtec Energy Limited is an Irish company offering a wide range of energy monitoring solutions to both the Irish and UK markets. They provide services which cover 3 Phase monitoring, domestic electricity, home heating and hot water along with survey, audits and site analysis. They have products for both domestic and commercial markets as listed above.

My Verdict:

Taking aside and just comparing the similar aspects of my proposed product with the OWL domestic product, the basic offerings were quite similar. Both the OWL and Utility Watch products offer remote access to a dashboard displaying current data. The main difference between both products was that the owl did not offer historical data, so it was difficult to compare this week's consumption with previous weeks. Another difference was the OWL offered a cloud service and the package also contained a handheld display which the Utility Watch does not. It will be a standalone unit employing a web application to display all requested information.



Fig.1 OWL energy monitors

Company: efergy

Market: Domestic

Domestic Products: Engage Platform – Home automation, energy consumption, water time.

Link: www.efergy.com²

Efergy (efficient energy) is a global company with offices in the UK (headquarters), Hong Kong, China, Australia, United States, Canada and South Africa. Their flagship product is the wireless electricity monitor, and they have many add on products which complement it. They primarily focus on domestic markets.

My Verdict:

Both the efergy and Utility Watch share similar basic offerings like historical data, home automation and temperature control. One functionality, which I really like but would be unable to replicate, is the water time feature. This tells the shower user the amount of water and energy being used which may encourage people to be more efficient.



Fig. 2 efergy energy Monitors

2. Galileo Board

The Intel® Galileo board is based on the Intel® Quark SoC X1000, a 32-bit Intel Pentium®-class system on a chip (SoC). It is the first board based on Intel® architecture designed to be hardware and software pin-compatible with shields designed for the Arduino Uno R3. The Galileo board is also software-compatible with the Arduino Software Development Environment, which makes getting started a snap.³



Fig. 3 Intel Galileo Board

2.1. Board features

In order to design my product, I needed to know and understand the full capabilities of the Galileo Board. What features the board acquired in relation to

- initial connectivity
- supported operating systems
- programming language used
- network connectivity
- available communication ports
- processor specifications
- storage options

2.2. Initial connectivity

The Galileo board has an underlying linux OS but it is very light-weight. The Arduino IDE must be downloaded to enable communication between your laptop and Galileo. The board and the laptop are connected using the client's mini usb port and the laptop usb port. Drivers must be installed and updated on the initial connection.

The IDE has a library of test sketches which can be uploaded and executed. One of the initial sketches which you are encouraged to run is the blink test. This is a basic test which tests connectivity between the board and laptop and ensures correct operation of the board. The code used in the sketches is open c, and a sample blink sketch is shown below.

```
/*  
  Blink  
  Turns on an LED on for one second, then off for one second, repeatedly.  
*/  
// Pin 13 has an LED connected on most Arduino boards.
```

```
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);             // wait for a second
  digitalWrite(led, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);             // wait for a second
}^4
```

2.3.Supported Operating Systems

The next step needed to allow progress was to install an operating system. The operating system I choose was an open source Linux OS. As previously mentioned, the board has an underlying linux OS. An alternative to using the inherent OS was to install a larger linux image. This image had to be installed onto an SD card and inserted into the SD slot on the board. On start up, the processor always checks the SD slot first to see if any OS exists and runs it if it finds one.

Once I was satisfied the image had installed correctly I had access to python, Wifi drivers, SSH and many more useful packages.

The above steps were completed using an online tutorial from sparkfun.com⁵. It gave good clear and detailed instructions and the location for the required downloads. The steps involved will be documented and shown in the Appendix.

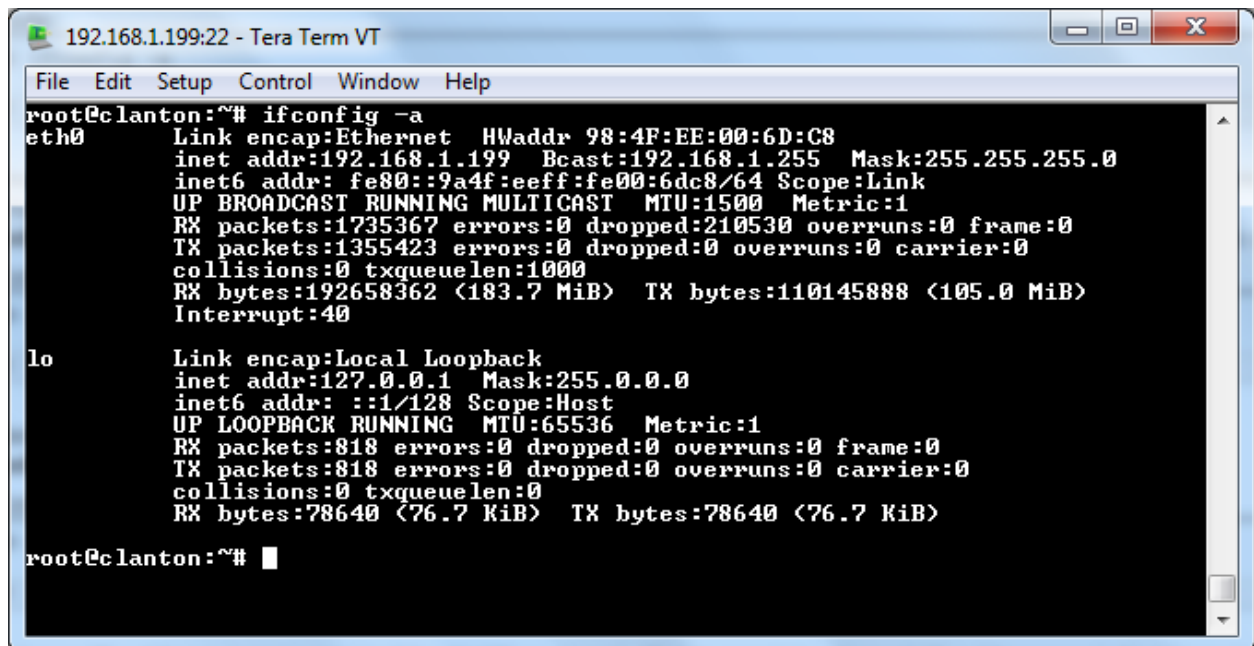
3. Network Configurations

3.1. Local configuration

One of the most important sections of the overall project was the ability to connect to the outside world. The user needs to be able to access the Galileo's web server from anywhere in the wide area network (WAN). The Galileo's network settings needed to be configured to enable connectivity on the local network. This would help improve the ease of connection, as it would eliminate direct connection to the board through the console port. Below is a sketch I used to configure the Galileo for my LAN.

```
void setup() {
  system("telnet -l /bin/sh");
  system("ifconfig eth0 192.168.1.199 netmask 255.255.255.0 up");
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```



```
192.168.1.199:22 - Tera Term VT
File Edit Setup Control Window Help
root@clanton:~# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 98:4F:EE:00:6D:C8
          inet addr:192.168.1.199  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::9a4f:eeff:fe00:6dc8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1735367  errors:0  dropped:210530  overruns:0  frame:0
          TX packets:1355423  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:1000
          RX bytes:192658362 (183.7 MiB)  TX bytes:110145888 (105.0 MiB)
          Interrupt:40

