

Bluetooth Low Energy
for Use with MEM Sensors

by

Clinton Francis Hughes

A Thesis Presented in Partial Fulfillment
Of the Requirements for the Degree
Master of Science

Approved November 2015 by the
Graduate Supervisory Committee:

Jennifer Blain Christen, Chair
Sule Ozev
Umit Ogras
James Aberle

ARIZONA STATE UNIVERSITY

December 2015

ABSTRACT

Designers creating the next generation remote sensing enabled smart devices need to overcome the challenges of prevailing ventures including time to market and expense.

To reduce the time and effort involved in initial prototyping, a good reference design is often desired and warranted. This paper provides the necessary reference materials for Designers to implement a wireless solution efficiently and effectively.

This document is intended for users with limited Bluetooth technology experience.

Many sensing-enabled devices require a 'hard-wire' or cable link to a host monitoring system. This can limit the potential for product advancements by anchoring the system to a single location preventing portability and the convenience of a remote system. By removing the "wired" or cabled portion from a design, a broader scope of devices becomes feasible.

One common problematic area for these types of sensors is within the internal medicine field. Proximity sensing is far more practical and less invasive to implement than surgical implantation. Bluetooth Low Energy (BLE) systems solve the hard wired problem by decoupling the physical sensor from the host system through a BLE transceiver that can send information to an external monitoring system. This wireless link enables new sensor technology to be leveraged into previously unobtainable markets; such as, internal medicine, wearable devices, and Infotainment to name a few. Wireless technology for sensor systems are a potentially disruptive technology changing the way environmental monitoring is implemented and considered.

With this BLE design reference, products can be created with new capabilities to advance current technologies for military, commercial, industrial and medical sectors in rapid succession.

ACKNOWLEDGMENTS

1. Special thanks to Joe Decuir (Senior Standards Architect at CSR) for some of the Graphics and table used in this report.
2. Thanks to the Bluetooth SIG consortium for their help and suggestions and also to their Developer Portal; who provided a wealth of information.
3. Thanks to the Bluetooth.org Developer Portal Community Forum for suggestions on App and Device Development
4. Thanks to Simon Finch for outing together an Environmental Sensor board Demonstration
5. Special to all thanks to the Committee Members

TABLE OF CONTENTS

	Page
LIST OF TABLES	v
LIST OF FIGURES	vi
CHAPTER	
1. INTRODUCTION.....	1
Purpose	1
Objective.....	2
2. WIRELESS SOLUTION.....	3
What is Bluetooth?	7
Bluetooth Low Energy (BLE)	9
3. DEVELOPING WITH BLUETOOTH LOW ENERGY.....	11
Bluetooth Low Energy Radio – The Physical Layer.....	12
4. PROJECT SUMMATION	13
Hardware Level	16
Software Level.....	16
Code Development and Deployment	21
5. HARDWARE	21
CSR μ Energy CSR1010 Module board.....	22
CSR1010 CSR μ Energy Starter Development Kit	23
Environmental Sensor Board H13229.....	23
6. FIRMWARE DEVELOPMENT ENVIRONMENT	25
Launch the xIDE	26
Open the Environment Sensor Project.....	27
Application Files	28
Customizing the Embedded Application	33
Compiling and Deploying the Application.....	42

CHAPTER	Page
7. SYSTEM ARCHITECTURE / PROJECT DESCRIPTION	51
Environment Sensor Application Overview	51
Firmware Code Overview.....	55
Internal State Machine	59
Service Characteristics Database	62
8. ANDROID DEVELOPMENT ENVIRONMENT	66
Setting Up the EnvironmentApp Project.....	67
Launch Android Studio IDE	73
Application Files	76
Compiling and Deploying the Application.....	79
9. PHONE APP OVERVIEW.....	83
Results of Prototype Implementation and Demo	89
Environmental Sensor Board demonstration	89
List of Code Debugs and Fixes	91
Issues seen during Design / Roadblock & Workaround.....	92
10. SUMMARY ABD FUTURE WORKS	94
11. DOCUMENTATION AND OTHER COLLATERAL	95
Documentation	95
Hardware and Development Kit	95
REFERENCES.....	96
APPENDIX	
A CSR1010 MODULE SCHEMATIC.....	98
B ENVIRONMENT MEMS SENSOR BOARD SCHEMATIC	100

LIST OF TABLES

TABLE	Page
2.1: Short range Wireless Application Areas	4
2.2: Short range Wireless Application Specs	4
6.2: Source Files	30
6.3: Header Files	32
6.4: Database Files	33
6.5: Advertising Parameters	37
6.6: Advertising Parameters	37
7.1: Environmental Sensing Profile Role	53
7.2: Application Topology	53
7.3: Responsibilities	53
7.4.1: Battery Service Database	53
7.4.2: Device Information Service Database	63
7.4.3: Environmental Sensing Service Database	64
7.4.4: GAP Service Database	65
7.4.5: GATT Service Database	65
7.4.6: CSR Custom Characteristics	53
8.3.1: App Java Files	76
8.3.2: App XML Files	77
8.3.3: BTSmart Library Files	78

LIST OF FIGURES

FIGURE	Page
3.1: Bluetooth Low Energy – Frequency Channels	142
4.1: Simplified Overview	14
4.2: Project Hierachy	15
5.1: CSR1010 Module	22
5.2: CSR μ Energy.....	23
5.3: Environment Sensor Board	23
5.4: Environment Sensor Board Attached to Starter Kit Board	22
6.1: xIDE Platform after Launch	23
6.2: Open Workspace	27
6.3: Project File Structure	22
6.4.1: Configuration Store Key File.....	35
6.4.2: Configuring the Device Name	59
6.4.4: Connection Parameter Update Procedure	23
6.5.1: Content of the CSR uEnergy Development Kit	42
6.5.2: Development Kit with USB cable.....	43
6.5.3: Pop-up Window Verifying Driver Installation	43
6.5.4: xIDE Build Process.....	44
6.5.5: Selecting Build Active Project.....	45
6.5.6: Build Output Window	45
6.5.7: Selecting 'Run' to Begin Deployment	46
6.5.8: Project Successfully Deployed to the Hardware	47
6.5.9: Stop Debugging.....	48
6.5.10: Programmer Board Arranged for Sensor Board.....	23
6.5.11: Programmer Board Compoent Configurations	49
6.5.12: Final Board Configuration.....	50
7.1: Environmental Sensing Profile	52

LIST OF FIGURES

FIGURE	Page
7.2: Primary Services	54
7.3: Environment Sensor (state transitions) Diagram.....	59
8.1.1.1: Android Setup Wizard.....	67
8.1.1.2: Downloading Components	68
8.1.2.3: File Edit.....	70
8.1.2.4: Importing Project	70
8.1.3.1: USB Driver for Windows	72
8.1.3.2: Configuring Phone for Android Development Environment	72
8.2.1: Importing Project	73
8.2.2: Building Project.....	73
8.2.3: Install Missing Components.....	74
8.4.1: Building the 'App'	79
8.4.2: Running the 'App'	80
8.4.3: Choose Device Pop-up Window.....	81
8.4.4: Session 'app' running	81
8.4.5: Running App Verified.....	82
9.1: Basic Flowchart	83
9.2: Activity Connection	84
9.3: Activity Main.....	85
9.4: Activity Information	86
9.5: Activity Battery	87
9.6: Activity Environment	88

DEFINITIONS

Abbreviation	Definition
ADK	Application Development Kit
ADT	Android Developer Tool. Is a plugin for Eclipse that provides GUI access to the command line SDK Tools and UI Design tools allow for rapid prototyping
APP	Application
ATT	Attribute Protocol
BLE®	Bluetooth Low Energy (now known as Bluetooth Smart): a Bluetooth technology designed for ultra-low power consumption
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
BluetoothSIG	Bluetooth Special Interest Group
Bluetooth Smart	Also known as Bluetooth Low Energy
Characteristic	A characteristic is a data value transferred between the client and the server
Client	The <i>client</i> is the device that initiates GATT commands and accepts responses. For this project, the Android device will act as the client as this is a typical use case. However, the Android BTLE API does allow the Android device to act as the server.
COTS	Commercial Off the Shelf
CS	Configuration Store
CSB	Chip Select Bar
CSR	Cambridge Silicon Radio
Demo	Demonstration
Descriptor	A descriptor provides additional information about a characteristic
DIV	Diversifier
e.g.	<i>exempli gratia</i> , for example
EEPROM	Electrically Erasable Programmable Read Only Memory
Etc	<i>et cetera</i> , and the rest, and so forth
Firmware	Is software that's installed on a small memory chip on a hardware device
GAP	Generic Access Profile
GATT Profile	All BTLE devices implement one or more profiles. A profile is a high level definition that describes how services can be used to enable an application. Low energy application profiles are based on the Generic Attribute Profile (GATT). This is a general

Abbreviation	Definition
	specification for sending and receiving short pieces of data (known as attributes) over a low energy link.
GND	Ground
IDE	Integrated Development Environment
i.e.	<i>Id est</i> , that is
I ² C	Inter-Integrated Circuit
IRK	Identity Resolving Key
ISM	Industrial Scientific Medical
L2CAP	Logical Link Control and Adaptation Protocol
LED	Light Emitting Diode
LM	Link Manager
MEMs	'microelectromechanical systems' are sub-millimetre devices able to sense mechanical information in their surroundings
NVM	Non Volatile Memory
PC	Personal Computer
PIO	Programmable Input Output
PnP	Plug and Play
PTS	Profile Testing Suite
Server	The server is the device that receives GATT commands or requests and returns responses
Service	A <i>service</i> is a group of characteristics that operate together to perform a specific function. Many devices implement the Device Information Service. This service is made up of characteristics such as manufacturer name, model number, serial number, and firmware revision.
SIG	Special Interest Group
SMP	Security Manager Protocol
SPI	Serial Peripheral Interface
Tx	Transmit
UUID	Universally Unique Identifier

1. Introduction

In today's world, there is an ever increasing need to communicate and / or transmit information from our surrounding environment through sensors. These sensors are key components in a vast array of consumer electronic devices, such as mobile handsets, tablets, gaming consoles and wearable technologies that provide the crucial input for this data. Control, connectivity, and information make it possible to create connected devices known as the 'Internet of Everything' (IoE). This allows us to make our lives easier, more convenient and to control / interact with everyday technology to feed our need for instant gratification.

Over the next few years the market for Micro Electro-Mechanical Systems (MEMS) sensors, in connected wearables, is set to increase dramatically. Of these, the fastest growing segment are activity monitors that require sensors, such as accelerometers, angular rate and pressure sensors to measure physical activity. Also included are the bioelectronics for monitoring sensors on and/or in the human body, like pacemakers, heart rate monitors, blood pressure meters, PH and blood glucose meters, thermometers and other hardware. Smart watches are another area that has seen considerable interest, particularly with the recent launch from Apple and Motorola (Google) 360.

1.1. Purpose

With the growing desire to transmit data from sensors, using wireless communication for human wearable devices, development time and cost become a limiting factor for rapid prototyping. Typically, these projects get wired directly from sensors to test equipment / monitoring devices limiting the developmental scope and design capability of the project. The goal was to devise a reference platform that could be readily utilized by others to minimize cost and development time without the use of wires.

It is also of interest to parallel this project to facilitate the needs of future academia, as a reference for those wanting to apply wireless sensor communications to their own projects. Students could reference this wireless solution as an alternative to their hardwired designs; such as, to transmit bioelectronics sensor data.

1.2. Objective

The design approach was to create a project based on something that is done at Cambridge Silicon Radio (CSR, employer). CSR is widely known and respected for world class Bluetooth technology and therefore became the most logical option to implement into this project.

Design guidelines:

- The project should use a low power wireless solution to transmit MEM and other sensor data to a remote monitoring/recording device such as a Personal Computer (PC) or Smartphone. The Bluetooth Low Energy (BLE) design has the lowest power consumption on the market for wireless communication. This allows for the longest battery life possible using coin-cell battery to allow for human wearable device to be lightweight and portable.
- To minimize cost and development time, the prototype used commercial off the shelf (COTS) hardware. For this particular design, the CSR μ Energy single-mode BLE solution (CSR1010 Reference module) was chosen along with the associated CSR μ Energy Development Kit. See Hardware [Section 5](#) for further information.

Specifications:

- The CSR μ Energy Development Kit allows for quick software (firmware) development, verification of functionality and to program the Flash local to the CSR1010 Module. Here are the boards designed around the CSR μ Energy Development Kit Interfaces:
 - CSR1010 Module: is a small Printed Circuit Board (PCB) that has all the components to operate remotely with a battery. It contains the Flash, Clocks, Antenna, etc. See [Appendix A](#) for Schematic and Image of PCB.
 - The μ Energy Environmental Sensor board, designed specifically for this project, is an add-on board for CSR's μ Energy Development kits that provides Gyroscopic, Magnetic, Temperature, Pressure, Humidity and Acceleration MEMS sensors [1]. Designed to be compatible with either the CSR μ Energy Starter Kit or the CSR μ Energy Development Kit, the board is attached via a set of header

sockets on the board [2]. Once physically connected, power is provided from the development board, which can be fitted with either a coin-cell battery or USB power cable. See [Appendix B](#) for Schematic and Image of PCB.

- Firmware: The CSR's Integrated Development Environment (xIDE), supplied with CSR μ Energy Software Development Kits (SDKs), and is the development platform. The API software was written to program flash to configure CSR1010 Bluetooth Transceiver PIO to communicate to the MEM Sensors through the I2C/SPI digital interfaces. CSR1010 Module can free run with battery. See Firmware Development / Code [Section 7.2](#) for further information [2].
- Android SDK was chosen as the APP Development Platform since Samsung Galaxy S4 (Android 19) was already utilized on other projects. See Software [Section 9.1](#) for further information.

2. Wireless Solution

In this section, we will look at some possible wireless solutions that are already available Commercial off the Shelf (COTS) that can be used to connect remote sensors and equipment to central monitoring systems. There are many technologies out there on the market for a wireless solution that can suit any and all needs, but choosing the right solution for a particular application can be a daunting task with numerous associated risks. If circumstances allow one to select an RF standard or off-the-shelf solution, then the options are fairly clear. In terms of range, penetration, frequency, data rate, and etcetera, there is a set base line of options available to choose from.

It is all about picking the right technology for the job. Listed here are the most common wireless solutions on the market today.

In **Table 2.1** is a list of some short range wireless application solutions that are categorized by use cases to help visually determine which technology is best for each application.

<u>Applications</u>	<u>Use Cases</u>				
	Voice	Data	Audio	Video	State
Bluetooth ACL / HS	X	Y	Y	X	X
Bluetooth SCO/eSCO	Y	X	X	X	X
Bluetooth low energy	X	X	X	X	Y
Wi-Fi	(VoIP)	Y	Y	Y	X
Wi-Fi Direct	Y	Y	Y	X	X
ZigBee (Legacy)	X	X	X	X	Y
ANT (Legacy)	X	X	X	X	Y

Table 2.1: Short range Wireless Application Areas [3]

State = low bandwidth, low latency data and low power

In **Table 2.2**, the chart compares various technical specs with wireless COTS technology.

Wireless	Wi-Fi	Bluetooth	Bluetooth LE	ZigBee	ANT
Frequencies	2.4GHz and/or 5GHz	2.4GHz	2.4GHz	915MHz	2.4GHz
Channels	16 @ 2.4GHz 80 @ 8GHz	79 @ 1MHz	40 @ 2MHz	10 @ 915MHz 26 @ 2.4GHz	Varies@1MHz
Range (Indoor)	70m	1m thru 100m	10m thru 100m	20m	30m
Range (Outdoor)	>160m	>100m	>150m	>100m	Antenna dependent>200m
Data Rate (Max)	54Mbps/s (12Mbps/s typical)	3Mbps/s	1Mbps/s	250Kbit/s	1Mbps/s
Transmission Scheme	DSSS	Adaptive FHSS	DSSS	DSSS	adaptive isochronous network
Power Sources	Wired	Battery/Wired	Battery/Wired	Battery/Wired	Battery/Wired
Uses	Cable replacement, large data transfer, networking	Short distance cable replacement	Cable replacement, Monitoring, Controlling, and Data Transfer	Monitoring and Controlling	Short distance cable replacement

Table 2.2: Short range Wireless Application Specs [4]

Here are some important design requirements that should be considered for any wireless design:

- **Range** (Distance): Range is determined by four elements: Transmitter Power, Receiver Sensitivity, Line of Sight (LOS), and Data Volume
- **Antenna:** depend on the communication requirements, like system array and range. *Two Types of Antenna:* Directional and Omni-Directional
- **System Configuration Types:** Point to Point, Point to Multipoint, Low Costs, Longevity, Swift Deployment, Easy Configuration, and Security

When looking for the optimum wireless solution, carefully consider these factors: Noise, Channel Interference, and Signal Echo. Several modulation and transmission schemes have been developed to counter the effects of these interferers. Here are two of the best to look for:

- **FHSS** (Frequency Hopping Spread Spectrum) - This scheme requires narrow bandwidth. Data is transmitted through a single channel at a time, but the channel is constantly and rapidly changing or hopping [5].
- **DSSS** (Direct Sequence Spread Spectrum) - This scheme requires large bandwidth. Data is transmitted simultaneously over every available channel, making it a bit more reliable in noisy environments.
 - **Note:** Use caution when designing wireless networking systems, make sure that all wireless transmitters, nodes and equipment support the same transmission scheme.

In addition to considering these transmission schemes, there are many proven wireless standards available that can be implemented and developed into one's design that already takes into consideration signal reliability, security, distance, speed, and efficiency. Determining the best solution depends on what the application is and the needs involved. Here is a quick-look at the wireless options one has along with some corresponding pros and cons.

Wi-Fi (IEEE 802.11 b/g/n)

- **Pros** – This is the typical method of networking for businesses, homes, and offices. Wi-Fi is widely used for its high data transfer rates between 12MB/s up to 54 MB/s [5].
- **Cons** - However, complying with this standard requires excessive overhead in relations to power consumption, processor resources, short range (160m max), software, and the physical component size, making it less than effective in most situations [5].

Bluetooth (IEEE 802.15.1)

- **Pros** - Bluetooth has gained popularity because of the compact physical size and its instantaneous network setup, which comprise of three different classes (Class1=1m, Class2= 10m, Class3=100m) that allow data to move from 3m to up to 100m away [5].
- **Cons** - Bluetooth has a relatively high duty cycle, low data throughput up to 3 Mbit/s, and with its low penetration qualities requires device to maintain a fairly direct line-of-site [5].

ZigBee (IEEE 802.15.4)

- **Pros** - It is far more power conservative than Wi-Fi and Bluetooth because of its advanced sleep and sniff abilities. Additionally, it operates with an even smaller physical footprint than Bluetooth, and has high penetration ability [5].
- **Cons** - Zigbee has poor interoperability with a low data rate up to 720 kbit/s. Because it is relatively unpopular, hardware developers are still trying to improve this architecture [5].

Bluetooth Low Energy (IEEE 802.11)

- **Pros** – This method has the lower power requirements on the market compared to other design like WIFI, Bluetooth, ZigBee, and Ant. It also has, in comparison, the lowest cost, better signal penetration, and has the fastest development platform available [5].
- **Cons** – designed for low energy, the communication rate was not a factor in the design so information is only transmitted in small bursts of data; of course this could be considered a 'Pro' or an advantage depending on the end use of this technology [5].