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:818  errors:0  dropped:0  overruns:0  frame:0
          TX packets:818  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:0
          RX bytes:78640 (76.7 KiB)  TX bytes:78640 (76.7 KiB)

root@clanton:~# █
```

Fig. 4 Local settings of Galileo Board

3.2. Global Configuration

Now that my board had connectivity on the LAN, it needed to be contactable from outside the LAN. Initially this was difficult as I needed to be outside the LAN to test, but inside the LAN to change the router settings. This led me to investigate important topics such as

- port forwarding

- dynamic addresses
- firewall access rules
- static NAT

A good guide to router configuration can be found at the portforwarding.com ⁶ website. It guides the user through the process once they select the make and model of the router. The guide also contains photos of the router pages making it very clear. My router guide can be found at this location http://portforward.com/english/routers/port_forwarding/TP-Link/TD-W8960N/defaultguide.htm

A **static NAT** rule was required which pointed the public ip address at the Galileo ip address. The application was going to run on port 5000, so any request to that port was forwarded to the Galileo server. The settings are displayed below in the router configuration page.

TP-LINK 300M Wireless N ADSL2+ Modem Router Model No. TD-W8960N

You have gone full screen. [Exit full screen \(F11\)](#)

NAT -- Virtual Servers Setup

Virtual Server allows you to direct incoming traffic from WAN side (identified by Protocol and External port) to the internal server with private IP address on the LAN side. The internal port is required only if the external port needs to be converted to a different port number used by the server on the LAN side. A maximum 32 entries can be configured.

Server Name	External Port Start	External Port End	Protocol	Internal Port Start	Internal Port End	Server IP Address	WAN Interface	Status	Enable/Disable	Edit	Remove
UtilityWatch	5000	5000	TCP/UDP	5000	5000	192.168.1.199	ppp0.2	Enabled	<input type="button" value="Disable"/>	<input type="button" value="Edit"/>	<input type="button" value="Remove"/>
Secure Shell Server (SSH)	22	22	TCP/UDP	22	22	192.168.1.199	ppp0.2	Enabled	<input type="button" value="Disable"/>	<input type="button" value="Edit"/>	<input type="button" value="Remove"/>
Telnet Server	23	23	TCP/UDP	23	23	192.168.1.199	ppp0.2	Enabled	<input type="button" value="Disable"/>	<input type="button" value="Edit"/>	<input type="button" value="Remove"/>
WordGame	10000	10000	TCP/UDP	10000	10000	192.168.1.199	ppp0.2	Created	<input type="button" value="Enable"/>	<input type="button" value="Edit"/>	<input type="button" value="Remove"/>
heating	5005	5005	TCP/UDP	5005	5005	192.168.1.199	ppp0.2	Created	<input type="button" value="Enable"/>	<input type="button" value="Edit"/>	<input type="button" value="Remove"/>
gateway	5050	5050	TCP/UDP	5050	5050	192.168.1.112	ppp0.2	Created	<input type="button" value="Enable"/>	<input type="button" value="Edit"/>	<input type="button" value="Remove"/>

Fig. 5 Static NAT for port 5000 to Galileo IP

Another area which required researching was **dynamic domain name server (DDNS)**. Most domestic public ip addresses are dynamic, and change from time to time at the discretion of the internet service provider (ISP). The public IP address remaining static was not guaranteed. There was 2 options; Firstly, pay for a static ip address or secondly, set up a free DDNS account and let it track any

changes to my public address. The router provided options for a free DDNS account and it listed the companies providing the service.

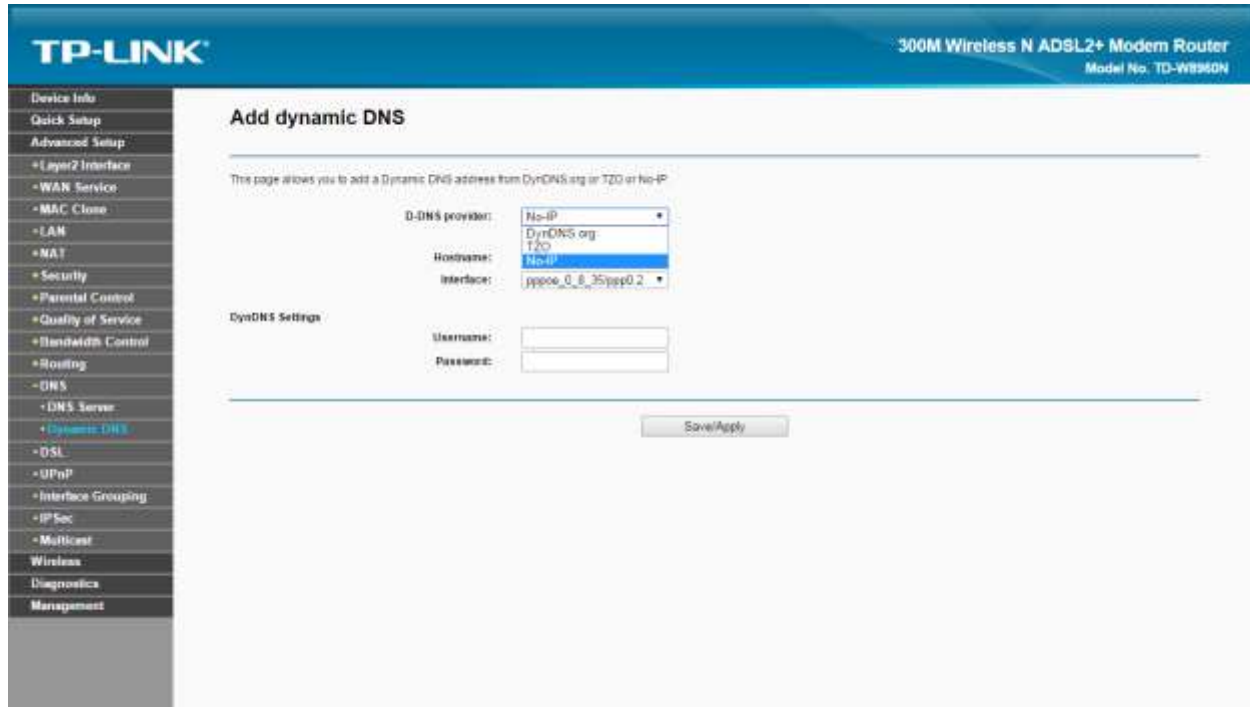


Fig. 6 dynamic DNS configuration page

Once the account was setup and domain name selected, the router gave confirmation of details.

Other rules were created on the router to allow ftp, telnet and winscp to access the board. These applications have specific ports assigned to them and it was a matter of repeating the process of pointing the router at the Galileo ip for these designated ports. Figure 2 displays the ports being open and enabled.

4. 3rd Party Hardware

In order for the Utility Watch to be able to interrogate the electricity meter, a 3rd party hardware would be required. This would consist of a transmitter, a clamp and remote relays. The clamp device would have to be clamped around the property mains supply and by some method transmit the data for

capture. All of the hardware I looked at were mainly wireless (RF) applications and reported the information back to a receiver.

There were many companies offering the full product to setup a monitoring system, but very few offered individual parts to allow a system to be designed. Sailwider sold hardware separately and customers could select which equipment they wanted.

This is the transmitter (Gateway) which communicates with the sensors and relays.

Energy Gateway



Fig. 7 Energy Transmitter

This is an RF switch which receives commands from the transmitter



Fig. 8 RF switch

This is an RF Socket which receives commands from the transmitter.



Fig. 8 RF Socket

4.1. Network analysis for data packets

Once I acquired the necessary hardware, I started to investigate the operations of these devices. I had the system installed in my home and begun to monitor the network activity being generated by the system. I used Fiddler⁷, a network analysis tool, for sniffing packets which were being sent between transmitter and receivers giving me the information I needed as shown in fig 9.

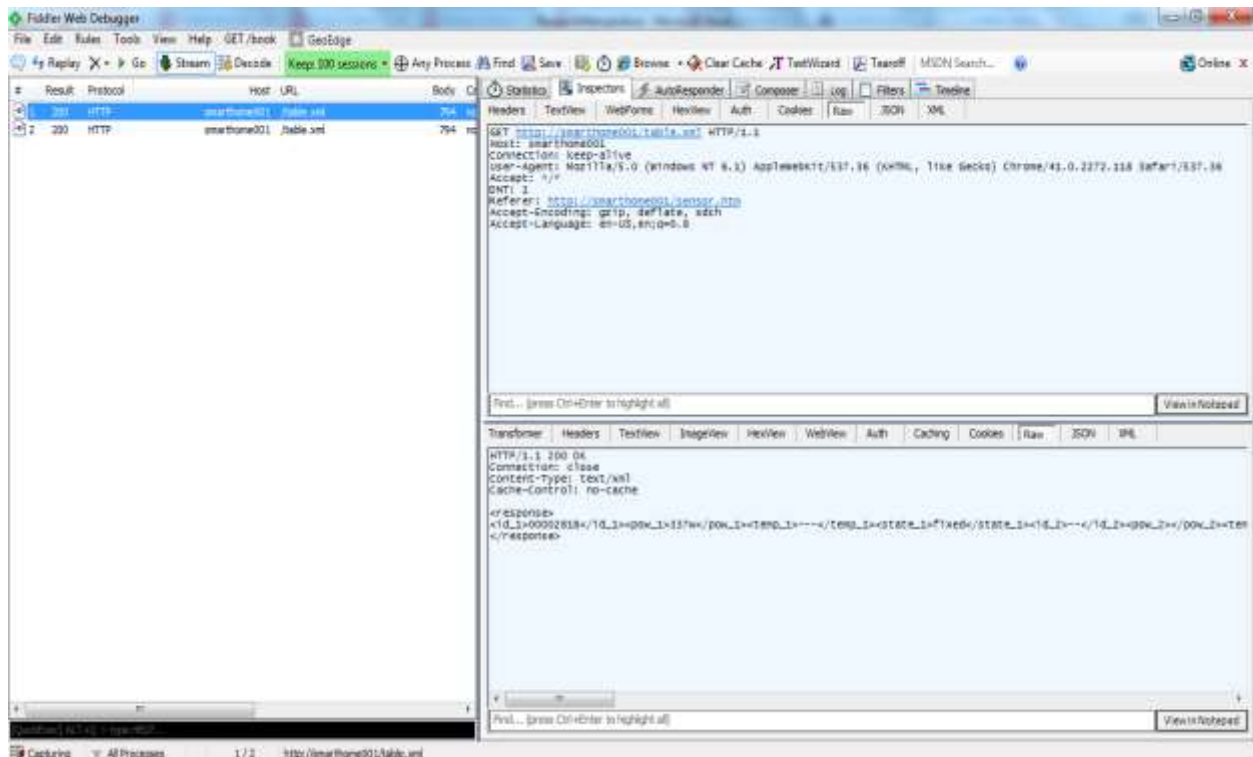


Fig. 9 Fiddler Network Analysis Tool

Shown below, is a test script I created to listen out for data packets being sent to the Galileo from the transmitter on port 5555. I captured that data and stored it in a local database to interpret what was in the packet.