With today's technology, the possibility of implementing a wireless solution that can exchange information, as a replacement for wires or cables, is easier than ever; but picking the proper wireless solution to connect these remote sensors and equipment can pose a varying degree of complexity. For instance, multiple obstacles could present unique challenges that would degrade signal integrity and range to these data collecting devices.

There are a wide variety of data communication solutions that could resolve these challenges, as listed above. BLE is the best technical choice for sending discrete sensor data and equipment information while maintaining portability, barrier penetration, size, and cost.

If it is not desired to build an RF system from scratch, *modules and SDKs can provide an excellent alternative approach while also providing a rapid prototyping and deployment platform. See Section 6 on Hardware for more information on the use and implementation of BLE using COTS SDK with the corresponding BLE module (all-in-one radio plus microcontroller (SOC)).*

2.1. What is Bluetooth?

Bluetooth technology is a wireless communications system (frequency bands from 2.4 to 2.485 GHz) intended to replace the cables connecting many different types of devices, from headsets and speakers to automotive infotainment systems and test equipment. It was invented in 1994 by Ericson Mobile as an alternative to using wired cables with the design based on frequency-hopping spread spectrum technology [6].

Bluetooth is managed by Bluetooth Special Interest Group (SIG) and directs the specifications for common short range wireless applications. These are written, tested and maintained by the Bluetooth SIG with over 25,000 member companies. The SIG owns the Bluetooth® trademarks and oversees the development of Bluetooth standards, the licensing of the Bluetooth technologies and trademarks to the manufacturers, like CSR and Qualcomm [7].

The Bluetooth radio [2]:

- Unlicensed 2.4 GHz ISM (Industrial Scientific Medical) band, 1 M symbols/s, GFSK, 4PSK or 8PSK
- 1 MHz channel spacing, with frequency hopping applied to combat interference and fading
- Adaptive Frequency Hopping, for co-existence with Wi-Fi, etc
- Up to 100 mW

IEEE Bluetooth 4.0 Generic Alternate MAC/PHY (AMP) can use additional radios [2]:

- IEEE 802.11(a,b,g,n) and WiMedia UWB (ECMA-368)

How much energy does traditional Bluetooth use?

Bluetooth is connection oriented, which means when a device is connected; a link is maintained, even if there is no data flowing.

When the device is in Sniffer mode, it is allowed to sleep, reducing power consumption to give months of battery life with a Peak transmit current typically around 25mA. Even though it has been independently shown to be lower power than other radio standards, it is still not low enough for coin battery cells and energy harvesting applications [2].

Bluetooth uses:

- Mobile phones, including 'Smartphones'
- Wireless controllers for video games
- Voice headsets and "Car kits"
- PCs
- M2M applications – credit card readers, industrial automation
- stereo headsets and speakers

2.2. Bluetooth Low Energy (BLE)

Bluetooth Low Energy, also known in the industry as "Bluetooth Smart", was originally introduced as an in-house project under the name 'Wibree' by Nokia in 2006. It is a light-weight subset of classic Bluetooth and was merged into the main Bluetooth standard by SIG as part of the Bluetooth 4.0 core specification in 2010.

Think of BLE as a stripped down lean version of Bluetooth. Bluetooth and BLE are used for two very different purposes. Depending on what the 'End User' requirements are will determine the appropriate wireless technology to develop.

Bluetooth can consume the life of a battery quickly because it was designed to exchange a lot of data in a short amount of time; plus, it costs a lot more. BLE is used for applications that do not need to handle large amounts of data (throughput) and can therefore remain on battery power for years. BLE is intended to provide considerably reduced power consumption, and low cost while maintaining very similar communication range to standard Bluetooth; otherwise known as, radio coverage.

The difference between Bluetooth and BLE is that there is no data throughput because BLE does not support streaming data. After a connection has been established (paired), BLE spends most of the time in sleep mode waiting to send/receive the next set of device status information also known as 'expose state", such as the Battery Level. It has a data rate of 1Mbps allowing for quick data transfer of small chunks or data packets (kB), exposing the state of the device to retrieve that information. This status update interval rate delay can be programmed from 7ms up to 4s between data polls. Once the data has been transferred, a few milliseconds, the BLE goes back to sleep to conserve battery; whereas, Bluetooth stays on the entire time regardless if information is being transferred.

Key Features of BLE that differ from standard Bluetooth [2]:

- The PHY or physical layer has parts that were derived from the Bluetooth Radio
- Advertising was altered to simplify the discovery & connection
- Asynchronous connection-less MAC: used for fast transactions with low latency, (e.g. 3ms from start to finish)
- The Interface model, Generic Attribute Profile (GATT), has been simplified between the devices and software
- Asynchronous Client / Server architecture was redesigned to have the lowest cost and ease of implementation
- BLE was designed for exposing state of devices and retrieving the information
 - Data can be read at any time by a client, such as a Smartphone App
 - It's good at small, discrete data transfers
 - Data can be triggered by local events

How much energy does BLE use?

Calculating the energy per transaction: the upper bound transaction period is roughly 3ms with a TX Power Amplifier drawing 10mW (65nm Si process) @ 1Mbit/s, this is 6.7mA for 1.5v battery. $0.01 \text{ W} \times 0.003 \text{ sec} = 30 \text{ micro Joules}$

How long could a sensor last on a coin cell battery?

- An example battery: Lenmar WC357, 1.55v, 180mAh, \$2-5.
- $180\text{mAh}/6.7\text{mA} = 27\text{Hr} = 97200 \text{ seconds} = 32.4\text{M transactions}$
- Suppose this sensor sends a report every minute = 1440/day
- For just the BLE transactions, this is 15,000 days, which is equivalent to more than 40 yr.
- This far exceeds the life of the battery and/or the product
- Sensors could run on scavenged power if so desired, e.g. ambient light

Simply put, Devices have data and Web Services want this data, BLE provides the technology to connect these two. With the rise in popularity, BLE is the new wireless radio standard enabling the 'Internet of Everything'. Bluetooth versus BLE, the IoE Difference: to read more go to: <http://www.csr.com/products/markets/internet-everything>

3. Developing with Bluetooth Low Energy

BLE is an intelligent, application-friendly technology that is supported by every major OS. While the power-efficiency of BLE makes it perfect for devices needing to run off batteries for long periods of time, the real beauty of BLE is in its ability to work with an application on tablets, smartphones and PCs that consumers already own [9].

Not only does the technology costs less than other wireless solutions on the market, but it also offers a flexible developmental architecture for creating applications to bring everyday objects like Physical activity monitors, Blood glucose monitors, and even the remote control into the connected world and have them communicate with these applications [8].

Support for BLE (which is a subset of BT 4.0) is available on most major platforms, such as, the versions listed below:

BLE Platform Support [10]

- iOS5+ (iOS7+ preferred)
- Android 4.3+ (numerous bug fixes in 4.4+)
- Apple OS X 10.6+
- Windows 8 (**XP, Vista and 7 only support Bluetooth 2.1**)
- GNU/Linux Vanilla BlueZ 4.93+

There are plenty of wireless solutions out there for developers like engineers and product designers, but what makes BLE platform so exciting is that it's undoubtedly the easiest way to design something that can communicate to any modern mobile platform out there (Android, Windows phones, iOS, and etcetera), and particularly with Apple devices. BLE is the only hardware design option available that doesn't require running around in circles to legally market products for Apple iOS devices.

BLE makes it easy for designers to create solutions that will work with the billions of Bluetooth enabled products already in the market today, which means developing with BLE is only limited by the imagination.

The Technical explanation of the BLE Profiles and Protocol Stack could be listed in this part of the report but would be extensive and outside the scope of the project. Being familiar with the BLE architecture and function are not necessary to get started with prototyping and referencing the project in this document. It is left up to the end-user to become familiar with the underlying BLE Technology.

3.1. Bluetooth Low Energy Radio – The Physical Layer

Below is the RF Physical level of BLE, it is put here for comparison to **Section 2.1**.

The BLE Radio [2]:

- Frequency Hopping in BLE is the same adaptive frequency hopping commonly used for all versions of Bluetooth technology. It minimizes interference from other technologies in the 2.4 GHz ISM Band.
- Latency supports connection setup and data transfer as low as 3ms, for a short communication burst before quickly tearing down the connection
- Larger modulation index gives better range than Bluetooth Basic Radio (~30 meters)
- 40 Channels on 2 MHz spacing with frequency hopping applied to combat interference and fading. Efficient multi-path benefits increase the link budgets and range. See next **Figure 3.1** for a graphical representation of the frequency spectrum used on BLE.

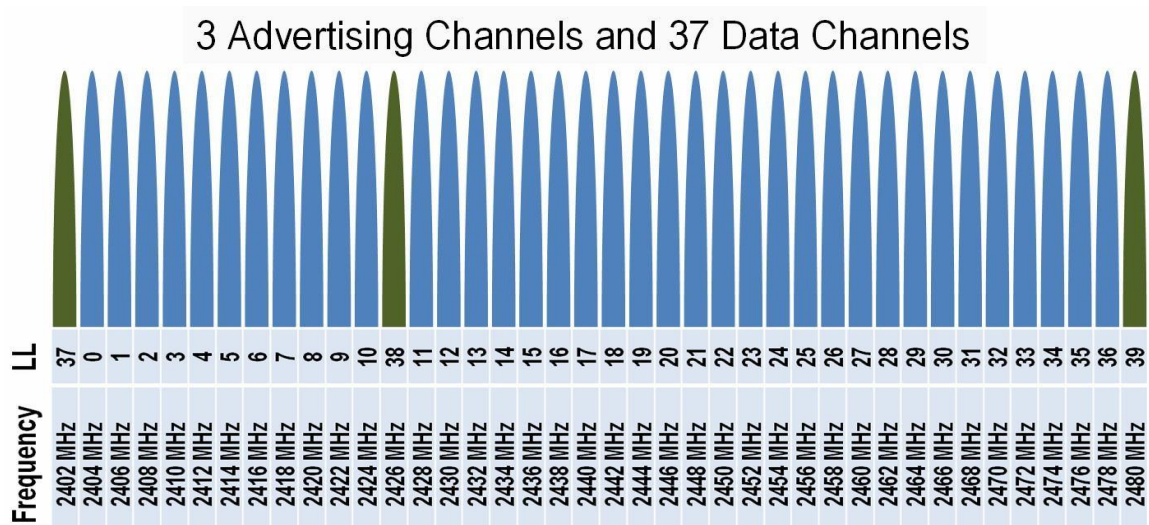


Figure 3.1: Bluetooth Low Energy - Frequency Channels [2]

Additional BLE Radio details [2] [10]:

- Range: ~150m open field. Increased modulation index provides a larger range > 100m
- Output Power: ~10mW (10dBm)
- Max Current: ~15mA
- Latency: allows an application to form a connection and then transfers the authenticated data within a few milliseconds
- Topology: Star configuration allows for one-to-many connections
- Data Transfers: data packets (8 octet min up to 27 octets max) are transferred at 1 Mbps
- Connections: > 2 billion devices use a 32 bit access address on every packet
- Modulation: GFSK @ 2.4 GHz ISM Band for all Data Transfers
- Robustness: Adaptive Frequency Hopping, 24 bit CRC on all packets ensuring the robustness
- Strong Security: 128bit AES CCM provide strong encryption and authentication of data packets
- Sleep current: ~ 1 μ A
- Modes: Broadcast, Connection, Event Data Models Reads, Writes
- Sniffer: advanced sniff-sub rating achieves ultra-low duty cycles, conserving battery life

4. Project Summation

The 'Project', Bluetooth Low Energy for use with MEM Sensors, was to design a Low power wireless solution to transmit MEM and other sensor data to a remote monitoring and / or recording device such as a PC or Smartphone. This design is meant to be utilized as a BLE reference platform for developers, designers, engineers and students who wish to implement a wireless solution into their own project.

The design and Implementation of a wireless solution though it is a complex and challenging task can be simplified through baseline reference designs and prefabbed hardware already on the market. Following this project as a guideline can reduce these complexity and challenges.

High level descriptions to sum up this project:

- Most design projects require the need to transmit sensors information to a receiving station where wires or cables are not desired. Most sensors send out information about the surrounding environment that corresponds to the function of that sensor.
- Sensors could be anything from bio-electronic devices, Human Interface Devices (HID), to even wearable devices; all these sensors supply information or data of some kind. The data just needs to get from point A to point B.
- See **Figure 4.1** for a graphical representation or example:
 - Here, some Environmental Sensors have been arbitrarily chosen to represent sensors (data sources) that need to transmit the state of the surrounding Environment; the type of transmitted data / information is based on the function of the specific sensor.
 - The BLE Transceiver for this task (project) is the CSR1010 Module, a readily available solution that could quickly be introduced into an existing design as a cable or wired replacement.
 - Most sensors are I2C / SPI compatible, the module supports these interfaces. If the desired sensor data has an analog signal output, a readily available Analog to digital converter (ADC) IC can convert the analog signal into a digital I2C or SPI format.
 - The Smartphone allows the information transferred from the Module to be displayed resident to that phone screen courteous of the APP Software
 - The CSR1010 handles all the wireless transactions and acts as the Slave Device to the Master Smartphone App.

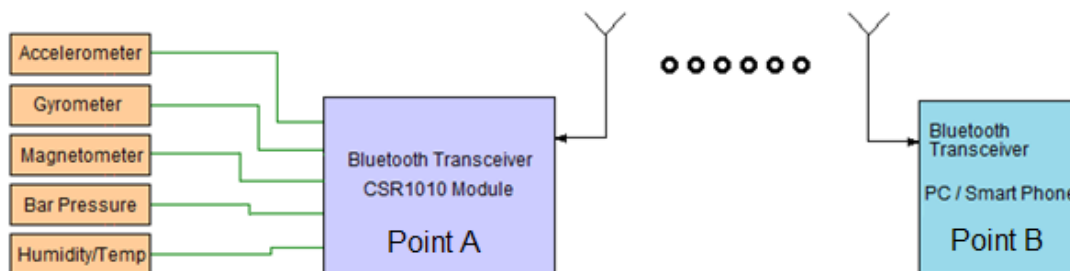


Figure 4.1: Simplified Overview

In **Figure 4.2**, represents the upper level project Development Hierarchy that shows the levels of the design in an organized manner. It is recommended to the End-User Designer to organize the project in a similar manner to maintain focus and a means to give content labels for documentation.

- Hardware Level – CSR1010 Module and the Android Smartphone. See [Section 4.1](#)
- Software Level – The Service is provided by the GATT Service resident on the CSR1010 Module (Hardware) while the Client is the Android App Software. See [Section 4.2](#)
- Code development and deployment – A brief walk-through. See [Section 4.3](#)

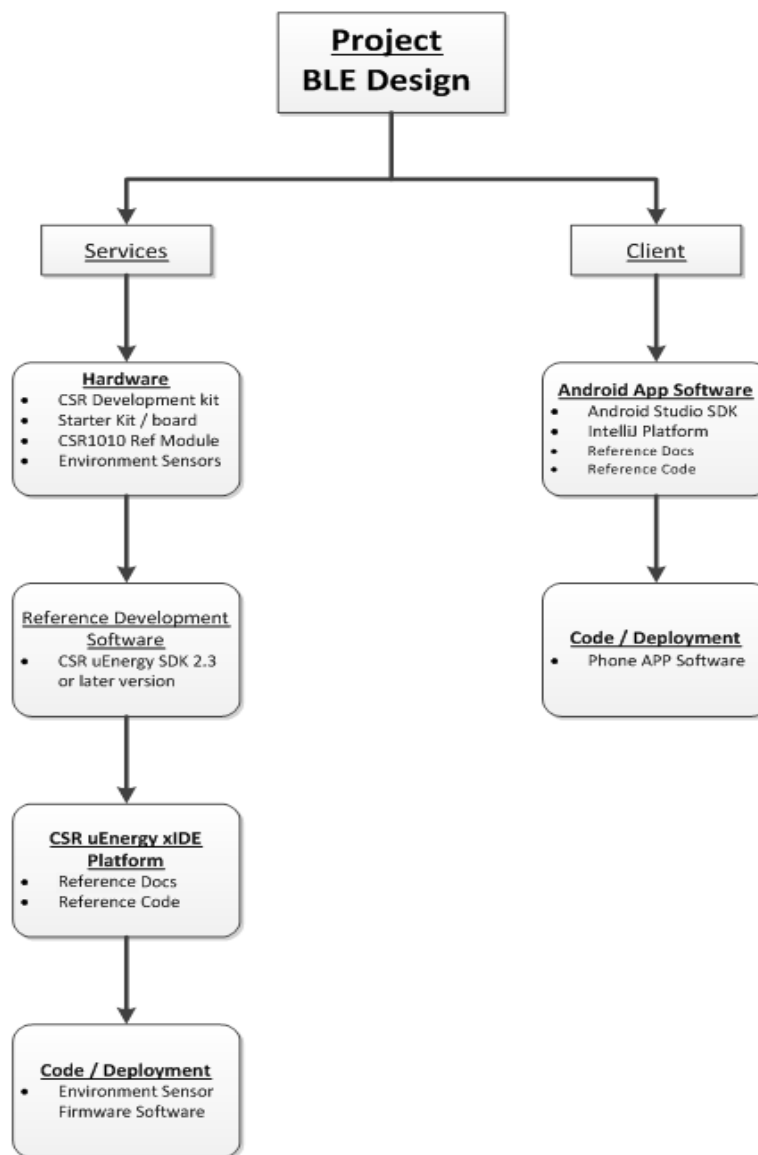


Figure 4.2: Project Hierarchy

4.1. Hardware Level

The hardware provides the wireless connectivity to allow for the MEMS sensors to exhibit the information or data to the display of the Android Smartphone.

- The hardware required for this reference design are necessary to reproduce the end results discussed in this document. It will be the base design that can be modified to suit the desire of the developer. See [Section 5 – Hardware](#), for details and purchasing info.
 - CSR1010 CSR μ Energy Starter Development Kit with the attached CSR μ Energy CSR1010 Module board. Kit also comes with the SDK on CD_ROM
 - Environmental Sensor Board
- The recommended Smartphone to be used with this Reference design is any Google Nexus (5 or later), or Samsung Galaxy (S4 or later). One of these types of Smartphones will be needed in [Section 9 – Smartphone App](#).

4.2. Software Level

Software installations are required by both the CSR1010 Module and the Android Smartphone to provide the development environments necessary for making modifications to the reference designs; it is also used to deploy the source code onto the hardware. The following platforms need to be installed and executed local to the machine that the hardware (CSR1010 Module and Smartphone) will be interfacing.

- CSR1010 Module design reference uses a custom software development platform created by CSR. This platform is the CSR's Integrated Development Environment (xIDE) and provides the means to develop and deploy the Environment Sensor Application, also known as the Firmware. Firmware is the Software that is installed onto the CSR1010 Module's on-board electrically erasable programmable read-only memory (EEPROM) to allow for the program to load upon power-on. This firmware provides the appropriate Services to communicate with the MEMS sensors and relay that information to the Client (Phone). The CSR xIDE has been bundled with reference source code applications, board design files and documents into a single executable installer utility called the CSR μ Energy Software Development Kit (SDK). See [Section 4.2.1](#) for installation instructions.

- The Smartphone Application design reference uses software that was developed in Java with the Android Studios SDK for the Android operating system and can only be deployed to Android capable Smartphones. The Android Studios SDK is the development environment that provides the platform to develop the java source code and deploy it onto an Android Smartphone, also known as the 'App'. When this 'App' is paired with the Server (CSR1010 Module), it will be able to retrieve information as the Client and display that information to the screen of the phone. In order for the Android Studios Environment to function properly, the Java Kernel and Java development environment also needs to be installed. This additional software needs to be installed before the Android Studio SDK software is installed in order to get the Android development platform environment up and running properly. See **Section 4.2.2** for installation instructions.

4.2.1. Installing the SDK Software

If you have already installed the software go to the next Section.

This section describes how to install the SDK software that came with the CSR μ Energy Starter Development Kit for the CSR1010 Reference Design Module; supplied on the CD_ROM. Alternatively, one could download the latest SDK Software directly from CSR's secure website. To gain access to CSR's secure website, register at <https://www.csrsupport.com/register.php>. This will give you access to all the reference designs, software and a wealth of information that is not available on the CD_ROM. After access has been granted, the latest CSR uEnergy SDK version can be download from these locations:

- https://wiki.csr.com/wiki/Main_Page
- <https://www.csrsupport.com/uEnergy/Software>
- https://www.csrsupport.com/download/56326/CSR_uEnergy_SDK-2.5.0.20.exe

It is also recommend to register and become a Bluetooth Special Interest Group (SIG) member so that access can be made to their Developer Portal, documentation, forums and other information to assist in Bluetooth development. <https://www.bluetooth.org/login/register/>

Install and launch CSR μ Energy SDK executable. Follow the guidance provided by the Install wizard. During the installation procedure, CSR recommends to accept the default configuration options that are provided at every prompt.

- a. If installer is from a CD: Run the **CSR_uEnergy_SDK-x.x.x.x.exe** installer application from the **SDK** directory of the CD-ROM (where x.x.x.x is the SDK version e.g. 2.3.0.31).
- b. Otherwise run the downloaded single executable installer directly

When the installation process is complete, it is recommended to check the box 'View the Support Documentation' before clicking the 'Finish' button to exit. An HTML based directory system will allow one to peruse though the support documentation. This supporting information can also be accessed from the home page of the xIDE tool once installed.

After the CSR uEnergy SDK utility completes the installation process, it will then be safe to connect the Target board (CSR1010 CSR μ Energy Starter Development Kit) to the PC using the mini-USB cable that came with the kit. The device drivers necessary to connect to the Target board are automatically installed on the PC at that time.

Also, after the SDK installation process has concluded, the necessary reference design files like the Environment Sensor Firmware source code and CS-314507-AN Environment Sensor Application Note can be found in **C:\CSR_uEnergy_SDK-2.x.x.x\apps** directory. This will be the directory that provides the workspace when the project is opened in the xIDE; those details are in [Section 6 – Firmware](#).

Note: During the project development cycle, chapters 5-7 of this document developed into the CS-314507-AN_CSRuEnergyEnvironmentSensorApplicationNote and is available with the CSR_uEnergy SDK installer, along with the Firmware source code. The files were originally developed using the CSR uEnergy SDK, version 2.3.0.31 utility. These reference design files were later bundled into subsequent revisions of the CSR uEnergy SDK. This means that there is a chance that the CD_ROM that came with the CSR μ Energy Starter Dev Kit, purchased through a third party vendor like Digikey, may have an SDK version 2.3 or older. Therefore, there is no guarantee what the latest SDK platform will have these files available on the accompanying CD_ROM. It is recommended to download the latest SDK from CSR.

4.2.2. Installing the Android Development Platform

If you have already installed the software go to the next Section.

This section describes how to install the Android Development environment. One of the most important parts of getting started with a new platform is setting up the environment. *Particularly* for beginners, it's important to take time here to follow each step methodically. Even if the steps are followed perfectly, there may be some small issues that require some troubleshooting later. Be sure to Google any issues that occur.

The Android development environment requires the following programs:

- **Java SE** (Standard Edition Kernel) Development Kit
- **IntelliJ IDEA** - Java Integrated Development Environment (IDE)
- **Android Studios** Software Development Kit (SDK)

The sequence of the software installation listed here is to prevent or at least minimize the occurrence of any future issues. Do not open these programs until **Section 8.1.1**. If the software is installed properly, there should be no issues with the Android development experience.

Java SE Development Kit (JDK)

The Java development kit provides the resources to compile the source code and execute in a Java Runtime Environment. The Android Studios relies on this engine to process the source code. It is recommended to download the latest version of software to minimize bugs and security risks found in previous revisions.

To install the Java SE Development Kit, in this case `jdk_8u40_windows_x64.exe`, go to:
<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>.

Find the appropriate file that matches your computer then accept license agreement and download JDK installer. Follow the prompts, recommend installing with the default directory.

Some recommended Java programming resources for Tutorials and reference guides:

- <https://docs.oracle.com/javase/tutorial/>
- <http://code.tutsplus.com/tutorials/java-tutorial--mobile-2604>
- http://www.tutorialspoint.com/java/java_basic_syntax.htm

Java IntelliJ IDEA IDE

Java IntelliJ IDEA is a free open source IDE for Android development and provides out-of-the-box Android support that includes intelligent code editing assistance, on-the-fly code analysis, built-in Android tools, and other features for a professional development of Android applications [1].

Until around the end of 2014, the officially supported integrated development environment (IDE) was Eclipse using the Android Development Tools (ADT) Plugin. As of 2015, Android Studios, made by Google and powered by IntelliJ, is the official IDE; however, developers are free to use others [2].

To install IntelliJ IDEA, go to: <https://www.jetbrains.com/idea/download/>
To begin download (free version), select the 'Download Community' button. Follow the prompts by the install wizard; it is highly recommended to install with all suggested defaults.
For more information about the IntelliJ IDE: <https://www.jetbrains.com/idea/help/android.html>

Android Studios SDK

The Android Studio is a program that has all the tools necessary to make a professional application. All of the files used in the development of the 'app' are managed by the IDE. The IDE program, IntelliJ, is used to edit the source code files and to manage the projects. Follow this hyperlink to directly download and install the Android Studio 1.1, bundled SDK 135.1740770 from google: <https://dl.google.com/dl/android/studio/install/1.1.0/android-studio-bundle-135.1740770-windows.exe>

Note: Warning, after installing the Android Studios Platform do not open the program until after the other files listed above have been installed. This will prevent complications later.

For more information regarding Android Studios, go to these recommended websites:

- <https://developer.android.com/sdk/index.html>
- <https://developer.android.com/tools/studio/index.html>
- <http://code.tutsplus.com/tutorials/getting-started-with-android-studio--mobile-22958>
- Here is a good source of open source android applications worthy of investigating:
https://en.wikipedia.org/wiki/List_of_free_and_open-source_Android_applications

4.3. Code Development and Deployment

Here is the introduction to the topic of code development and deployment that will be discussed in upcoming chapters. The following sections will give a brief description on the interfacing of the hardware to the PC, the development environment, source code, functional overview and the deployment instructions on compiling and downloading the application.

Firmware Development Environment: Starting with [Section 6](#), will provide guidance to set up and use the xIDE for the CSR μ Energy Starter Development Kit with the CSR1010 Module.

Android App Development Environment: Starting with [Section 8](#), will provide guidance to setup and use the Android Studios Environment for the Android Smartphone.

5. Hardware

The 'Project', Bluetooth Low Energy for use with MEM Sensors, defines a readily available (COTS) reference design hardware that is meant to go along with this document. For this particular design, the CSR μ Energy single-mode BLE solution was chosen (CSR uEnergy BLE Development kit with the CSR1010 Reference module, and the MEMS Sensor Board). It is highly recommended to acquire the mentioned reference design hardware as to gain a better understanding of the concepts, designs and programming environments.

Important Note: Do not connect hardware before installing the appropriate software

Items recommended:

- Purchase CSR uEnergy BLE Development Kit and the necessary SDK from Digikey.com:

[DK-CSR1010-10169-1A-ND](#): EVAL KIT BLUETOOTH LOW ENERGY (\$99)

For additional information regarding the H13137V3 CSR μ Energy Starter Development Kit, see **Figure 5.2**. Further documentation can be accessed at:

<http://www.csr.com/products/bluetooth-smart-starter-development-kit>.

- Purchase CSR's μ Energy Environmental MEMS Sensor Board from www.Digikey.com:

[DK-ENV_SENS-10224-1A-ND](#): KIT ENERGY TEMP/PRESSUR4 SENSOR (\$49)

For more information regarding the H13229V2 Environmental Sensor board can be seen in **Figure 5.3**. Additional information can be accessed at:

<http://www.csr.com/products/bluetooth-smart-environmental-sensor-board>

5.1. CSR μ Energy CSR1010 Module board

The CSR1010 is a Bluetooth SIG qualified design and is a single-mode Bluetooth low energy solution. This qualification has also been extended to include the CSR1010 low-cost module as seen below. For more information on the SIG qualification design listings visit the website: https://www.bluetooth.org/tpg/QLI_viewQDL.cfm?qid=17702

- A feature of the single-mode is a lightweight Link Layer providing ultra-low power idle mode operation, simple device discovery, and reliable point-to-multipoint data transfer with advanced power-save and secure encrypted connections at the lowest possible cost.

Figure 5.1 shows the CSR μ Energy CSR1010 Module board.

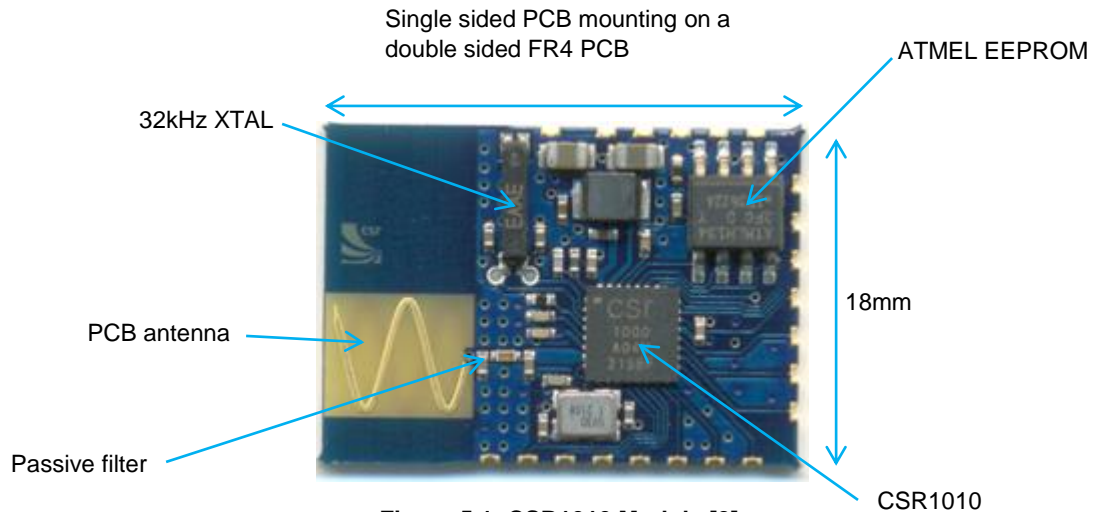


Figure 5.1: CSR1010 Module [2]

For more information on the CSR1010 module, schematics, design walk-through and its reference designs: See CS-218270-DD CSR1010 Hardware Design Review Template [19] on the <https://www.csrsupport.com/document.php?did=39334>

Board design files (gerbers and etc) are also provided at this directory that a developer may use as a reference in their own designs.

- The CSR1010 QFN IC Data Sheet can be located at C:\CSR_uEnergy_SDK-2.4.5.13\doc\support\docs\CS-231985-DS_CSRuEnergyCSR1010QFN.
- For more information regarding the onboard CSR1010 Module:
<https://www.csrsupport.com/document.php?did=39359>

5.2. CSR1010 CSR μ Energy Starter Development Kit

Figure 5.2 shows the CSR μ Energy Starter Kit board with the solder mounted CSR1010 Module. CSR1010 kit part no. DK-CSR1010-10169

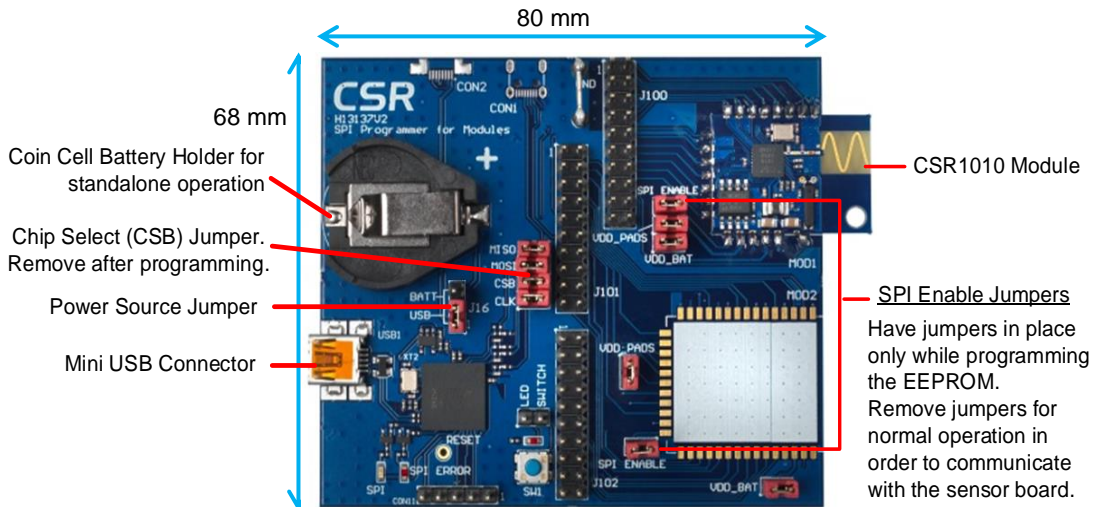


Figure 5.2: CSR μ Energy Starter Kit [2]

See *CSR μ Energy Starter Dev Kit Quick Guide* for more information on the CSR μ Energy Starter Kit.

5.3. Environmental Sensor Board H13229

Figure 5.3 shows the H13229 Environmental Sensor board that was developed for this project. The board is now in production and is available: [CSR part no. DK-ENV_SENS-10224-1A](#) For board design files, go to <https://www.csrsupport.com> if these files are not otherwise available.

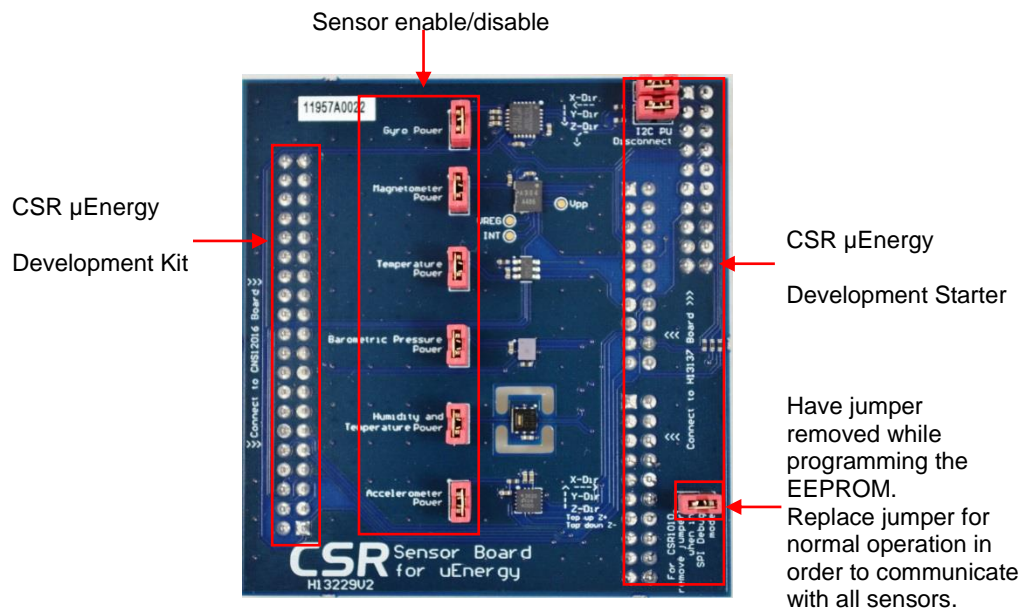


Figure 5.3: Environmental Sensor Board [1]

The board features the following sensors:

- Temperature sensor - STMicroelectronics STTS751 sensor
- Pressure sensor - Saw Components T5400 sensor
- Humidity sensor - Sensirion SHT21 sensor
- Accelerometer sensor - Analog Devices ADXL362 sensor
- Magnetometer sensor - Aichi Steel AMI304E sensor
- Gyro/Angular Rate sensor - InvenSense ITG3050 sensor

Figure 5.4 shows the Environment Sensor Board attached to the Starter Kit board using the three small header sockets located on the right hand side of the board.

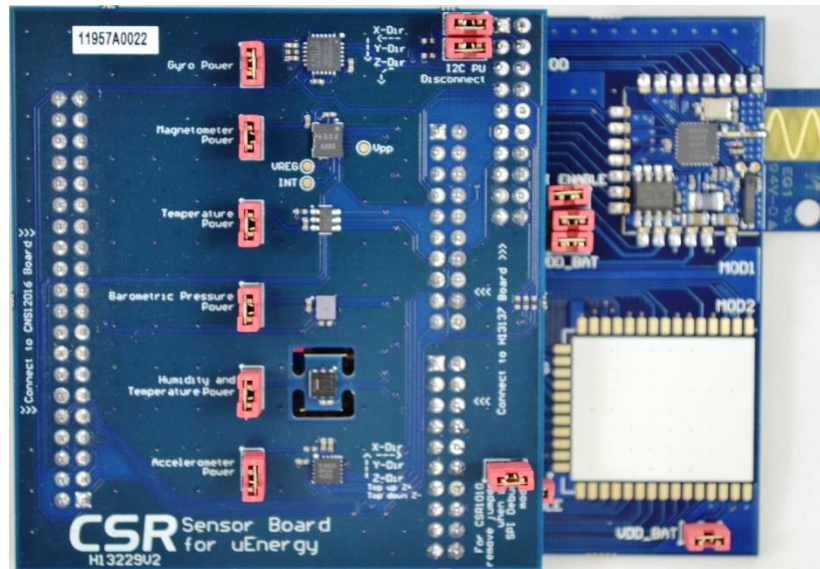


Figure 5.4: Environment Sensor Board Attached to the Starter Kit Board

Board Jumper configurations Use Cases:

- To program the Device (deploy to embed the firmware to EEPROM):
 - To program the device, the jumpers need to be in place with the names: CSB (SPI Chip Select), and SPI Enable
 - Jumper on bottom of the SDK board needs to be configured in the USB position, with USB cable tethered to computer

- To operate the Device after programming, including remotely / untethered (Battery)
 - To operate the device, need to remove jumpers with the names: CSB (SPI Chip Select), and SPI Enable
 - Jumper on bottom of the SDK board needs to be configured in the Battery position in order to operate remotely.

To power on the device:

- Ensure the power source is provided i.e. either the board is attached to a PC using the mini-USB cable, or the coin cell battery is fitted.
- Ensure that the corresponding power source is selected, use power source jumper J16.

To power off the device, either:

- Remove the power source currently selected by jumper J16 i.e. by disconnecting the mini-USB cable, or by removing the battery.
- Set the power jumper to a power source that is not provided. In **Figure 5.2** shows the jumper in the Battery position.

6. Firmware Development Environment

The reference source code applications, the supporting documentation and the CSR xIDE were bundled with the CSR uEnergy SDK, as stated in Section 4.2.1. This reference software (Environment Sensor) should now be located in C:\CSR_uEnergy_SDK-2.x.x.x\apps\ directory; provided that the default prompts were adhered to during the installation process.