```
import socket
import sqlite3
from time import gmtime, strftime

def Main():
    HOST='192.168.1.199'
    PORT = 5555

    s = socket.socket()
    s.bind((HOST,PORT))
    s.listen(1)

    print("listening for raw data.....")
    c,addr = s.accept()
    print("Connection from: " + str(addr))
    data = ""
    while True:
        data = c.recv(10000)
        connection = sqlite3.connect('utilitywatch.db')
        queryCursor = connection.cursor()
        sql = """ INSERT INTO RAWDATA(rawdata) VALUES(?) """
        queryCursor.execute(sql, (data,))
        connection.commit()

        timestamp = strftime("%H:%M:%S %d-%m-%Y ", gmtime())
        print(timestamp)
        print(len(data))

        if not data:
            connection.close()
            break

    c.close()
if __name__ == '__main__':
    Main()
```

The following is the response generated on execution of the test code.

```
<response>
<id_1>0000281B</id_1><pow_1>85w</pow_1><temp_1>---</temp_1><state_1>fixed</state_1>
<id_2>---</id_2><pow_2></pow_2><temp_2></temp_2><state_2>fixed</state_2>
<id_3>---</id_3><pow_3></pow_3><temp_3></temp_3><state_3>fixed</state_3>
<id_4>---</id_4><pow_4></pow_4><temp_4></temp_4><state_4>fixed</state_4>
<id_5>---</id_5><pow_5></pow_5><temp_5></temp_5><state_5>fixed</state_5>
<id_6>---</id_6><pow_6></pow_6><temp_6></temp_6><state_6>fixed</state_6>
<id_7>---</id_7><pow_7></pow_7><temp_7></temp_7><state_7>fixed</state_7>
<id_8>---</id_8><pow_8></pow_8><temp_8></temp_8><state_8>fixed</state_8>
<id_9>---</id_9><pow_9></pow_9><temp_9></temp_9><state_9>fixed</state_9>
<id_10>---</id_10><pow_10></pow_10><temp_10></temp_10><state_10>fixed</state_10>
</response>
```

The raw data detailed all the necessary information for the Utility Watch to generate informative graphs, tables and live detailed displays.

4.2. Network analysis for control packets

Further test code was required to replicate the control of the rf sensors and sockets. Again, the network analysis tool was employed to decipher the execution commands from the transmitter to the rf receiver devices. The following lines of code were created from what information was obtained during the network analysis process.

1. `conn = httplib.HTTPConnection('192.168.1.112')`
2. `conn.request("GET", "/setsocket.xml?num=4&set=on&ran=2717")`

Line 1 establishes the connection to the transmitter and line 2 sends the GET request telling the transmitter which channel to apply the 'turn on' request. With the data reading code and now the control code established, all that remained was to parse the response into a format that could be used.

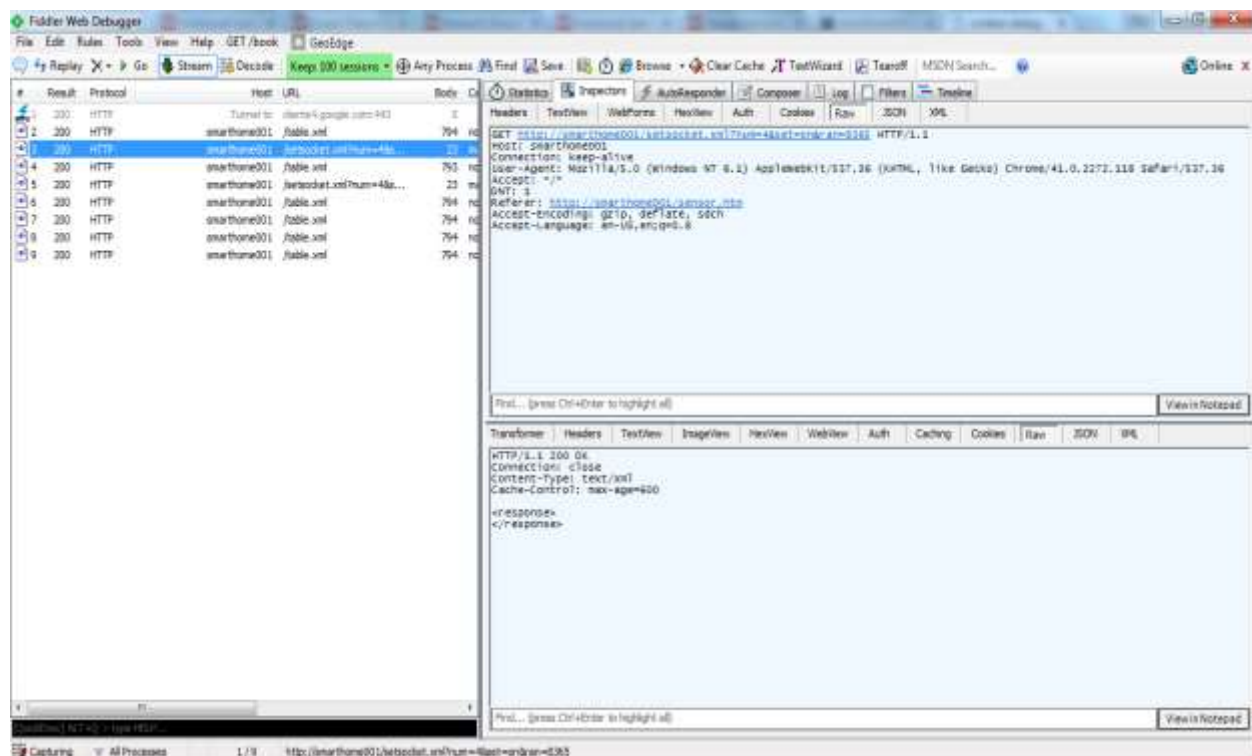


Fig. 10 Fiddler displaying the control request.

5. Required Software and Technologies

Various software packages and technologies were required to knit the whole project together. The majority of these I had not experienced before and required a lot of research and trial and error to obtain a level where I became proficient enough to complete the project.

5.1. Python

Python was the chosen development language for the backend program. I wanted to try a language I had not used before and it was part of the 4th year syllabus, so I could learn as I went along. Also, python was part of the linux image installed on the SD card obtained from Intel. It was version 2.7. I found the documentation⁸ for python really useful and easy to follow. It gave sample code to compliment the topic being discussed, which helped with explaining the topic.

5.2. Flask

Flask is a microframework for Python based on Werkzeug and Jinja 2⁹. It provides the web server functionality and a debugger for troubleshooting. Flask was executable in local and public mode. Changing the host address in `app.run ()` to `app.run (host= '0.0.0.0')` allowed other clients on the network to see the server.

The debug mode could also be turned on or off, by configuring the `app.run ()` command to `app.run (debug=True)`. This enables the Werkzeug application to give detailed feedback if the application crashes. Obviously, if the application is accessible to the public, this feature needs to be set to 'False'. Again, the documentation was simple and gave good coded examples.

5.3.Jinja2

Jinja2 is a Python library used to generate documents based on one or more predefined templates¹⁰. Any variables passed into the html code from the python application could be referenced and retrieved using this templating schema. Here is a sample of the code used in a javascript block in an html file.