The ...apps directory has a selection of reference applications that represent a large variety of wireless technology currently being used out on the market. These resources can be the basis for new product designs and remarketed with new capabilities.


Much like the Environment Sensor reference application used in this project, Developers can make use of these other reference applications as the basis for developing custom applications. These reference applications demonstrate basic functionality and conform to the relevant Bluetooth Smart Profiles defined by Bluetooth SIG. Adopting this approach greatly reduces the effort required to develop a final product application and allows software engineers to concentrate on further developing additional features and functionality for new products.

This chapter describes the procedure for loading a reference application as a project in the xIDE and then running that code on a hardware development platform. It is intended to provide developers with the information required to begin using xIDE to develop applications with the single-mode CSR1010 development platform.

Note: The xIDE provides a common programming environment with all the tools and utilities required to write, build, run and debug code. Chapter 6 is only focused on developing reference applications using xIDE, it is not intended to go into great detail of all the features and the code. For more information on how to use the xIDE software, go to the SDK support docs: C:\CSR_uEnergy_SDK-2.4.5.13\doc\support\docs\CS-212742-UG_CSRuEnergyXIDEUserGuide

6.1. Launch the xIDE

This section details how to launch / open the xIDE.

To launch the xIDE, click the xIDE shortcut  on the desktop (if configured), or from the Windows **Start** menu, chose the CSR μ Energy SDK folder, and then click on the CSR μ Energy SDK (xIDE) to open the development environment.

A window should pop open that looks just like **Figure 6.1**.

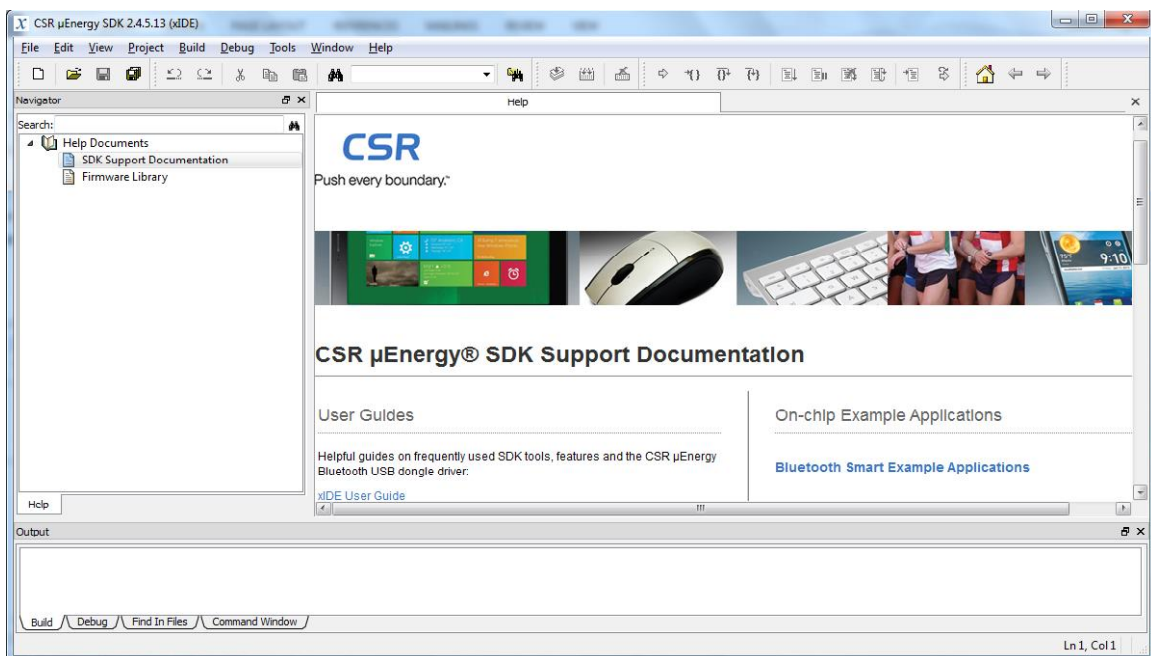


Figure 6.1 – xIDE Platform after Launch

6.2. Open the Environment Sensor Project

This section details how to open the reference design application workspace in the xIDE as a project. To open the environment_sensor workspace in a xIDE project:

1. Click on **Project** in the menu bar.
2. Select **Open Workspace** in the xIDE **Project** menu. An **Open workspace** window appears.
3. Browse to the folder containing the example applications provided in the SDK.
e.g. C:\<CSR_uEnergy_SDK-Version>\apps\environment_sensor
4. Select the project file environment_sensor.xiw and click **Open**, as displayed in **Figure 6.2**

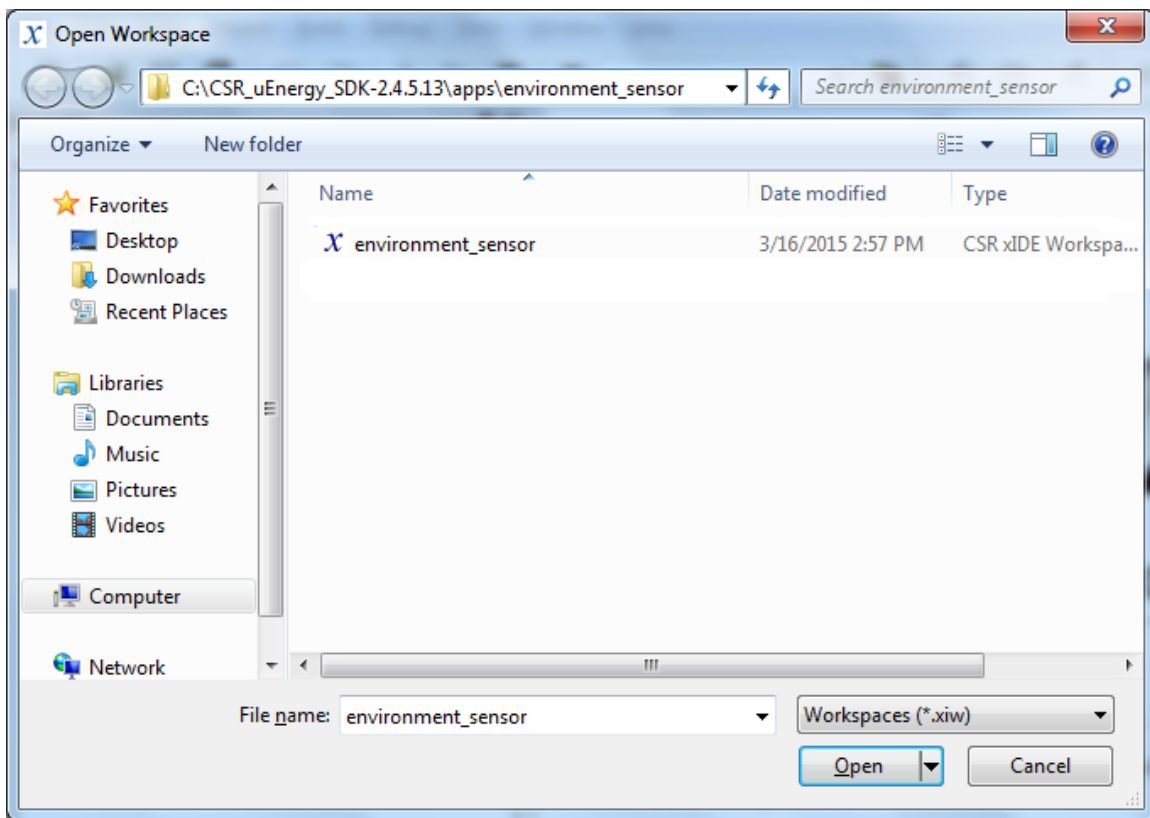


Figure 6.2 – Open Workspace

The remainder of Chapter 6 – Firmware Development Environment, will go over the project files in the Environment Sensors Workspace (**Section 6.3**), how to customize some application features (**Sections 6.4**) and how to build and deploy the new firmware application (**Section 6.5**).

6.3. Application Files

This section of the document lists the files used for developing the Environment Sensor Firmware Application with CSR's xIDE. Point is not to cover any specific C coding guidelines. It is assumed that the end-user referencing this design has at least the basic programming skills.

With the reference material, one does not have to be an expert C programmer to create applications in the xIDE framework. A basic conceptual knowledge is helpful to get caught up to speed. The reference source code is fairly simple and well documented; after reviewing the reference source code, one should get a good understanding and be able to duplicate the results. If there are some programming hurdles, more information about C-programming can be searched on-line. Location list of components: C:\CSR_uEnergy_SDK-2.4.5.13\apps\environment_sensor

As shown in **Figure 6.3**, three folders make up the Application design files:

- C Files Folder: Section 6.3.1
- Header Files Folder: Section 6.3.2
- GATT db files Folder: Section 6.3.3

The next three sub-sections will describe briefly the files that are in each folder.

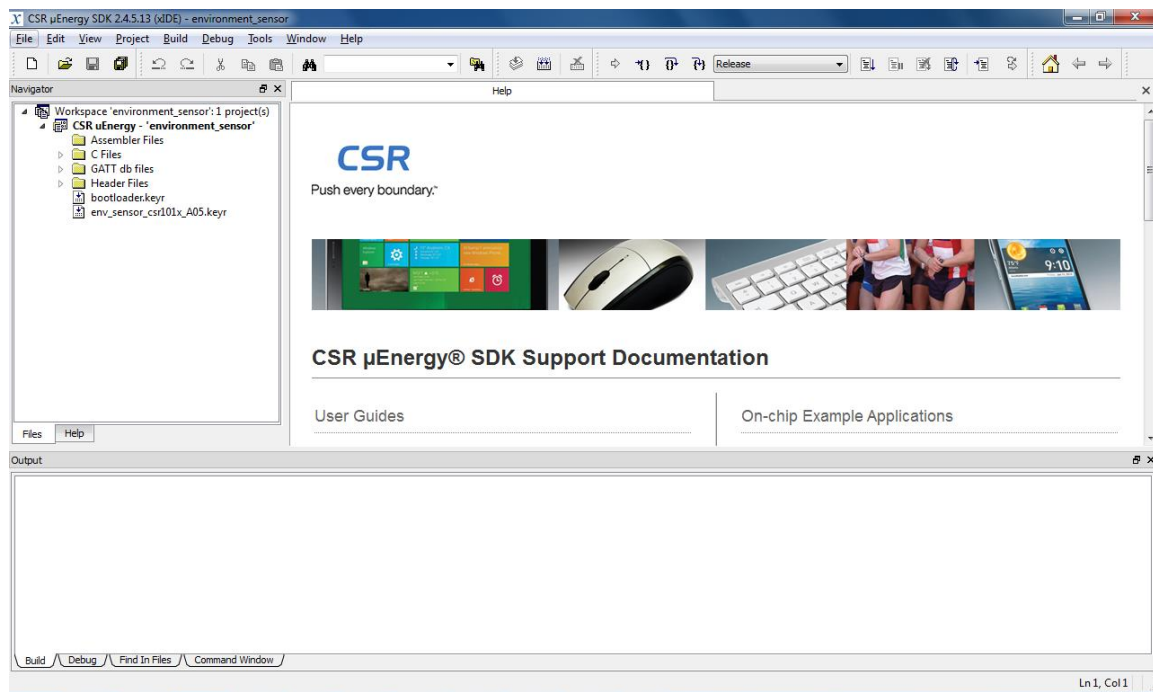


Figure 6.3 – Project File Structure

6.3.1. Source Files

Source files are the .c files and have specified functions; where the action happens.

Table 6.2 listed here, are the source files with the purpose defined for each.

File Name	Purpose
accelerometer.c	Implements the interface functions for communicating with an accelerometer.
accelerometer_emulator.c	Implements functions for emulating an accelerometer.
adxl362_accelerometer.c	Implements functions for communicating with the Analog Devices ADXL362 accelerometer.
ami304e_magnetometer.c	Implements functions for communicating with the Aichi Steel AMI304E magnetometer.
battery_service.c	Implements routines required for the Battery Service e.g. handling read/write access indications on the Battery Service attributes.
dev_info_service.c	Implements routines required for the Device Information Service e.g. handling read/write access indications on the Device Information Service Attributes.
env_sensing_service.c	Implements routines required for the Environmental Sensing Service e.g. handling read/write access indications on the Environmental Sensing Service attributes.
env_sensor.c	Implements all the entry functions e.g. Applnit(), AppProcessSystemEvent() and AppProcessLmEvent(). Events received from hardware and firmware are first handled here. This file contains handling functions for all the LM and System events.
env_sensor_gatt.c	Implements routines for triggering advertisement procedures.
env_sensor_hw.c	Implements routines for hardware Initialization indicating different states by blinking the LED.
gap_service.c	Implements routines for the GAP Service e.g. handling read/write access indication on the GAP Service characteristics, reading/writing device name on NVM etc.
gatt_service.c	Implements routines for the GATT Service.
gyroscope.c	Implements interfacing functions for communicating with a gyroscope.
gyroscope_emulator.c	Implements functions for emulating a gyroscope.
humidity_sensor.c	Implements interfacing functions for communicating with a humidity sensor.

File Name	Purpose
humidity_sensor_emulator.c	Implements functions for emulating a humidity sensor.
i2c_comms.c	Implements the I ² C communicating routines.
itg3050_gyroscope.c	Implements functions for communicating with the InvenSense ITG3050 gyroscope.
magnetometer.c	Implements interfacing routines for communicating with a magnetometer.
magnetometer_emulator.c	Implements functions for emulating a magnetometer.
nvm_access.c	Implements NVM read and write access routines.
pressure_sensor.c	Implements interfacing functions for communicating with a pressure sensor.
pressure_sensor_emulator.c	Implements routines for emulating a pressure sensor.
sht21_humidity_sensor.c	Implements routines for communicating with the Sensirion SHT21 humidity sensor.
stts751_temperature_sensor.c	Implements routines for communicating with the STMicroelectronics STTS751 temperature sensor.
t5400_pressure_sensor.c	Implements routines for communicating with the Saw Components T5400 pressure sensor.
temperature_sensor.c	Implements interfacing functions for communicating with a temperature sensor.
temperature_sensor_emulator.c	Implements routines for emulating a temperature sensor.

Table 6.2: Source Files [13]

6.3.2. Header Files

Header files (.h) are the common shared files that contains stuff that is usually shared with other parts of the code. **Table 6.3** lists the header files and the defined purposes of each.

File Name	Purpose
accelerometer.h	Contains macro definitions for PIO number and function prototypes for communicating with an accelerometer.
accelerometer_emulator.h	Contains prototypes of accelerometer emulator functions defined in accelerometer_emulator.c.
adxl362_accelerometer.h	Contains prototypes of the Analog Devices ADXL362 accelerometer interfacing functions defined in adxl362_accelerometer.c.

File Name	Purpose
ami304e_magnetometer.h	Contains prototypes of the Aichi Steel AMI304E magnetometer interfacing functions defined in ami304e_magnetometer.c.
app_gatt.h	Contains macro definitions, user defined data type definitions and function prototypes which are being used across the application.
appearance.h	Contains the appearance value macro of the Environment Sensor application.
battery_service.h	Contains prototypes of externally referred functions defined in battery_service.c.
battery_uuids.h	Contains macro definitions for UUIDs of the Battery Service and related characteristics.
dev_info_service.h	Contains prototypes of the externally referred functions defined in dev_info_service.c.
dev_info_uuids.h	Contains macros for UUID values of the Device Information Service.
env_sensing_service.h	Contains prototypes of externally referred functions defined in env_sensing_service.c.
env_sensing_uuids.h	Contains macros for UUID values for the Environment Sensing Service.
env_sensor.h	Contains user defined data types and macros for NVM offsets.
env_sensor_gatt.h	Contains prototypes of externally referred GATT routines defined in env_sensor_gatt.c.
env_sensor_hw.h	Contains user defined data types, macros for LED and buzzer routine parameters and prototypes of externally referred functions defined in env_sensor_hw.c.
gap_conn_params.h	Contains macro definitions for fast/slow advertising, preferred connection parameters, idle connection timeout values etc.
gap_service.h	Contains prototypes of the externally referred functions defined in gap_service.c.
gap_uuids.h	Contains macros for UUID values of the GAP Service and related characteristics.
gatt_service.h	Contains prototypes of the externally referred functions defined in gatt_service.c.
gatt_service_uuids.h	Contains macros for UUID values for the GATT Service.
gyroscope.h	Contains prototypes of externally referred functions defined in gyroscope.c.

File Name	Purpose
gyroscope_emulator.h	Contains prototypes of emulator functions defined in gyroscope_emulator.c.
humidity_sensor.h	Contains prototypes of externally referred functions defined in humidity_sensor.c.
humidity_sensor_emulator.h	Contains prototypes of humidity sensor emulator functions defined in humidity_sensor_emulator.c.
i2c_comms.h	Contains prototypes of I ² C communication functions.
itg3050_gyroscope.h	Contains prototypes of externally referred functions defined in itg3050_gyroscope.c.
magnetometer.h	Contains prototypes of externally referred functions defined in magnetometer.c.
magnetometer_emulator.h	Contains prototype of externally referred magnetometer emulator functions defined in magnetometer_emulator.c.
nvm_access.h	Contains prototypes of externally referred NVM read/write functions defined in nvm_access.c.
pressure_sensor.h	Contains prototypes of externally referred functions defined in pressure_sensor.c.
pressure_sensor_emulator.h	Contains prototypes of externally referred functions defined in pressure_sensor_emulator.c.
sht21_humidity_sensor.h	Contains prototypes of externally referred functions defined in sht21_humidity_sensor.c.
stts751_temperature_sensor.h	Contains prototypes of externally referred functions defined in stts751_temperature_sensor.c.
t5400_pressure_sensor.h	Contains prototypes of externally referred functions defined in t5400_pressure_sensor.c.
temperature_sensor.h	Contains prototypes of externally referred functions defined in temperature_sensor.c.
temperature_sensor_emulator.h	Contains prototypes of externally referred functions defined in temperature_sensor_emulator.c.
user_config.h	Contains macros for customising the application including selection of development boards, real sensor versus emulation modes etc.

Table 6.3: Header Files [13]

6.3.3. Database Files

The xIDE uses database files, see **Table 6.4**, to generate attribute database for the application. For more information on how to write database files, see the *GATT Database Generator User Guide*, available to registered users at www.csrsupport.com.

- GATT Server Database provides handles, permissions and references to the LE profiles/services
- Link Manager (LM) messages and the its database access routines

File Name	Purpose
app_gatt_db.db	Master database file which includes all service specific database files. This file is imported by the GATT Database Generator.
battery_service_db.db	Contains information related to Battery Service characteristics, along with its descriptors and values. See Table 7.4.1 for more information on Battery Service characteristics.
dev_info_service_db.db	Contains information related to Device Information Service characteristics, along with its descriptors and values. See Table 7.4.2 for Device Information Service characteristics.
env_sensing_service_db.db	Contains information related to Environmental Sensing Service characteristics, along with its descriptors and values. See Table 7.4.3 for Environmental Sensing Service characteristics.
gap_service_db.db	Contains information related to GAP Service characteristics, along with its descriptors and values. See Table 7.4.4 for GAP characteristics.
gatt_service_db.db	Contains information related to GATT Service characteristics, along with its descriptors and values. See Table 7.4.5 for more information GATT Service characteristics.

Table 6.4: Database Files

6.4. Customizing the Embedded Application

Any customization required at compile time or run time should be pre-determined during the design phase. All compile time customizations should be made in the [user_config.h](#) file, while the run-time configurations are done using the [env_sensor_csr101x_A05.keyr](#) file.