```
<script>
    var chart_id = {{ chartID|safe }}
    var credits = {{ credits|safe }}
    var series = {{ series|safe }}
    var title = {{ title|safe }}
    var xAxis = {{ xAxis|safe }}
    var yAxis = {{ yAxis|safe }}
    var chart = {{ chart|safe }}
</script>
```

The code with the braces contains the variables passed in from the python program.

5.4.JQuery

jQuery is a javascript library that works across many web browsers. It was employed in the project for rendering the data charts, HTML document traversal and manipulation and event handling. It also assisted with AJAX request for updating live gauges. It has an extensive api¹¹ with a modern look and feel code example section. It was one of the best and most user friendly api's I used.

5.5.AJAX

AJAX is the art of exchanging data with a server, and updating parts of a web page - without reloading the whole page,¹² and that is exactly what it was used for. The Utility Watch requires both the temperature and live meter gauge to be updated as soon as any change occurs without refreshing the full page. It is an old technology but was ideal for providing the functionality I needed.

5.6. SQLite

As the project had a lot of data being generated and manipulated, and the linux OS being restricted, there was a need for a light-weight database system. SQLite comes preinstalled with most python packages, but unfortunately not with this skeleton linux image. There was access to precompiled binaries from the SQLite website¹³, which installed quite easily and without any issues.

SQLite was ideal for the Utility Watch application in every way except one. It had a limited write restriction of only one open connection to any database at any one time. There could be any amount of read connections, but only one write connection. This lead to multiple databases being created like tables would normally be. A database diagram will be available in the design document which will make it clearer. The test script for continuously writing to a database is shown below.

```
import sqlite3

def main():
    connection = sqlite3.connect('utilitywatch.db')
    queryCursor = connection.cursor()
    i=1
    while True:
        sql = ''' INSERT INTO test(VALUE) VALUES(?) '''
        queryCursor.execute(sql, (i,))
        connection.commit()
        i+= 1

if __name__ == '__main__':
    main()
```

Testing the SQLite involved running this program while having many reading programs access the same database. This was tested over a 5-hour period without any issues. As soon as another write program was started, the server crashed. Below is the code for the read program.

```

import sqlite3
from time import gmtime, strftime

def main():
    connection = sqlite3.connect('utilitywatch.db')
    queryCursor = connection.cursor()
    i=1
    timestamp1 = strftime("%H:%M:%S %d-%m-%Y ", gmtime())
    print("Start: ",timestamp1)
    while True:
        sql = "SELECT * FROM readingTbl"
        queryCursor.execute(sql)
        the_data = queryCursor.fetchall()
        connection.commit()
        i+= 1
        print((the_data[-1]))
    connection.close()
    timestamp2 = strftime("%H:%M:%S %d-%m-%Y ", gmtime())
    print("Start: ",timestamp1,"Finish: ",timestamp2)

if __name__ == '__main__':
    main()

```

5.7. Matplotlib vs HighCharts

When deciding on the best chart plotting library to use, I compared the python based matplotlib and JQuery based HighCharts. Matplotlib was very straight forward, with little complexity and offered a wide range of options with sample code. HighCharts was a bit more muddled with references to online API locations or javascripts and required a lot of dependencies to be added in order to function. It also had a large range of graphs to choose from and it looked very polished with added animation and user interaction.

I had chosen matplotlib initially, for its simplicity, and had sample code working locally. Unfortunately, upon deploying it to the Galileo, it wouldn't work as some of the dependencies were not compatible with the python version on the board. Even though both versions of python were 2.7, it was throwing compatibility errors.

HighCharts took a bit of work to get functioning. JavaScripts had to be copied into static folders and references made to these scripts in the html headers. This made the code look complex and messy. After many tutorials and trial and error, it eventually worked and looked impressive. Working with javascript took a little getting used to.

5.8. Responsive Web Design

With mobile technology being a major platform of how we access the internet, it is vital we include smaller resolution devices such as phones or tablets when creating web apps. This means certain fluidity must be applied when designing the web pages.

Twitter Bootstrap¹⁴ is a front end framework aimed at mobile technology. It contains defined stylesheets for html tags, tables and text. It also has javascript components in the form of JQuery adding more functionality relative to mobile platform. Like HighCharts, all the required CSS and javascript files need to be loaded into the header.

I have used bootstrap for some basic formatting but it added a lot of padded code around even the most basic html tags. It seemed a bit of overkill for what it was doing.

5.9. Git

Git¹⁵ is a repository hosting service, a secure and efficient way of controlling source code and revision control. I found git very useful and with a few basic commands quickly became proficient with it. It involved setting up a local repository and initialising it. All files added locally could then be pushed up to the remote repository using add, commit and push commands. It also offered a rollback option if a previous version of any file was required. Any project created nowadays would have some form of source control, and git would be the most popular.

6. Appendix

These steps have been obtained from <https://learn.sparkfun.com/tutorials/galileo-getting-started-guide> as stated in the references.

Galileo Gen1 Win 7 Setup

Preparation

1. Download Arduino software and Linux Galileo SD image from <https://communities.intel.com/docs/DOC-22226> and Unzip to folder of choice.

Software Downloads

created by mmmccoy on Jan 7, 2014 4:29 PM, last modified by intel_kud on Sep 29, 2014 2:46 PM

Software package release version 1.0.3.
See the [Getting Started Guide](#) for step-by-step information on installing the software.
Important: software downloads are also found in the [release notes](#)

Download the zip file for your operating system (OS). Each OS zip file includes the latest firmware so you can use the IDE to update your board.

Link	Software	Operating System	Intel Board	File Size	File Type
Download	Arduino Software 1.5.3	Linux 32 Bit	Galileo	72.2MB	zip
Download	Arduino Software 1.5.3	Linux 64 Bit	Galileo	73.5MB	zip
Download	Arduino Software 1.5.3	Mac OS X	Galileo	54.7MB	zip
Download	Arduino Software 1.5.3	Windows	Galileo	104MB	zip

Additional downloads

Link	Software	Intel Board	File Size	File Type	Notes
Download	SD-Card Linux Image	Galileo	50MB	zip	Enable WiFi
Download	Board Support Package (Yocto archive 1 of 2)	Quark	1.5GB	zip	
Download	Board Support Package (Yocto archive 2 of 2)	Quark	1.2GB	zip	
Download	Intel customized Arduino IDE 1.5.3 (source code)	Galileo	223MB	zip	The source code to the Intel customized version of Arduino IDE 1.5.3
Download	ESP Patches and Build Instructions	Galileo	5KB	zip	Instructions on how to build the Intel® Galileo fast time image from the Quark® BSP release 1.0.3, including Intel® Galileo specific patches.
Download	Flash Missing PDAT Release	Galileo	8MB	zip	This image is for in-System-Programming of the Intel® Galileo SPI flash, using a tool such as a OedProg SP-100. The file must be prepared with the SPI flash-tool in the Quark™ BSP with the platform ID of the Intel® Galileo or Intel® Galileo Gen2 and the MAC address from the sticker on your board.
Download	Flash Missing PDAT Release	Galileo	7MB	zip	This image is already embedded in the IDE therefore it is only necessary if you need to flash SPI directly via serial connection.
Download	Cross Compiler Toolchain	Galileo	240MB	zip	Cross-Compiler Toolchain.

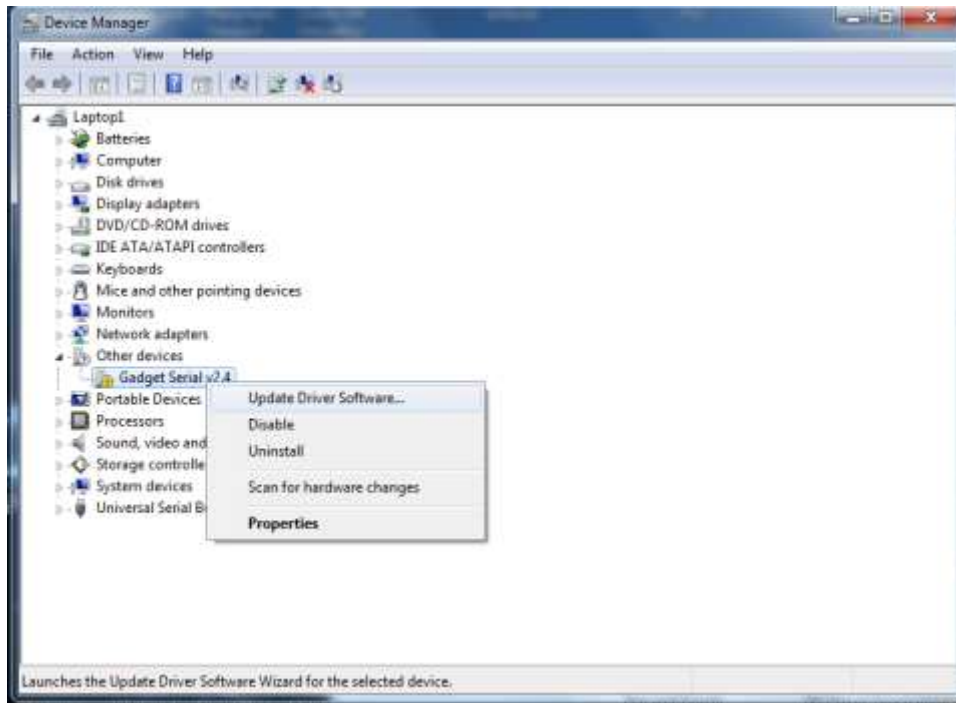
2. Format your SD Card by using windows 7 Quick Format.



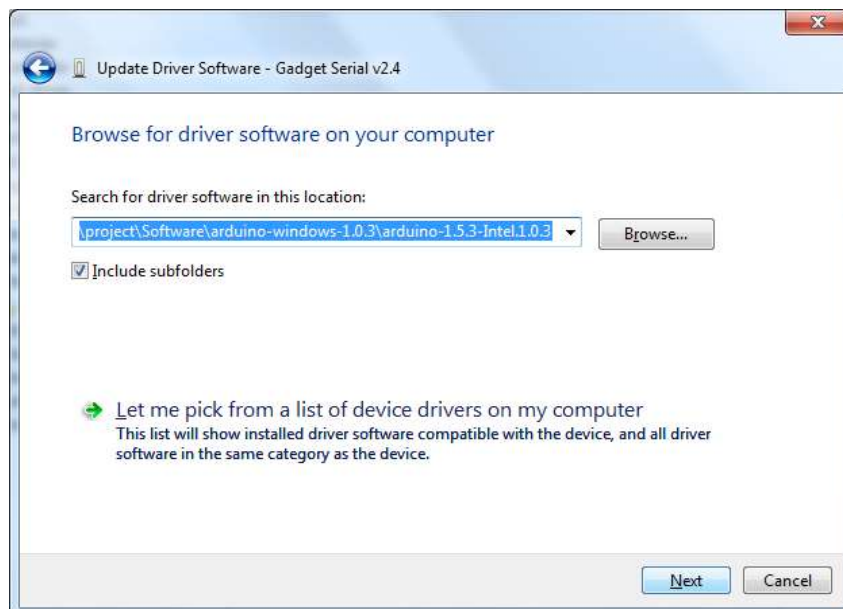
3. Download latest version of Tera Term. A terminal emulator to view Galileo's SD card.

Galileo initial setup and testing

1. Connect power supply to Galileo board.
2. Connect USB on computer to micro USB client on Galileo board.
3. Open Windows Device Manager (Win+Pause -> Device Manager)
4. Expand *Other devices* find Gadget Serial v2.4 and right click, then left click *Update Driver Software*

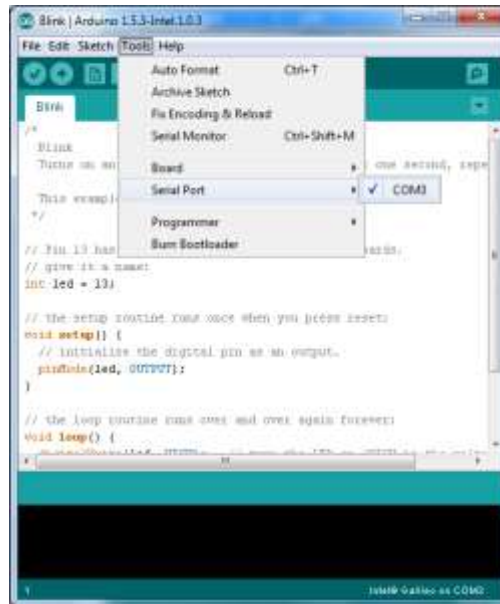
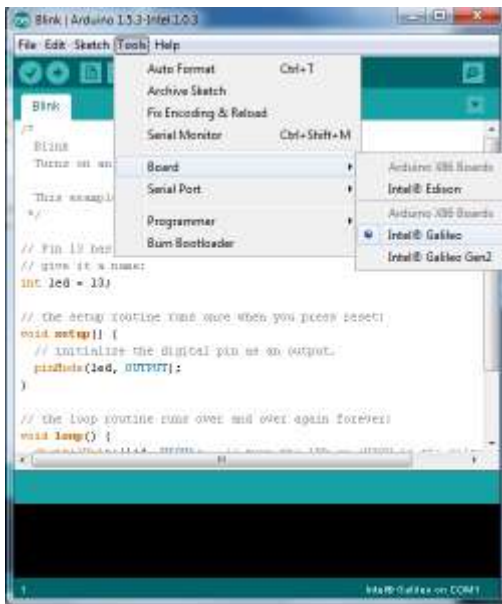


5. Then select *Browse my Computer for driver software*
6. Change Location to folder you unzipped Arduino and press Next.



7. Start andruino.exe.
8. On the Arduino menu bar go to Tools > Board and select *Intel Galileo*.

- On the Arduino menu bar go to Tools > Serial Port and select Com 3. The Com port that was setup during driver Installation.



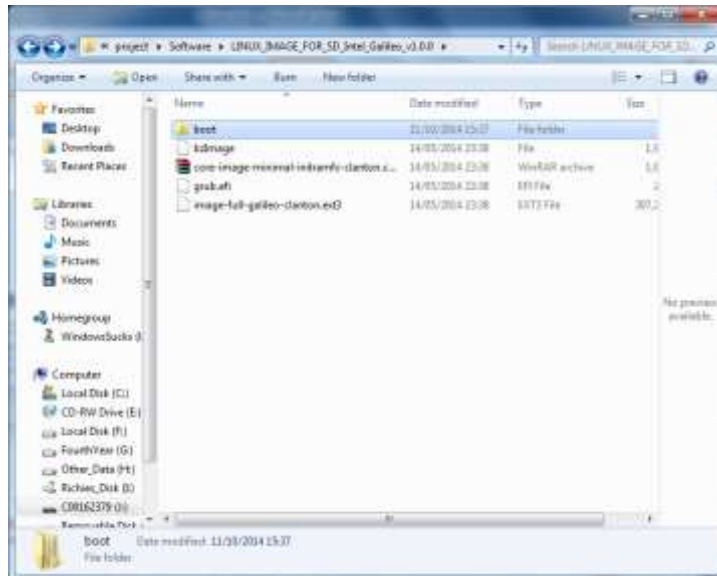
- On the Arduino menu bar go to File > Examples > Basics and select *Blink*. This program will make a led blink on your Galileo board.
- Select Upload. This will compile the program (which may take up to 30secs) and then run on the Galileo board. Communications are now set up.



Linux image on Galileo SD card setup and testing

Connection will be done using a R424 Ethernet cable

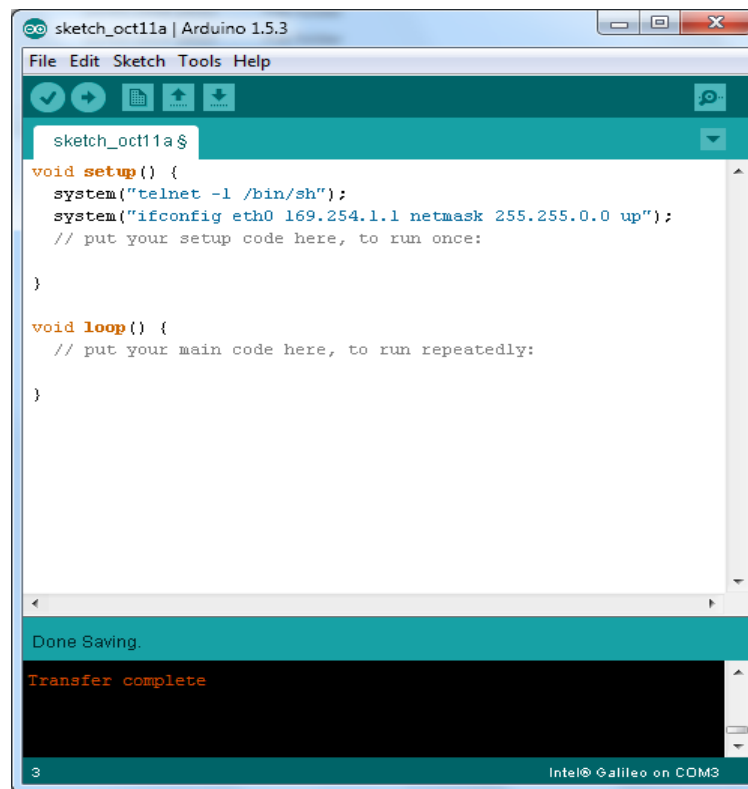
1. Copy downloaded image on to the formatted SD card. This is all that should be on the card.



2. Open the grub.conf file in boot > grub > *grub.conf* in your text editor. This will need to be altered in order to avoid conflict at boot time by only leaving in references to the SD card. See changes below.