Initialization of the keys should be performed during application initialization and propagated to the appropriate modules whenever required. The developer can easily customize the application by modifying the parameter values in these files.

In sections 6.4.1 – 6.4.2, there are some critical file settings that need to be changed to ensure that the starter development kit is working properly and is uniquely identifiable. These three variables need to be modified:

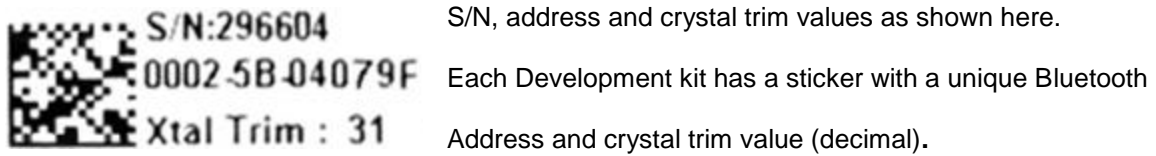
- **Bluetooth Address – Section 6.4.1**
 - Each Bluetooth device must have a unique MAC address, this Address is a programmable Bluetooth/MAC Address, provided by the CSR. This section allows for one to update / modify the Address from the given generic address given with the reference application.
- **Crystal Trim – Section 6.4.1**
 - Bluetooth Low Energy (BLE) is a timing-sensitive technology and requires *crystal trimming*. The trimming values are programmable and are loaded during power-on or reset event. This trimming provides the best performance with a stable frequency.
 - A simple calibration process is done during production, the main frequency calibration up to 1ppm @ 2.4GHz. This calibrated trim value is then marked on the label of the Starter Development Kit. This section walks-through the process to modify the file trim values that came with the CSR Starter Development Kit.
- **Device Name – Section 6.4.2**
 - The reference firmware application provides a default local name that is transmitted during the connection / pairing event, and is seen publically. This section gives the option to change the local name as seen by others.

Additional application customizations can be implemented in later sections 6.4.3 – 6.4.6.

6.4.1. Configuring the Store Key File

The `env_sensor_csr101x_A05.keyr` file defines the Configuration Store Key values for the specific type of target hardware, the CSR1010 Module on-board the CSR Starter Development kit. This file loads into memory during run-time to configure the device's MAC Address and crystal trimming calibrated values. These Configuration Store settings are downloaded to the target CSR1010 Module during the application deployment activities, see **Section 6.5.3** for more details on code deployment.

It is important to have the Bluetooth MAC Address and crystal trim settings in the Configuration Store Key file match the settings on the Starter Development Kit label. Locate the label on the outside of the box that the Starter Development Kit came in. The label will have the



In the CSR uEnergy xIDE, click on the [env_sensor_csr101x_A05.keyr](#) to open the file in the editor, as shown here in **Figure 6.4.1**. The values should be adjusted in the editor to match the values on the label. Click on the 'Save' icon in the menu to make the changes permanent.

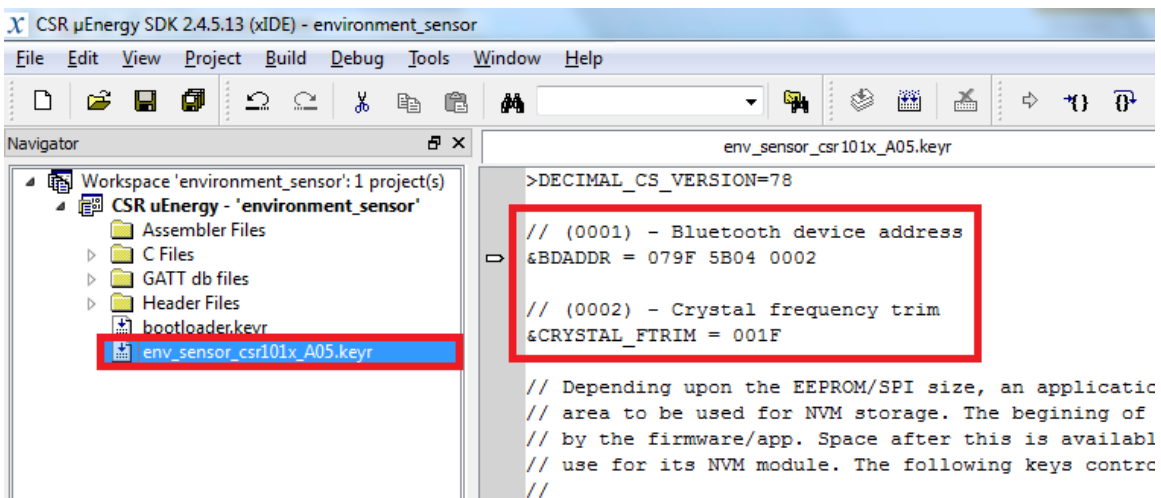
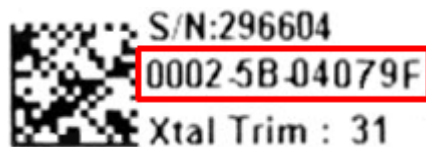


Figure 6.4.1 – Configuration Store Key File

Update Bluetooth MAC Address

Determining the Address; this example shows the syntax of how to transfer the Bluetooth address from the sticker label on the development kit into the .keyr file.

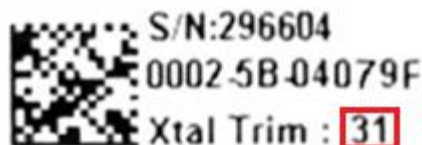
Note the endian-ness of the Bluetooth address in the .keyr file: **&BDADDR = 079F 5B04 0002**



Update Crystal Trim Setting

Determining the crystal trim value; this example shows the syntax of how to transfer the crystal trim value from the sticker label. Note that the value on sticker is in decimal, but is in

hexadecimal in the .keyr file: **31 = 0x1F**
&CRYSTAL_FTRIM = 001F



6.4.2. Configuring the Device Name

By default, all reference design applications provided have the ‘Device Name’ set to the name of the reference design application. The device name for this project is set by default to ‘CSR Env Sensor’ in the `gap_service.c` file on line 58. The maximum length of the device name is 20 octets.

In the CSR uEnergy xIDE, click on the `gap_service.c` to open the file in the editor, as shown here in **Figure 6.4.2**. The values should be adjusted in the editor to match the values on the label. Click on the ‘Save’ Icon in the menu to make the changes permanent.

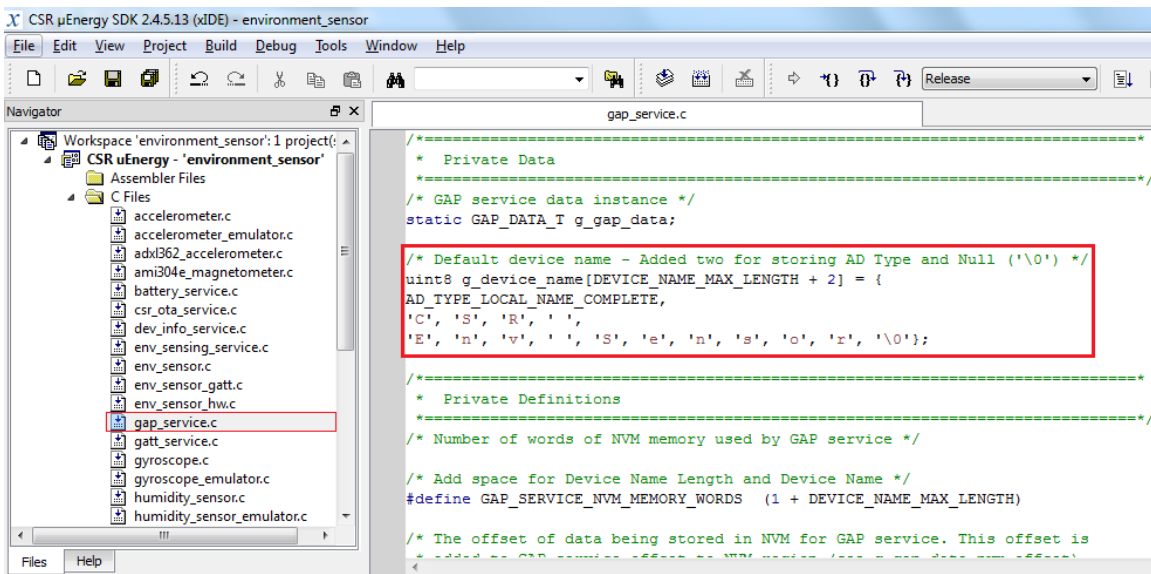
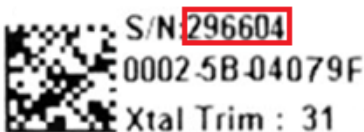


Figure 6.4.2 – Configuring the Device Name

When the end-user wants to connect and pair with a Bluetooth device, the letters in single quotes, on line 58 (`gap_service.c`), form the ‘Local Name’ that will be broadcasted. Here the device would be read on a Bluetooth capable device, CSR Env Sensor’ (default configuration). If one were to change the name of the device, as seen by the public, these characters will have to be modified.

For example, the S/N could be chosen as a unique designator as shown here. Replace the word “Sensor” with the Serial Number from the board label.



```
uint8 g_device_name[DEVICE_NAME_MAX_LENGTH+2] = {
AD_TYPE_LOCAL_NAME_COMPLETE,
'C', 'S', 'R', ' ',
'E', 'N', 'V', '2', '9', '6', '6', '0', '4', '\0'};
```

6.4.3. Advertising

The GAP allows for the application to establish the interactions between two devices and configure the different modes of operation; such as, establish secure a connection, broadcast data and perform other fundamental operations like advertising.

In advertising mode, the BLE device periodically transmits advertising information. Advertising intervals can be set in a range of 20 ms to 10 s. It specifies the interval between consecutive advertising packets. Advertising is done sequentially on the three BLE available channels.

Advertising Parameters

The application uses the parameters in **Table 6.5** for fast and slow advertisements. The macros for these values are defined in file `gap_conn_params.h`. These values have been chosen by considering the overall current consumption of the device. See *Bluetooth Core Specification Version 4.1* [15] for advertising parameter range.

Parameter Name	Slow Advertisements	Fast Advertisements
Minimum Advertising Interval	1280 ms	60 ms
Maximum Advertising Interval	1280 ms	60 ms

Table 6.5: Advertising Parameters [13]

Advertisement Timers

There are two pre-set **values** for **GAP** settings (**fast** and **slow** modes). The application enters the appropriate state on expiry of the advertisement timers. See **Section 7.3** for more information. The macros for these timer values are defined in file `user_config.h`.

Timer Name	Timer Values
Fast Bonded Advertisement Timer Value	10 s
Fast Advertisement Timer Value	30 s
Slow Advertisement Timer Value	60 s

Table 6.6: Advertisement Timers [13]

6.4.4. Connection Parameter Update

The application requests the peer Collector to update connection parameters according to its power requirements.

The application requests a connection parameter update as per the recommendations in *Bluetooth Core Specification Version 4.1 [Vol. 3]* [15], Part C Section 9.3.9, or 30 seconds after the peer device changes the connection parameters. See **Figure 6.4.4** regarding the Connection Parameter Update Procedure. Upon connection establishment with the Collector, the following procedure is used to send a connection parameter update request:

1. Upon connection, the 5s $TGAP_{(conn_pause_peripheral)}$ timer is started.
2. Upon the expiry of $TGAP_{(conn_pause_peripheral)}$, the 1s $TGAP_{(conn_pause_central)}$ timer is started.
3. During this 1s $TGAP_{(conn_pause_central)}$ period, if the application receives a GATT_ACCESS_IND LM event, the timer will be deleted and re-created. The receipt of this event means that the service discovery procedure is in progress and application should not request a connection parameter update.
4. Upon the expiry of $TGAP_{(conn_pause_peripheral)}$, a connection parameter update request will be sent from the application.

The peer Collector may or may not accept the requested parameters. If it rejects the new requested parameters, the application again requests an update after 30 seconds. The macro for this time value is defined in file [app_gatt.h](#) and can be modified as required. If a user wants to use a value less than 30 seconds, it has to be communicated to firmware by calling `LsSetTgapConnParamTimeout` with the appropriate value. See *Bluetooth Core Specification Version 4.1 [Vol. 3]* [15], Part C Section 9.3.9. After two failed attempts, the application tries to update with the connection parameters another two times.

Note: This procedure requires all the characteristics to be declared as FLAG_IRQ in the GATT database, `gap_service_db.db`. This has been done so that the application receives an event GATT_ACCESS_IND for all the reads/writes requested by the Collector device.

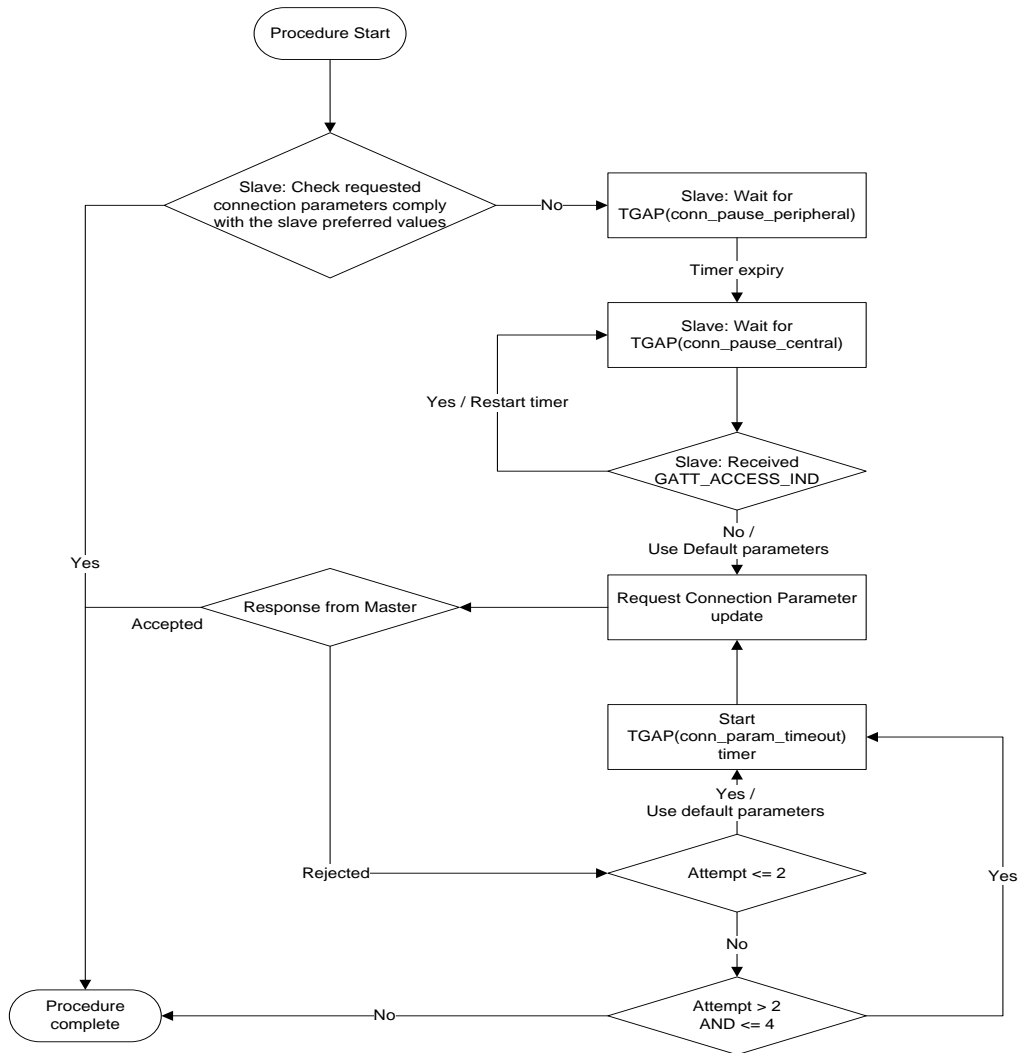


Figure 6.4.4: Connection Parameter Update Procedure [13]

6.4.5. Non-Volatile Memory

The application can use one of the following macros to store and retrieve persistent data in either the EEPROM or Flash-based memory, depending on the type of memory designed to interface with the CSR1010 Module. This particular reference design uses EEPROM.

- NVM_TYPE_EEPROM for I²C EEPROM
- NVM_TYPE_FLASH for SPI Flash

Note: The macros are enabled by selecting the NVM type using the Project Properties in xIDE. This macro is defined during compilation to let the application know which NVM type it is being built for. If EEPROM is selected NVM_TYPE_EEPROM will be defined and for SPI Flash the macro NVM_TYPE_FLASH will be defined.

6.4.6. Environment Sensors – Macro Defined

Macros are defined in [user_config.h](#) to allow individual sensors to be enabled (or disabled by commenting out) during the application image build time. At least one sensor must be enabled. See **Figure 5.3** for Environment Sensor Board Jumper Setting.

Temperature Sensor

Enabled by the `INCLUDE_TEMPERATURE_SENSING` macro. The application currently implements a communication interface for the STMicroelectronics STTS751 sensor. The I²C interface has been used for communicating with the sensor.

Note: The temperature sensor jumper on the Environmental Sensor board H13229V2 must also be fitted. The macro `TEMPERATURE_SENSOR_EMULATION` enables the sensor to be emulated.

Humidity Sensor

Enabled by the `INCLUDE_HUMIDITY_SENSING` macro. The application currently implements a communication interface for the Sensirion SHT21 sensor. The I²C interface has been used for communicating with the sensor.

Note: The humidity sensor jumper on the Environmental Sensor board H13229V2 must also be fitted. The macro `HUMIDITY_SENSOR_EMULATION` enables the sensor to be emulated.

Pressure Sensor

Enabled by the `INCLUDE_PRESSURE_SENSING` macro. The application currently implements a communication interface for the Saw Components T5400 sensor. The I²C interface has been used for communicating with the sensor.

Note: The pressure sensor jumper on the Environmental Sensor board H13229V2 must also be fitted. The macro `PRESSURE_SENSOR_EMULATION` enables the sensor to be emulated.

Angular Rate Sensor

Enabled by the `INCLUDE_GYROSCOPE_SENSING` macro. The application currently implements a communication interface for the InvenSense ITG3050 sensor. The I²C interface has been used for communicating with the sensor.

Note: The gyroscope sensor jumper on the Environmental Sensor board H13229V2 must also be fitted. The macro `GYROSCOPE_EMULATION` enables the sensor to be emulated.

Accelerometer Sensor

Enabled by the `INCLUDE_ACCELEROMETER_SENSING` macro. The application currently implements a communication interface for the Analog Devices ADXL362 sensor. The SPI interface has been used for communicating with the sensor.

Note: The accelerometer sensor jumper on the Environmental Sensor board H13229V2 must also be fitted. The macro `ACCELEROMETER_EMULATION` enables the sensor to be emulated. When using the CSR μ Energy Starter Kit board, the **SPI ENABLE** jumper closest to the CSR1010 module and chip select jumper (**CSB**) must be removed immediately after programming. See **Section 6.5.3** for further jumper details.

Magnetometer Sensor

Enabled by the `INCLUDE_MAGNETOMETER_SENSING` macro. The application currently implements a communication interface for the Aichi Steel AMI304E sensor. The I²C interface has been used for communicating with the sensor.

Note: The magnetometer sensor jumper on the Environmental Sensor board H13229V2 must also be fitted. The macro `MAGNETOMETER_EMULATION` enables the sensor to be emulated.