```
1: default 1
2: timeout 1
3:
4: color white/blue white/cyan
5:
6: title clantox SVP kernel-SPI initrd-SPI IPR-On IO-APIC/HPET NoMMU
7: kernel --spi root=/dev/ram0 console=ttyS1,115200n8 earlycon=uart8250,mmio32,
8: $EARLY_CON_ADDR_REPLACE,115200n8 ymalloc=384M reboot=efi,war= apic=debug co
9: initrd --spi
10:
11: title clantox SVP kernel-NoSSStorage initrd-NoSSStorage image-full IPR-On IO-APIC/HPET NoMMU debug
12: root (hd0,0)
13: kernel /bzImage root=/dev/ram0 console=ttyS1,115200n8 earlycon=uart8250,mmio32,
14: $EARLY_CON_ADDR_REPLACE,115200n8 ymalloc=384M reboot=efi,war= apic=debug co
15: LABEL=boot debugshell=5 root=image-image-full-galileo-clantox-ext3
16: initrd /core-image-minimal-initramfs-clantox-core.gz
17:
```

3. Disconnect the power supply from the Galileo board and insert SD card.
4. Reconnect power supply to the Galileo board.
5. Connect Ethernet cable from Galileo board to PC.
6. Open cmd.exe on Windows and type *ipconfig*. You can then check that the default IP address of 169.254.*.* with a subnet mask of 255.255.0.0 is issued.
7. Next the IP Address on the Ethernet port on the Galileo board must be statically set. Create a new file with the Arduino editor and write in the program as below. Then upload. Remember the Arduino editor is still communicating through the USB on COM 3.
8. Open Tera Term and on the menu bar go to File > New connection and copy the settings below and click OK.



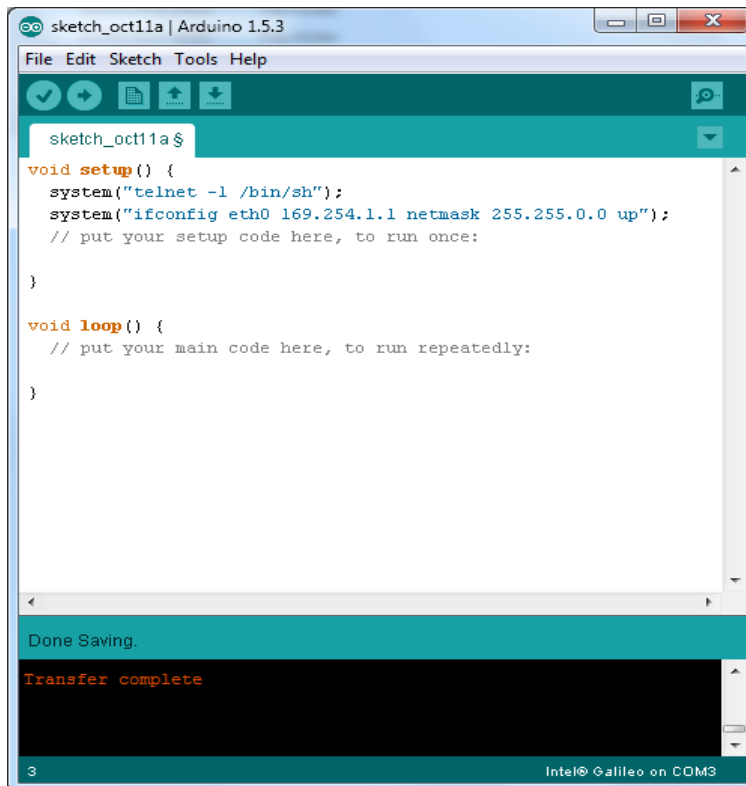
The screenshot shows the Arduino IDE interface for a sketch named 'sketch_oct11a'. The code in the main editor is as follows:

```
sketch_oct11a $
void setup() {
  system("telnet -l /bin/sh");
  system("ifconfig eth0 169.254.1.1 netmask 255.255.0.0 up");
  // put your setup code here, to run once:
}

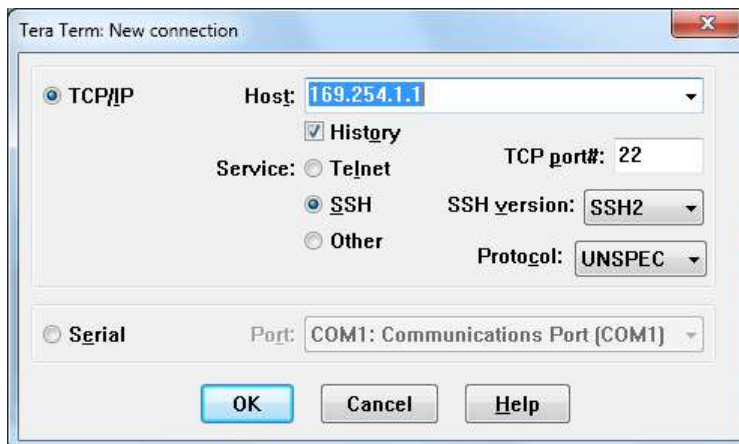
void loop() {
  // put your main code here, to run repeatedly:
}
```

Below the code editor, a status bar indicates 'Done Saving.' and 'Transfer complete'. At the bottom of the window, it shows 'Intel Galileo on COM3'.

- Open Tera Term and on the menu bar go to File > New connection and copy the settings below and click OK.