6.5. Compiling and Deploying the Application

By following the previous instructions, the Environment Sensor Application workspace should have been loaded into the xIDE project and customized to the designer's specifications; it is now time to compile and deploy the firmware application to the CSR1010 Module hardware. In the following three sections will give instructions on how to connect the hardware, compile the project and then deploy the firmware application to the hardware. After these processes have been completed, the hardware should be able to operate remotely (under battery power) and be ready to communicate with the Android Smartphone.

6.5.1. Connecting the Hardware to the PC

If not already done so, open the box of the CSR μ Energy Development Kit and remove the board from the anti-static packaging. Below in **Figure 6.5.1** should be the contents of the box.



Figure 6.5.1: Content of the CSR μ Energy Development Kit [11]

Before the CSR μ Energy Development Kit can be attached to the PC make sure that the jumpers are populated and configured in the default positions as the image shown in **Figure 6.5.2**, which is necessary for the deployment (programming) of the CSR1010 Module EEPROM external memory device. These jumper configurations are necessary for the steps in **Sections 6.5.3**. For further information on connecting the development hardware to the PC, refer to the CS-308421-UG CSR μ Energy Starter Dev Kit Quick Start Guide that is provided on the CD_ROM.

It is now safe to connect the CSR μ Energy CSR1010 Development (Target) board to the PC using the USB cable provided in Development Kit, as shown in **Figure 6.5.2**.



Figure 6.5.2: Development Kit with USB cable [11]

Upon connecting the USB cable between the Target board and the PC, the device drivers will automatically be installed on the PC as seen in **Figure 6.5.3**. These drivers were installed during the installation of the CSR μ Energy SDK application.

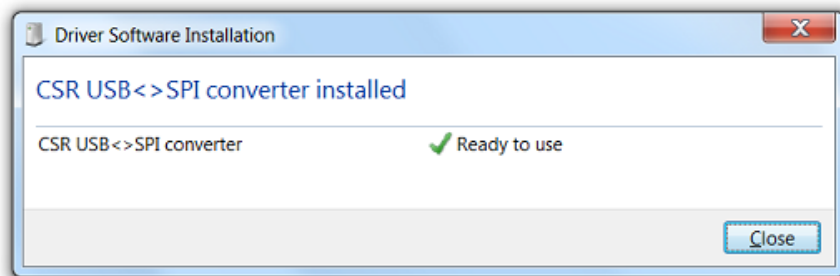


Figure 6.5.3: Pop-up Window Verifying Driver Installation

6.5.2. Building the Application

This section goes into the details of how to compile the project. The xIDE is used for developing application software to run on CSR μ Energy chips and is a program that transforms source code written in C programming language into a binary computer language (the target language), also known as object code. This process is referred to as 'building' or 'compiling' the code.

The xIDE's build process combines the application code, GATT database code and optional 8051 PIO Controller assembly code contained within the project, together with the firmware library to create the binary image for the CSR1010 Module's external EEPROM. Below in **Figure 6.5.4**, is a flow diagram of the xIDE build process.

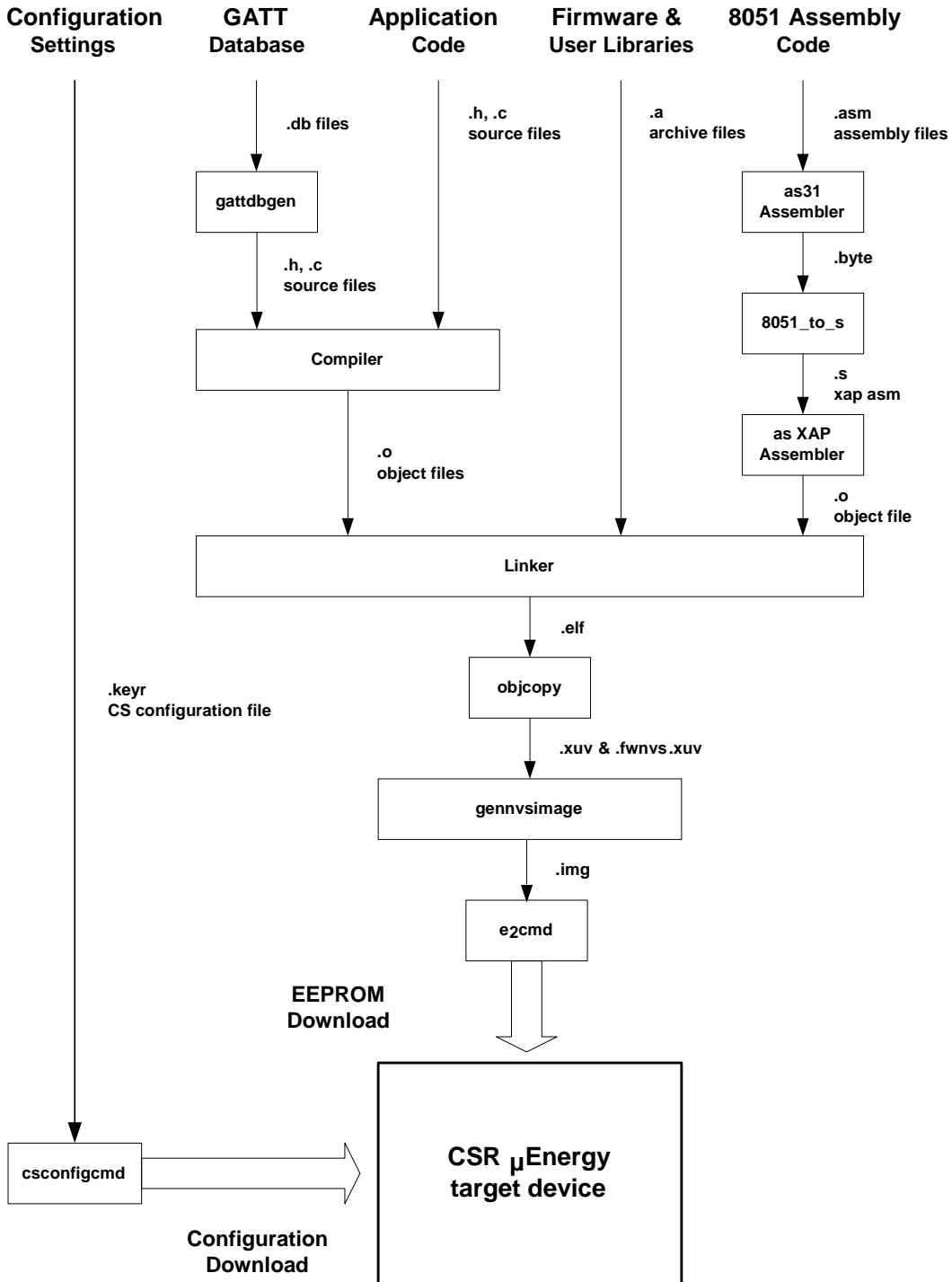


Figure 6.5.4: xIDE Build Process [12]

Here in, **Figure 6.5.5**, describes how to begin building the active project Environment Sensor Application Project using the xIDE program. To begin the build, select the **Build Active Project** from the xIDE **Build** menu or press the **F7** key.

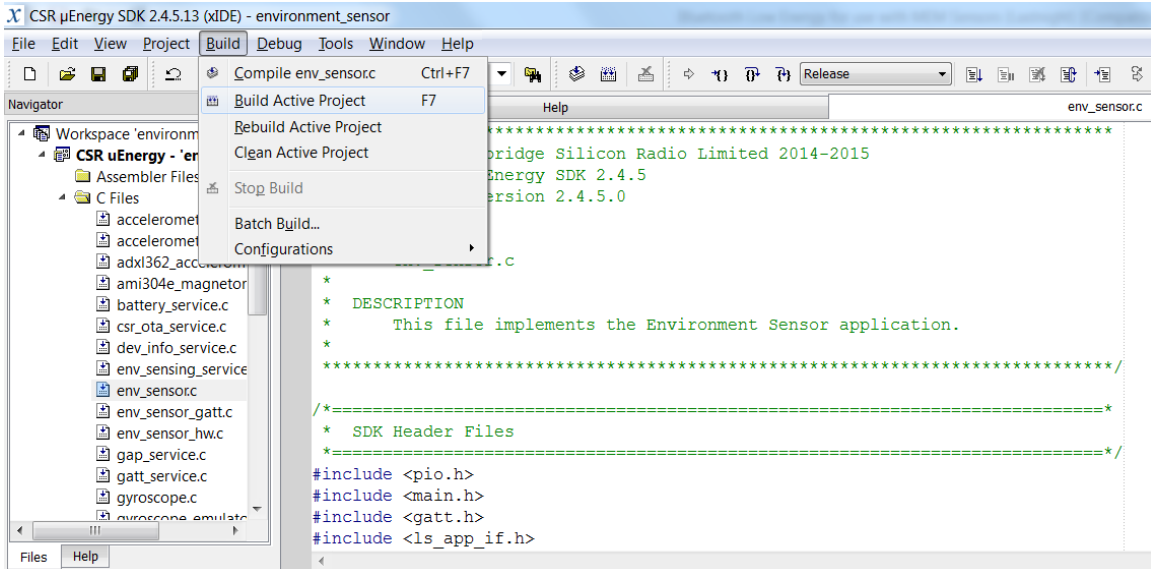


Figure 6.5.5: Selecting Build Active Project

This builds all the files included in the Active project from the current Workspace using the selected configuration into a binary image for the Target board.

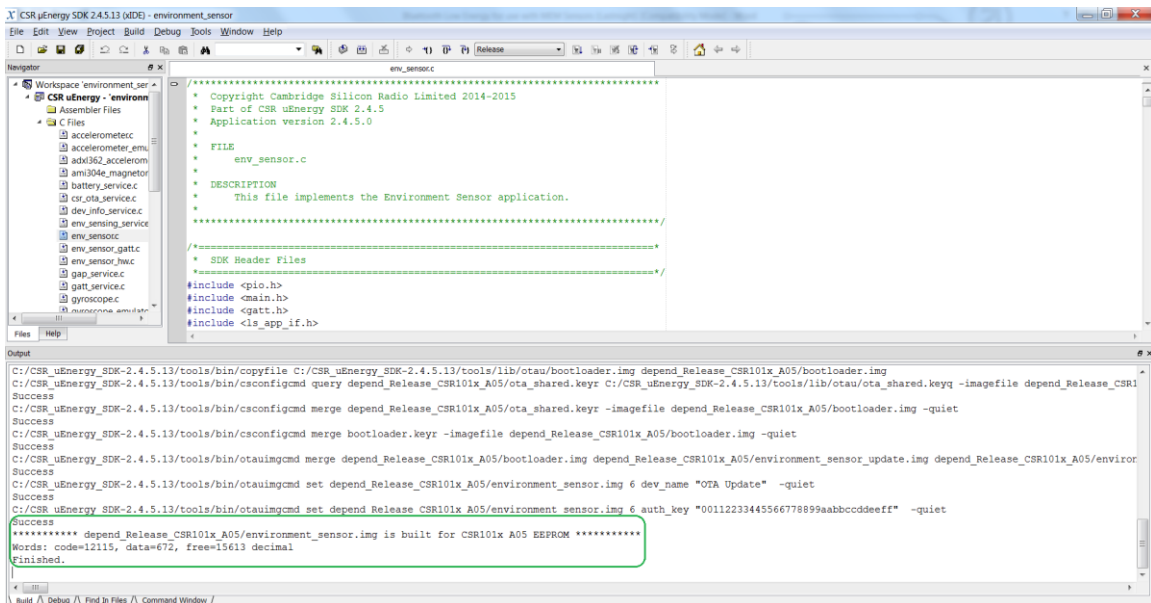


Figure 6.5.6: Build Output Window

If the compilation was successful or an error occurred, it would be displayed here in the Build Output window, highlighted in a green box as shown in **Figure 6.5.6**.

6.5.3. Deploying the Application

In the previous section the *CSR μEnergy xIDE* was used to build the Environment Sensor application. This section describes how the Environment Sensor application (binary image) is transferred (downloaded) to the CSR1010 Module board through the CSR1010 CSR μEnergy Starter Development board. This embeds the Application into the EEPROM (RAM) for later retrieval. See the *CSR μEnergy xIDE User Guide* for details steps on coding and programming: C:\CSR_uEnergy_SDK-2.4.5.13\doc\support\docs\CS-212742-UG_CSRuEnergyXIDEUserGuide.

Here is what needs to be done to deploy the Environment Sensor App to the device: With the project already compiled with the xIDE, select **Run** from the **Debug** menu or press the **F5** shortcut key to begin the deployment process; as seen in **Figure 6.5.7**. The xIDE Software platform will program the hardware by downloading the firmware application as described in this project, otherwise known as deployment.

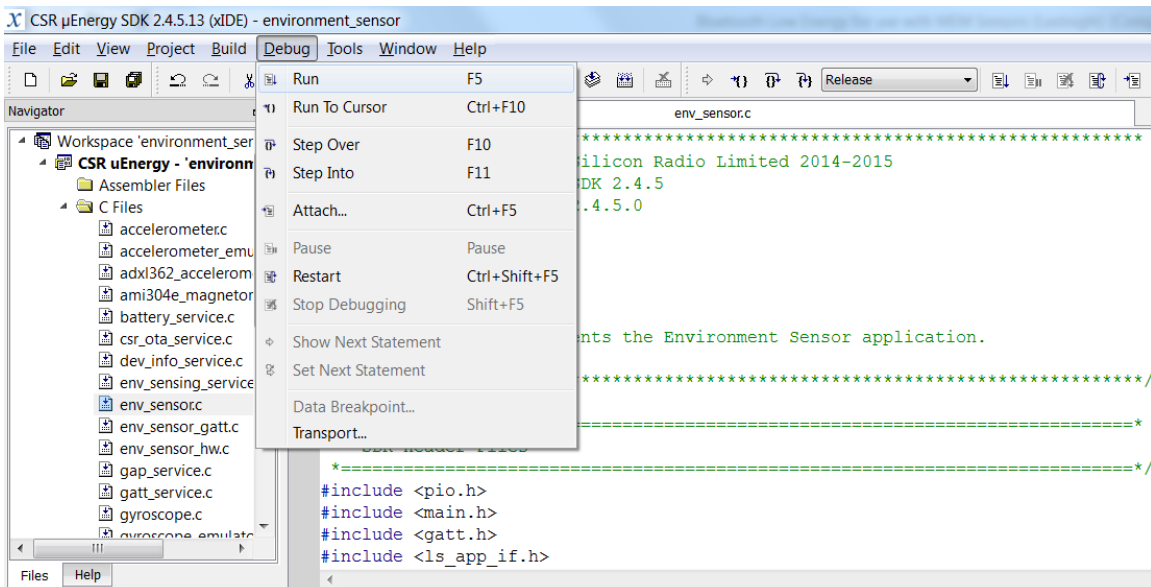


Figure 6.5.7: Selecting 'Run' to Begin Deployment

The Target board must remain connected during the Project deployment. Upon a successful programming, a message in the Debug Output Window will print to the screen 'Download Successful', as seen in **Figure 6.5.8**.

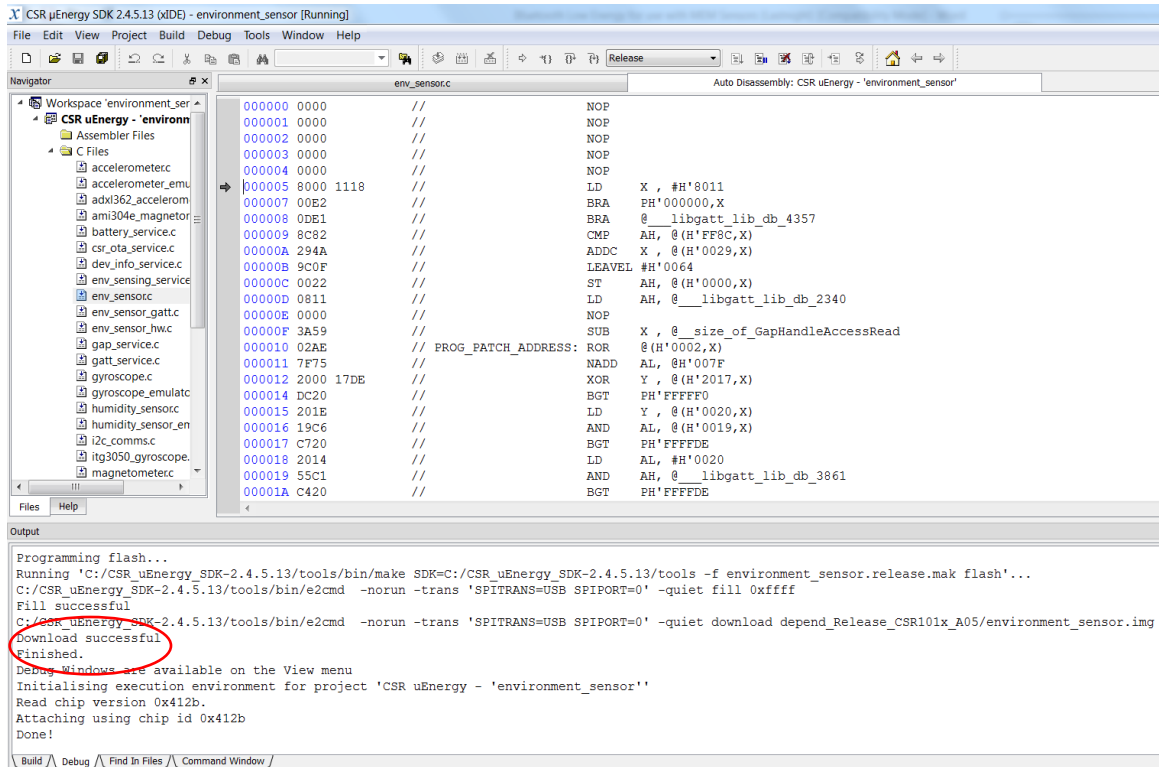


Figure 6.5.8: Project Successfully Deployed to the Hardware

After the EEPROM has been programmed (software downloaded), the xIDE debugger will continue to be running while the Environment Sensor application is running on the hardware, demonstrated by a flashing LED (Light Emitting Diode). When the LED starts flashing, the hardware will begin advertising or broadcasting its Bluetooth presence as available for connecting / pairing, known as discovery mode. The device will stop advertising after a couple of minutes, causing the device to go into sleep mode, in order to conserve battery if configured to operate in battery mode. When this time-out event occurs, the device name will no longer be visible to any searching Bluetooth capable phones. The button (SW1) can be connected up to PIO11 with a jumper (not supplied) to re-enable adverts or repower the PCB by disconnecting / reconnecting the USB cable to restart the CSR1010 Module back into discovery mode.

Modifications to the timer values, as defined in the env_sensor_hw.c file, can be used to control the advertising and LED blink rate.

Note: The LED will stop flashing by exiting the debug mode, either by selecting the ‘Stop Debugging’ from the Debug menu or by pressing the Shift F5 shortcut key, as seen in **Figure 6.5.9**. Alternatively, could exit the xIDE program and unplug the CSR uEnergy Programmer Board from the USB Cable.