```
sketch_oct11a | Arduino 1.5.3
File Edit Sketch Tools Help
sketch_oct11a $
void setup() {
  system("telnet -l /bin/sh");
  system("ifconfig eth0 169.254.1.1 netmask 255.255.0.0 up");
  // put your setup code here, to run once:
}
void loop() {
  // put your main code here, to run repeatedly:
}
Done Saving.
Transfer complete
Intel Galileo on COM3
```



Tera Term: New connection

TCP/IP Host: 169.254.1.1

History

Service: Telnet TCP port#: 22

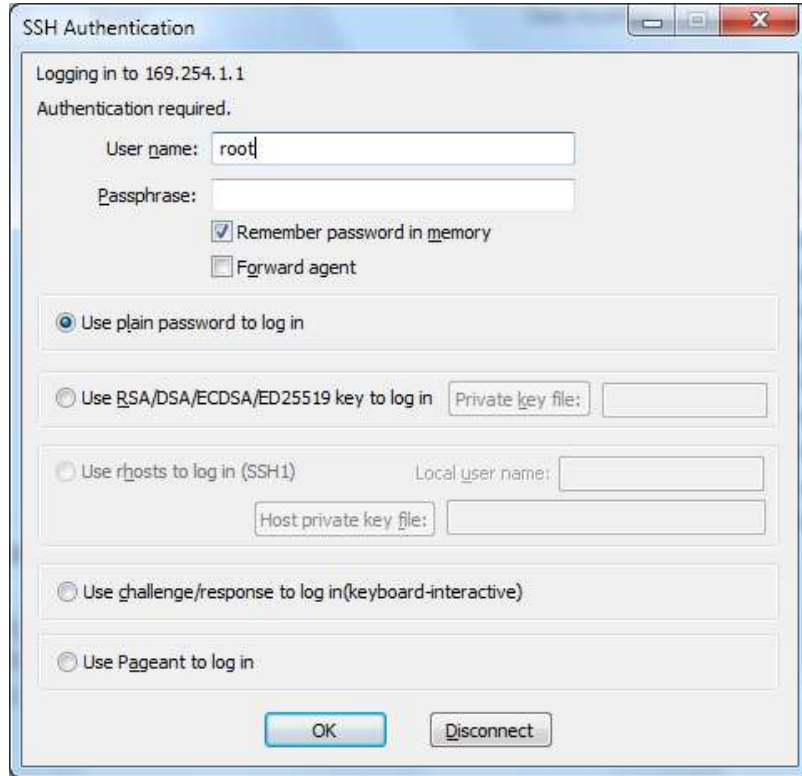
SSH SSH version: SSH2

Other Protocol: UNSPEC

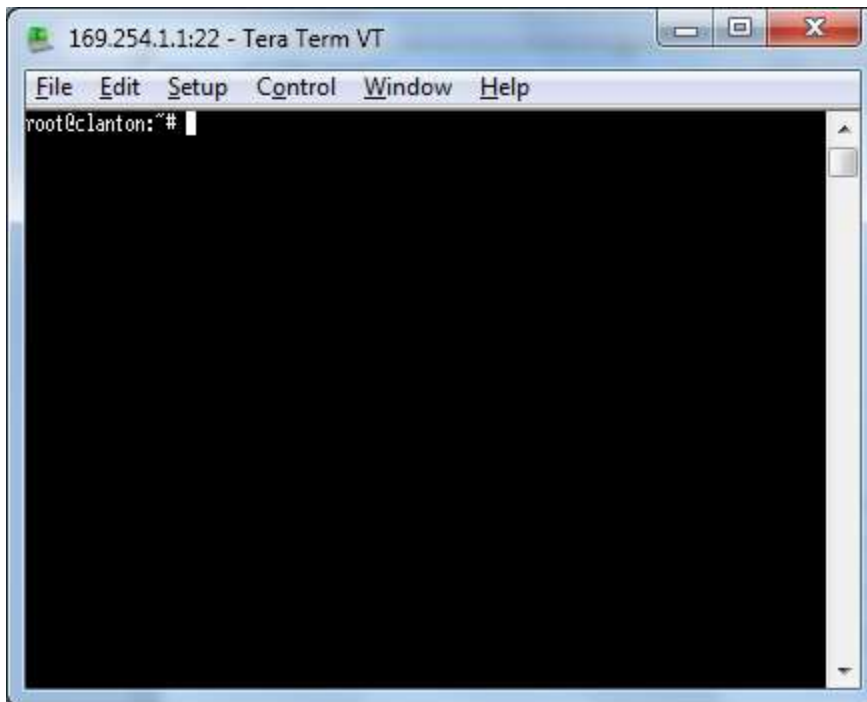
Serial Port: COM1: Communications Port (COM1)

OK Cancel Help

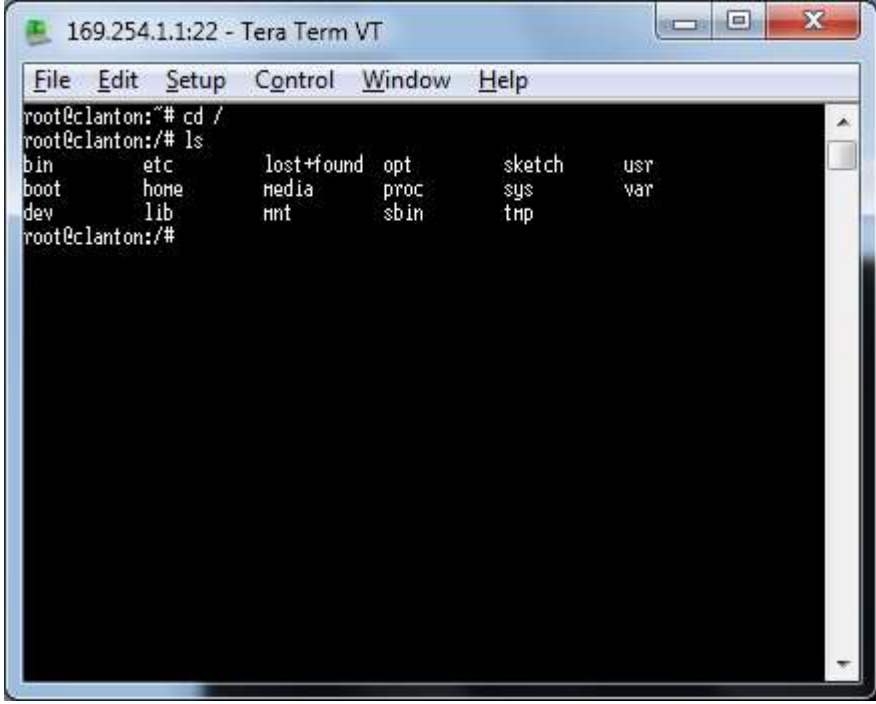
- A SSH Authentication window should open as below. Type in *root* as Username



11. A Tera Term Console will open in the user home directory.

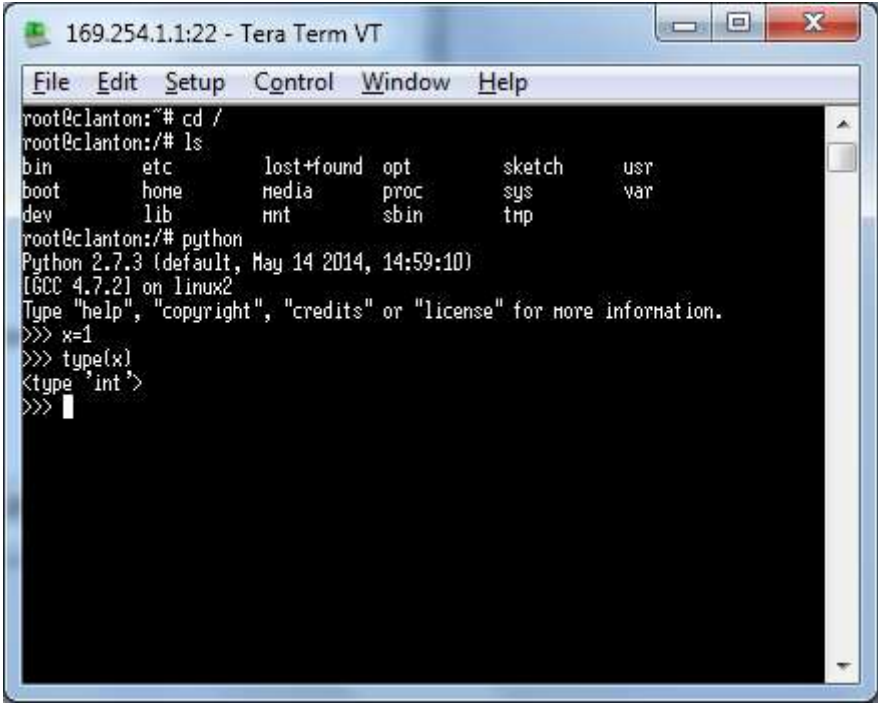


12. Change to the root directory and get a listing of all the folders



```
169.254.1.1:22 - Tera Term VT
File Edit Setup Control Window Help
root@clanton:~# cd /
root@clanton:/# ls
bin      etc      lost+found  opt      sketch    usr
boot     home     media       proc     sys       var
dev      lib      mnt        sbin     tmp
root@clanton:/#
```

13. Check Python version and run some commands on the Python Shell.



```
169.254.1.1:22 - Tera Term VT
File Edit Setup Control Window Help
root@clanton:~# cd /
root@clanton:/# ls
bin      etc      lost+found  opt      sketch    usr
boot     home     media       proc     sys       var
dev      lib      mnt        sbin     tmp
root@clanton:/# python
Python 2.7.3 (default, May 14 2014, 14:59:10)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> x=1
>>> type(x)
<type 'int'>
>>> |
```

6. References

- ¹ Energy Monitoring (2014) - retrieved from theowl.com dated 23/10/2014
- ² e2 Classic. (2014) -retrieved from <http://efergy.com/uk/e2-classic-2-0> dated 23/10/2014
- ³ Product Brief (13th December 2013) - retrieved from http://download.intel.com/support/galileo/sb/galileoprodbrief_329680_003.pdf dated 21/10/2014
- ⁴ *Examples - Basics (2014) - retrieved from* <http://arduino.cc/en/Tutorial/Blink?from=Tutorial.BlinkingLED> dated 23/10/2014
- ⁵ Galileo Getting Started Guide - retrieved from <https://learn.sparkfun.com/tutorials/galileo-getting-started-guide> dated 21/10/2014
- ⁶ How to Port Forward your router (2014) – retrieved from http://portforward.com/english/routers/port_forwarding/ dated 2/11/2014
- ⁷ The free web debugging proxy for any browser, system or platform (2002-2014) – Retrieved from <http://www.telerik.com/fiddler> dated 2/11/2014
- ⁸ What’s new in Python 2.7 (1990-2014) – retrieved from <https://docs.python.org/2/whatsnew/2.7.html> dated 14/10/2014
- ⁹ Flask is Fun (2014-Armin Ronacher) – retrieved from <http://flask.pocoo.org/> dated 15/10/2014
- ¹⁰ A quick start guide to using Jinja2 Template Engine (2010-2012 Bryan Hill) – retrieved from <http://kagerato.net/articles/software/libraries/jinja-quickstart.html> dated 23/10/2014
- ¹¹ JQuery API – retrieved from <http://api.jquery.com/> dated 5/02/2015
- ¹² AJAX Tutorial – retrieved from <http://www.w3schools.com/ajax/> dated 30/03/2014
- ¹³ SQLite Download Page – retrieved from <http://www.sqlite.org/download.html> dated 03/11/2014
- ¹⁴ An overview of Bootstrap, how to download and use, basic templates and examples, and more – retrieved from <http://getbootstrap.com/getting-started/> dated 28/12/2014
- ¹⁵ Downloading Git – retrieved from <http://git-scm.com/download/win> 27/11/2014