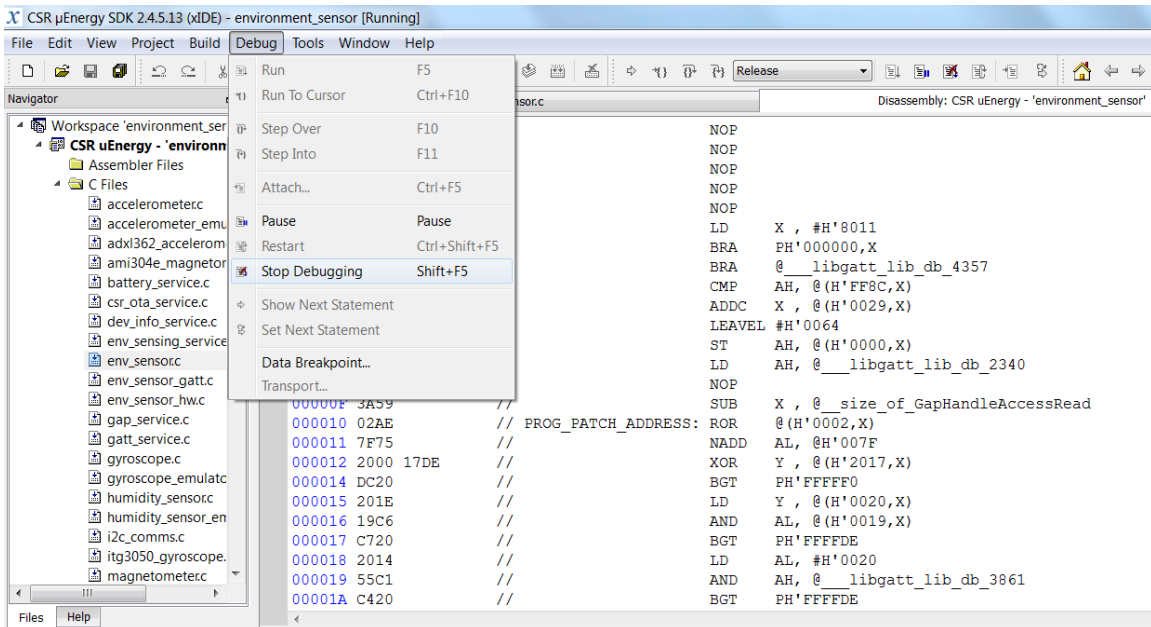


Figure 6.5.9: Stop Debugging

The Firmware has successfully been deployed to the CSR1010 Module’s EEPROM on-board Memory. Now that there is no need to keep the xIDE software running, exit from the xIDE program and disconnect the USB cable from the CSR uEnergy Programming board and the PC.

The CSR uEnergy Programming board must be properly configured, with jumpers, to disable the SPI capable USB Host Controller device from interfering on the SPI bus used by the CSR1010 Module to communicate with the MEMS sensors. Once the Programming board is arranged correctly, the Environment Sensor daughter board can be attached and ready to operate with the CSR1010 Module. The following steps will provide the necessary configurations:

- Insert coin cell battery into the coin cell battery holder.
- Remove the SPI_ENABLE jumpers to disable the USB Host controller chip used for programming / downloading the firmware into the CSR1010 EEPROM.
- Move the power source jumper (J16) from USB mode to Battery mode

As previously listed, if the configuration steps were followed correctly then the CSR uEnergy Programming board should look exactly like **Figure 6.5.10**.

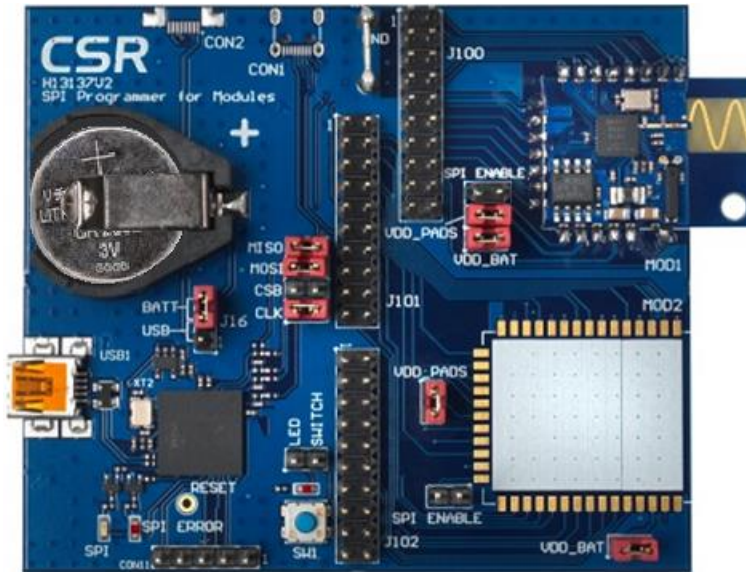
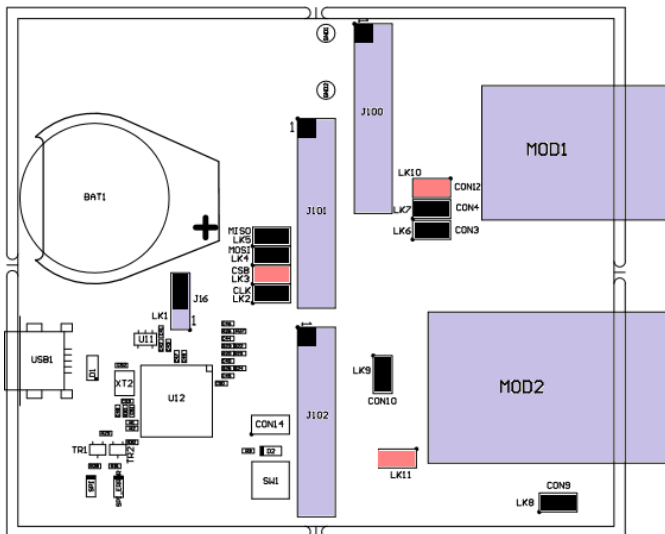


Figure 6.5.10: Programmer Board Arranged for Sensor Board

The Programmer Board Component Descriptor in **Figure 6.5.11** provides the component layout for the CSR uEnergy Programming board. It is displayed here to accompany **Figure 6.5.10** to give insight into the board components and jumper settings needed for various functions.

Programmer Board Component Descriptor



- MOD1 – CSR1010 Module
- MOD2 – Reserved CSR1011 Module
- J100, J101, J102 Headers used to interface with the sensor board
- J16 – option to run board power from USB interface or directly from battery for complete wireless
- LK2, LK10, LK11, remove jumpers after EEPROM is programmed

Figure 6.5.11: Programmer Board Component Configurations

Refer to H13137V3_Component_Layout_VAR0a.pdf for further details.

Connect the H13229V2 – Environmental Sensor board to the H13137V3 Programmer Board as shown in **Figure 6.5.12**. The CSR1010 Module and MEMS Sensor board will now be in free running mode, meaning that all activity is now battery operated and can operate remotely free from the USB tethered power.

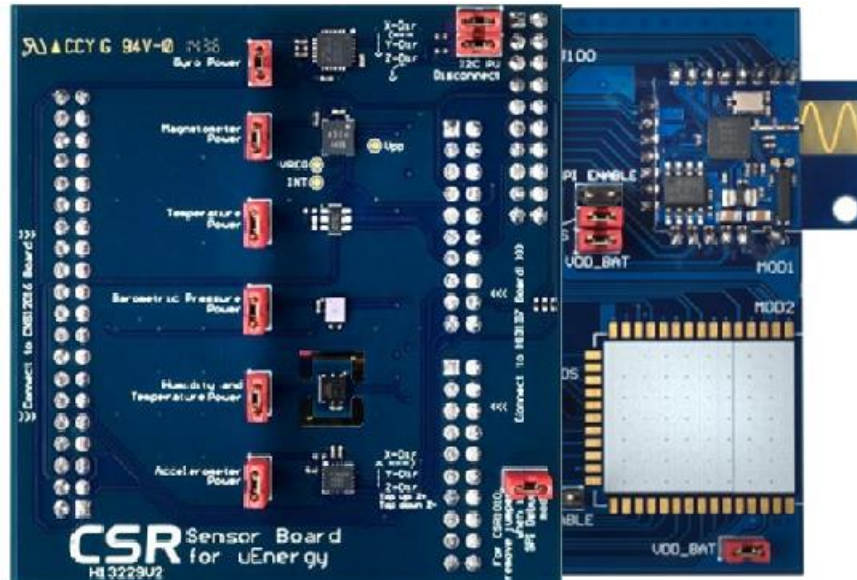


Figure 6.5.12: Final Board Configuration

Upon booting (loading the application into RAM) puts the CSR1010 Module w/sensors into discoverable mode. Discoverable mode is when the device broadcasts its availability. Also, pressing the button on the starter development kit board can reset the device to restart the discovering mode. The environmental data is received by the Environmental Collector part of the Environmental Profile which allows the Android APP to discover the Environmental Sensor and connect/pair with it. This will be discussed later in Section 9 – Android Development Environment.

Note: As stated earlier, the device will stop advertising after a couple of minutes so will not be visible to any searching phones.

After the connection is established, the Android APP can be used to read data from each of the sensors on the board, the user can experiment with the sensors to verify that data is being collected. Full source code for the Environmental Sensor is provided with the CD_ROM that came with the CSR uEnergy Development kit. It is possible to customize the software for further evaluation, or use the reference examples as the basis for further development of a new application to suit one's own needs.

7. System Architecture / Project description

This section describes the Environment Sensor application and provides guidance to developers on how to customize this on-chip application with the CSR1010. The Environment Sensor application is the firmware that is programmed into the EEPROM for later retrieval. Upon power-up (booting) the firmware is loaded into RAM (memory) to be used by the CSR1010 to communicate and retrieve data from the external sensors.

As mentioned earlier, the Environment Sensor App Note was created to accompany the Firmware and can be found in the same directory as Environment Sensor application source code. This chapter references this App Note since it goes into a little more detail than what is listed here, it covers the state machine, and portions of the firmware, memory usage, event status flags to just name a few.

7.1. Environment Sensor Application Overview

The Firmware or Embedded Software known here as the Environment Sensor Application has two main functions called Sensing Profile and Sensing Service. These requirements are determined by the Bluetooth SIG and must meet their specifications in order to meet the BLE standard compliance.

The Environment Sensor application outlined in this section correlates with the Environmental Sensing profile and Environment Sensing Service requirements as specified by the Bluetooth SIG. See the following sub-section for further details.

7.1.1. Environmental Sensing Profile

The Environmental Sensing Profile is used to enable a data collection device to obtain data from an Environmental Sensor that exposes the Environmental Sensing Service.

The Environment Sensor application supports the Environmental Sensing Profile. For more information about the Environmental Sensing Profile, see *Environmental Sensing Profile Specification version 1.0*. Can be downloaded at <https://www.bluetooth.org/en-us/specification/adopted-specifications>

This profile enables a Collector device to connect, interact and obtain data from an Environment Sensor that exposes the Environmental Sensing Service.

- Supports mandatory features only
 - Only reads supported characteristics
- Includes additional custom CSR characteristics
 - Gyroscope, Accelerometer, Magnetometer data
 - Read the calibration registers and add appropriate calibration writes/updates
 - PD to show the 'direction' from true north (calculated from X,Y,Z) in degrees / min
- Modes supported
 - Automatically read data at regular intervals
 - Configuring from UI (100ms, 1,5,10,30,60s)

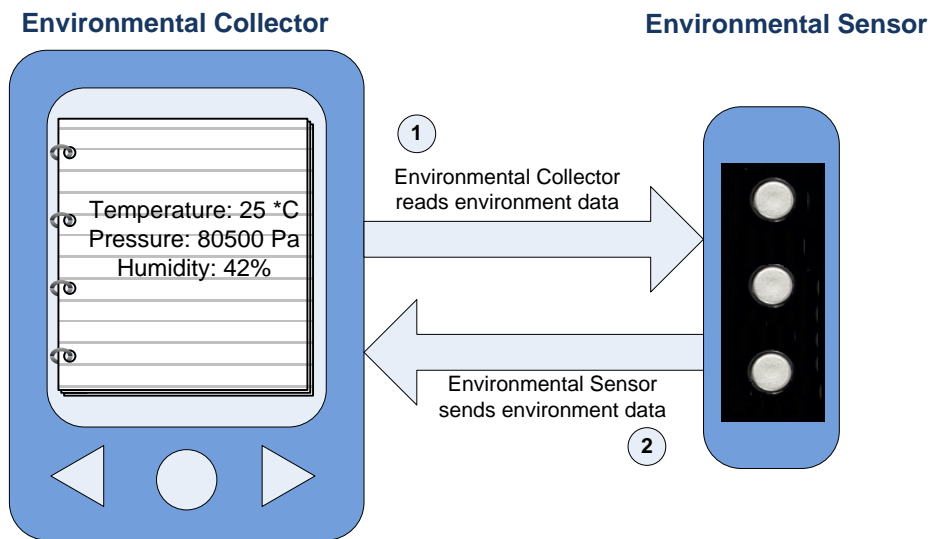


Figure 7.1: Environmental Sensing Profile [13]

The Environmental Sensing Profile defines two roles, described in **Table 7.1**.

Role	Description
Environmental Sensor	Environmental Sensor is a device that exposes the Environmental Sensing Service.
Environmental Collector	Environmental Collector is a device that receives environment data from an Environmental Sensor.

Table 7.1: Environmental Sensing Profile Roles [13]

7.1.2. Application Topology

The Environment Sensor application implements the Environmental Sensing Profile in Environmental Sensor role, see **Table 7.2**; the responsibilities are described in **Table 7.3**.

Role	Environmental Sensing Profile	GAP Service	GATT Service	Device Information Service	Battery Service
GATT Role	GATT Server	GATT Server	GATT Server	GATT Server	GATT Server
GAP Role	Peripheral	Peripheral	Peripheral	Peripheral	Peripheral

Table 7.2: Application Topology [13]

Role	Responsibility
GATT Server	It accepts incoming commands and requests from the client and sends responses, indications and notifications to the client.
GAP Peripheral	It accepts connection request from the remote device and acts as a slave in the connection.

Table 7.3: Responsibilities [13]

For more information about GATT server and GAP peripheral, see *Bluetooth Core Specification Version 4.1 [15]*.

7.1.3. Environmental Sensing Services

The Environmental Sensing Profile mandates or requires only the Environmental Sensing Service to be present. The GAP and GATT are mandated or required to be present by *Bluetooth Core Specification Version 4.1 [15]*. The *Device Information* and *Battery Information* are both optional, see **Figure 7.2**.

The CSR Over-the-Air (OTS) Upgrade Application Service allows updating the CSR1010 module (firmware) independently without having to use the CSR μ Energy SDK programming board. This is a new feature that was recently added to the xIDE but was not utilized for this project and will not be discussed moving forward.

Note: If the user is interested in implementing CSR OTA then refer to the CS-304564-AN-4 Adding OTA-Update Support to an Application.pdf describes the changes that must be made to a CSR μ Energy application to add support for CSR Over-the-Air Update (OTAU) functionality.

The application exposes the following services:

- Environmental Sensing v1.0
- Device Information v1.1
- Battery v1.0
- GAP
- GATT
- CSR OTA Upgrade Application Service (not used)

For more information on Environmental Sensing, Device Information and Battery Services, see *Environmental Sensing Service Specification Version 1.0 [16]*, *Device Information Service Specification Version 1.1 [18]* and *Battery Service Specification Version 1.0 [17]*. For more information on GAP and GATT Services, see *Bluetooth Core Specification Version 4.1 [15]*. See [Section 7.4.6](#) for more information on characteristics supported for each service.

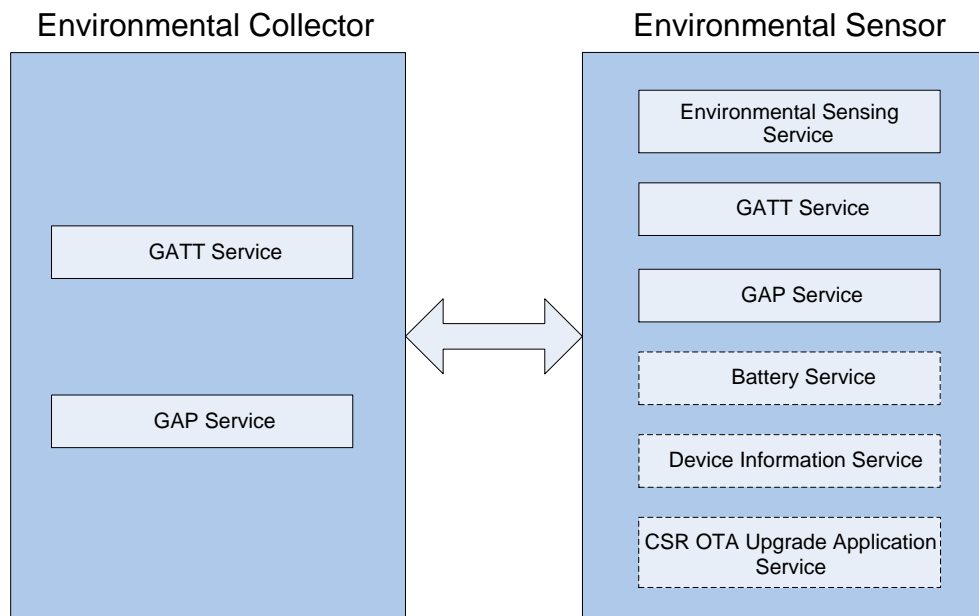


Figure 7.2: Primary Services [13]

This design fulfils the role of the Environmental Sensor in the Environmental Sensor Profile (Ref Bluetooth SIG - ESS_V1.0.0.pdf) and allows environmental data to be exposed.

7.2. Firmware Code Overview

This section is meant to be a quick code overview for functionality only. It placed here to give a little more depth to the previous section 7.1 without having to dissect each file line by line.

The behaviour of the Firmware (Embedded) Application described in this section adheres to the BLE Profile/Service Specifications standard from Bluetooth SIG, as mentioned in the previous section. The Environmental Sensor Application was designed to meet these Specific Compliances:

- Environmental Sensing Profile (v1.0) – Sensing role
- Environmental Sensing Service (v1.0)
- Battery Service (v1.0)
- Device Information Service (v1.1)

The next several sections in this chapter cover some important functions that provide further customization to the design.

7.2.1. Applnit()

This function is invoked when the application is powered on or the chip resets and performs the following initialization functions [13]:

- Initializes the application timers, application data structures and hardware
- Configures GATT entity for server role
- Resets the white list to remove any filtering of discovered devices. See *Bluetooth Core Specification Version 4.1* [15] for more information on white list
- Configures the NVM manager to use I²C EEPROM or SPI flash
- Initializes all the services
- Reads the persistent store
- Registers the attribute database with the firmware

7.2.2. AppProcessLmEvent()

This function is invoked whenever a Link Manager (LM)-specific event is received by the system. The following events are being handled in this function [13]:

7.2.2.1. Database Access

- GATT_ADD_DB_CFM: This confirmation event marks the completion of database registration with the firmware. On receiving this event, the application starts advertisements.
- GATT_ACCESS_IND: This indication event is received when the remote Collector tries to access an ATT characteristic managed by the application.

7.2.2.2. LS Events

- LS_CONNECTION_PARAM_UPDATE_CFM: This confirmation event is received in response to the connection parameter update request by the application. The connection parameter update request from the application triggers L2CAP connection parameter update signaling procedure. See *Bluetooth Core Specification Version 4.1* [15] for more information on the procedure.
- LS_CONNECTION_PARAM_UPDATE_IND: This indication event is received when the remote central device updates the connection parameters. On receiving this event, the application validates the new connection parameters against the preferred connection parameters set and triggers a connection parameter update request if the new connection parameters do not comply with the preferred connection parameters set.

7.2.2.3. SMP Events

- SM_KEYS_IND: This indication event is received on completion of the bonding procedure. It contains keys and security information used on a connection that has completed the short term key generation. The application stores the received diversifier (DIV) and Identity Resolving Key (IRK), if the collector device uses resolvable random address, to NVM. See *Bluetooth Core Specification Version 4.1* [15] more information on keys.

- **SM_SIMPLE_PAIRING_COMPLETE_IND:** This indication event indicates that the pairing has completed successfully or otherwise. In the case of a successful completion of the pairing procedure, the sensor application is bonded with the collector and bonding information is stored in the NVM. The bonded device address will be added to the white list, if it is not a resolvable random address.
- **SM_DIV_APPROVE_IND:** This indication event is received when the remote connected device re-encrypts the link or triggers encryption at the time of reconnection. The firmware sends the diversifier in this event and waits for the application to approve or reject the encryption. The application shall reject the encryption if the bond has been removed by the user or diversifier does not match with the diversifier that is stored during the bonding procedure.
- **SM_PAIRING_AUTH_IND:** This indication is received when the remote connected device initiates pairing. The application can either accept or reject the pairing request from the peer device. The Environment Sensor application does not mandate bonding and accepts the pairing request from the peer device.

7.2.2.4. Connection Events

- **GATT_CONNECT_CFM:** This confirmation indicates that the connection procedure has completed. If it has completed successfully, the application moves to connected state. If the directed advertisements have timed out the application start fast advertisements otherwise it moves to idle state and waits for user activity.
- **GATT_CANCEL_CONNECT_CFM:** This confirmation event confirms the cancellation of connection procedure. When the application stops advertisements to change advertising parameters or to save power, this signal confirms the successful stopping of advertisements by the application.

- LM_EV_CONNECTION_COMPLETE: This event is received when the connection with the peer device is considered to be complete and includes the new connection parameters.
- LM_EV_DISCONNECT_COMPLETE: This event is received on link disconnection. Disconnection could be due to link loss, locally triggered or triggered by the remote connected device.
- LM_EV_ENCRYPTION_CHANGE: This event indicates a change in the link encryption state.
- LM_EV_CONNECTION_UPDATE: This event indicates that the connection parameters have been updated to a new set of values and is generated when the connection parameter update procedure is initiated by either the master or the slave. These new values are stored by the application for comparison against the preferred connection parameters set, see [Section 7.3.4](#).

7.2.3. [AppProcessSystemEvent\(\)](#)

This function handles the system events such as a low battery notification or a PIO change. It currently handles two system events [13]:

- sys_event_battery_low: This event is received whenever the battery voltage crosses the threshold battery voltage. If connected and notifications are configured, the application notifies the battery level to the collector device.
- sys_event_pio_changed: This event indicates a change in PIO value. The application configures events on certain PIOs. If the PIO value changes and the application receives a PIO change event then it will take the appropriate action.

7.3. Internal State Machine

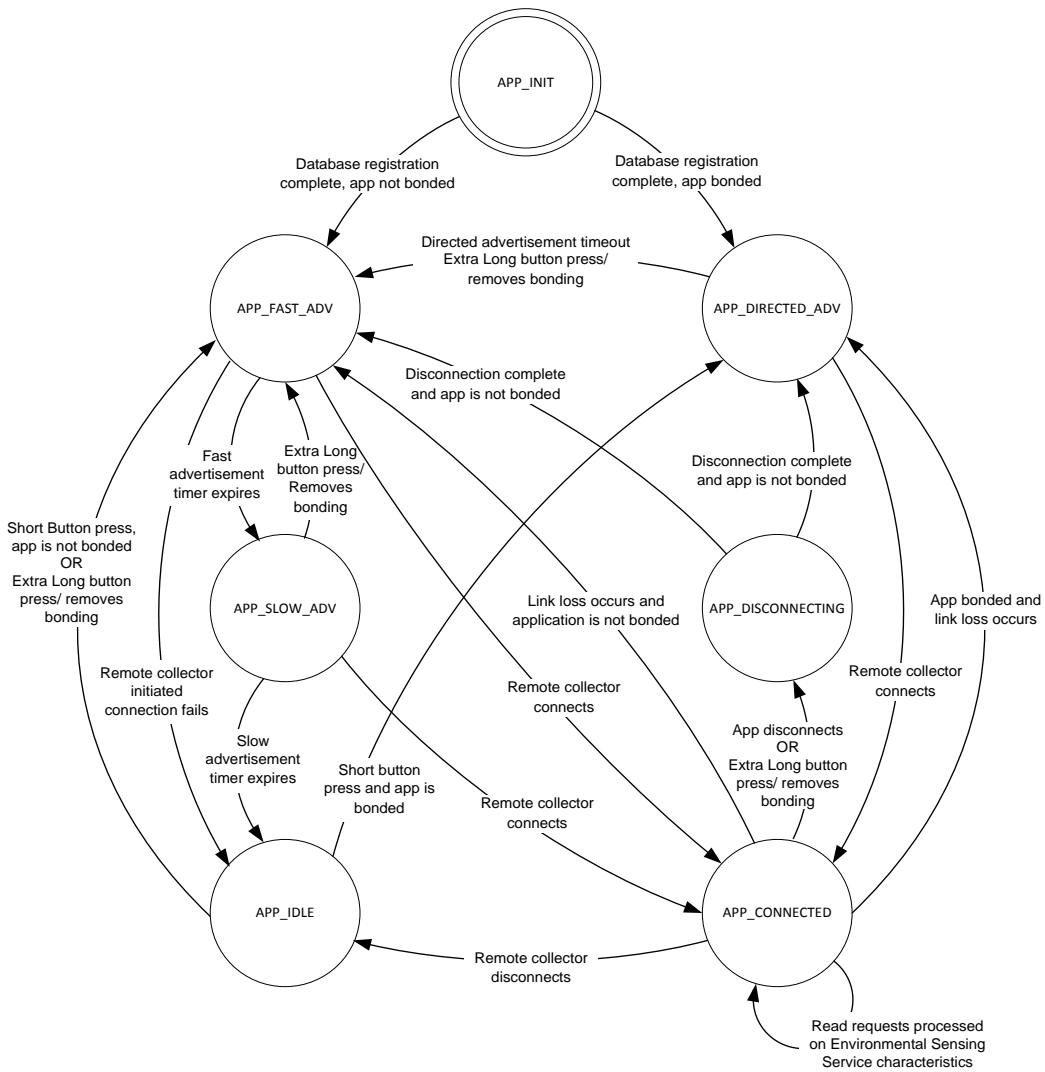


Figure 7.3: Environment Sensor (state transition) Diagram [13]

The application has five internal states including three advertising sub-states as described in sections 7.3.1 to 0 [13].

7.3.1. APP_INIT

When the application is powered on or the chip resets, it enters this APP_INIT state, the application initializes GATT server functionality and other various modules while in this state. It registers GATT database with the firmware and waits for the database registration confirmation.

7.3.2. APP_IDLE

In this state, the application is not connected to any Collector device and responds to only button presses.

- On a *short* button press, less than 2 seconds, the application triggers advertisements and enters the APP_ADVERTISING state.
- A *long* button press, greater than or equal to 2 seconds and less than 4 seconds, is handled in a similar way to *short* button press.
- On an *extra-long* button press, greater than or equal to 4 seconds, the application removes bonding information, clears the white list and enters the APP_ADVERTISING state.

7.3.3. APP_ADVERTISING

The application enters General Discoverable Mode and beeps twice to indicate the start of advertisements. If speaker is implemented. It is not implemented in this reference design.

- Sub state APP_DIRECTED_ADV: If the application is bonded to a Collector device, advertisements start from this state. In this state, the application sends directed advertisements. If the bonded Collector connects to it, the application stops advertisements and enters the APP_CONNECTED state. If the Directed Advertisement Timeout happens before connection establishment, the application moves to APP_FAST_ADV state.
- Sub state APP_FAST_ADV: The application sends Undirected Connectable advertisements in this state and uses fast advertising parameters. The application behaves differently if it bonded or not.
 - The application is bonded: The application starts a timer of value FAST_BONDED_ADVERT_TIMEOUT_VALUE and uses a white list to filter out the unwanted connection requests. Upon expiry of the timer, it disables the white list and sends advertisements for (FAST_CONNECTION_ADVERT_TIMEOUT_VALUE-FAST_BONDED_ADVERT_TIMEOUT_VALUE) seconds.

- The application is not bonded: The application does not use white list and sends advertisements for FAST_CONNECTION_ADVERT_TIMEOUT_VALUE seconds.
- If a Collector connects to it, the application stops advertisements and enters the APP_CONNECTED state. If the fast advertisement timer expires before connection establishment, the APP_SLOW_ADV sub state is entered. See section 0 for more information on advertisement timer value.
- Sub state APP_SLOW_ADV: The application sends Undirected Connectable advertisements in this state and uses slow advertising parameters. If a remote device connects to it, the application stops advertisements and enters the APP_CONNECTED state. If the slow advertising timer expires before connection establishment, the APP_IDLE state is entered.
- An *extra-long* button press in any of the advertisement states deletes the bonding information and starts fast advertisements in APP_FAST_ADV state. Removal of pairing is indicated by three beeps, if speaker is implemented.
- A *short* button press or a *long* button press does not have any effect in this state.

Note: See Volume 3, Part C, section 9 of *Bluetooth Core Specification Version 4.1* [15] for more information on General Discoverable modes, Directed and Undirected Connectable advertisements.

7.3.4. APP_CONNECTED

The application enters this state on connection establishment with a Collector device. The application processes all the Read Requests received in this state. A *short* button press is indicated by a *short* beep (100 ms) in this state and a *long* button press is indicated by a *long* beep (500 ms). The application does not perform any action on a *short* button press or a *long* button press. An *extra-long* button press in this state removes the pairing and disconnects the link.

Note: A speaker is not installed nor configured to work in this design. Therefore no beep can be audible. The beep feature is not used, but mentioned here for completeness.

7.3.5. APP_DISCONNECTING

The application enters this state upon initiating a disconnection. The application waits for a disconnect confirmation for the disconnection initiated by it. When it receives the disconnect confirmation, it checks if it is bonded to any Collector.

- If the application is bonded to a Collector, it starts directed advertisements and enters APP_DIRECTED_ADV state.
- If the application is not bonded, it starts fast advertisements and enters the APP_FAST_ADV state.

On an *extra-long* button press, the application removes the bonding information and starts fast advertisements in APP_FAST_ADV state.

7.4. Service Characteristics Database

Characteristics are managed by either the firmware or the application. The characteristics managed by the application have flags set to FLAG_IRQ in the corresponding database files.

When the remote connected device accesses that characteristic, the application receives a GATT_ACCESS_IND LM event that is handled in the AppProcessLmEvent() function defined in the env_sensor.c file. See **Section 7.2.2** for more information on the handling of the GATT_ACCESS_IND LM event. For more information on flags, see the *GATT Database Generator User Guide*, available to registered users at www.csrsupport.com.

7.4.1. Battery Service Database

For information on the Battery Service, see *Battery Service Specification Version 1.0* [17]. For information on Security permissions, see *Bluetooth Core Specification Version 4.1* [15].

Characteristic Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
Battery Level	0x0034	Read, Notify	Application	Security Mode 1 and Security Level 2	Current battery level
Battery Level-Client Configuration Descriptor	0x0035	Read	Application	Security Mode 1 and Security Level 2	Current client configuration for Battery Level characteristic

Table 7.4.1: Battery Service Database [13]

7.4.2. Device Information Service Database

For more information on the Device Information Service, see *Device Information Service Specification Version 1.1* [18]. For more information on Security permissions, see *Bluetooth Core Specification Version 4.1* [15].

Note: All the characteristics have been declared as FLAG_IRQ but they are still being handled by the firmware. This has been done for the connection parameter update procedure. See **Section 6.3.3** for the connection parameter update procedure.

Attribute Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
Serial Number String	0x0018	Read	Firmware	Security Mode 1 and Security Level 2	BLE-ENV-001
Hardware Revision String	0x001a	Read	Firmware	Security Mode 1 and Security Level 2	<Chip Identifier>
Firmware Revision String	0x001c	Read	Firmware	Security Mode 1 and Security Level 2	<SDK version>
Software Revision String	0x001e	Read	Firmware	Security Mode 1 and Security Level 2	<Application Version>
Manufacturer Name String	0x0020	Read	Firmware	Security Mode 1 and Security Level 2	Cambridge Silicon Radio
PnP ID	0x0022	Read	Firmware	Security Mode 1 and Security Level 2	Vendor Id source is BT Vendor Id is 0x000a Product Id is 0x014c Product Version is 1.0.0

Table 7.4.2: Device Information Service Database [13]

7.4.3. Environmental Sensing Service Database

For more information on the Environmental Sensing Service, see *Environmental Sensing Service Specification Version 1.0* [16]. For more information on Security permissions, see *Bluetooth Core Specification Version 4.1* [15].

Note: Acceleration and Angular Rate are CSR custom defined characteristics for the Environmental Sensing Service. See **Section 7.4.6** for more information on these characteristics.

Attribute Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
Temperature	0x0025	Read	Application	Security Mode 1 and Security Level 2	<Measured Value>
Humidity	0x0027	Read	Application	Security Mode 1 and Security Level 2	<Measured value>
Pressure	0x0029	Read	Application	Security Mode 1 and Security Level 2	<Measured value>
Acceleration	0x002b	Read	Application	Security Mode 1 and Security Level 2	<Measured Value>
Angular Rate	0x002d	Read	Application	Security Mode 1 and Security Level 2	<Measured Value>
Magnetic Flux	0x002f	Read	Application	Security Mode 1 and Security Level 2	<Measured Value>
Magnetometer Calibration	0x0031	Write	Application	Security Mode 1 and Security Level 2	1 or 0

Table 7.4.3: Environmental Sensing Service Database [13]

7.4.4. GAP Service Database

For more information on GAP service and security permissions, see *Bluetooth Core Specification Version 4.1* [15].

Attribute Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
Device Name	0x0011	Read, Write	Application	Security Mode 1 and Security Level 2	Device name. Default name: CSR Env Sensor
Appearance	0x0013	Read	Firmware	Security Mode 1 and Security Level 1	General Tag 0x2000
Peripheral preferred connection parameters	0x0015	Read	Firmware	Security Mode 1 and Security Level 1	Min connection interval - 10 ms Max connection interval - 10 ms Slave latency - 100 Connection timeout - 6 s

Table 7.4.4: GAP Service Database [13]

7.4.5. GATT Service Database

For more information on GATT, see *Bluetooth Core Specification Version 4.1* [15].

Attribute Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
Service Changed Characteristic	0x0003	Indicate	Application	Security Mode 1 and Security Level 2	Handle Range
Service Changed Client Characteristic Configuration Descriptor	0x0004	Read, Write	Application	Security Mode 1 and Security Level 2	Current Client Characteristic configuration for Service Changed characteristic

Table 7.4.5: GATT Service Database [13]

7.4.6. Custom Defined Characteristics

The following are CSR custom defined characteristics for the Environmental Sensing service, see **Table 7.4.6**.

Characteristic	UUID	Size	Description
Acceleration	0x0000aaa1d10211e19b2300025b00a5a5	6	Instantaneous acceleration value in units of 0.01 m/s ² . Two octets per axis are being reported in order (X-accel, Y-accel, Z-accel)
Angular Rate	0x0000aaa2d10211e19b2300025b00a5a5	6	Instantaneous angular rate in units of degree/s. Two octets per axis are being reported in order (X-angular Rate, Y-angular Rate, Z-angular Rate)
Magnetometer Calibration	0x0000aaa4d10211e19b2300025b00a5a5	1	Writing a value of 1 will start calibration and a value of 0 will stop calibration of the magnetometer

Table 7.4.6: CSR Custom Characteristics [13]

8. Android Development Environment

This chapter will provide guidance to setup and use the Android Studio Environment for the Android Smartphone. This will be just a top level software discussion to introduce the reader or end-user to the programming environment in order to get started. Keeping the project in perspective, a software code review would go beyond the scope of this document. The review would be too lengthy to have a reasonable discussion in this report. It is left to end-user to become familiarized with the file script structure and function. This chapter describes the procedure for configuring the Android Studio IDE, loading the reference application as a project and then running or deploying that code onto the hardware platform.

For convenience of those interested in duplicating or expanding on the software that is mentioned in this document the programming environment setup instructions have been included in **Section 8.1.2**. It is intended to provide developers with the information required to begin developing applications on the Android OS platform. The reference application source code discussed there could greatly reduce the effort required to develop a final product application and spring board the software engineer toward developing other new products.

8.1. Setting Up the EnvironmentApp Project

From **Section 4.2.2**, the **Java Resource Environment**, **IntelliJ IDE**, and the **Android Studios** (SDK), the files required for the Android development environment to operate were the installed. Before jumping into the Android Studio with the EnvironmentApp project, the Android development environment needs to be set up and configured with the proper version of the SDK Components like the API Tools / Libraries, drivers and other environment settings.

- In **Section 8.1.1** will go into details on how to configure the Android SDK Environment
- In **Section 8.1.2** will go into details on how to configure the project workspace before importing the directory into the Android Studio in **Section 8.2**
- In **Section 8.1.3** will go into details how to prepare the Smartphone Hardware to interface with the PC

8.1.1. Preparing the Android Studio Environment

In order for the Android Studio development environment to be able to operate properly, with the EnvironmentApp reference design, the environment must be in the same configuration that was used with the original EnvironmentApp Project. This requires installing the correct API Tools and other supporting files using the SDK Manager.

- From the Windows **Start** menu, chose the Android Studio folder and then click on the Android Studio to open the development environment.
- The Android Studio Setup Wizard window will appear, select SDK Manager.
- Select to install Android API's and components for versions 19 – 22 only

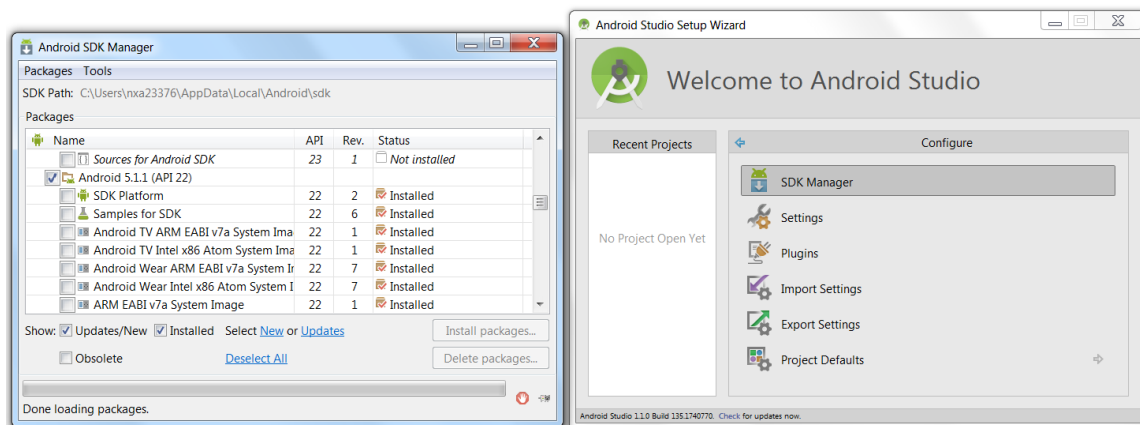


Figure 8.1.1.1: Android Setup Wizard

- The Download Components window will appear, as in **Figure 8.1.2.2**. Continue to follow the prompts until the Wizard has completed. Exit the program and move on to **Section 8.1.2.2** to configure the Environment Workspace directory before importing the source code into the project.

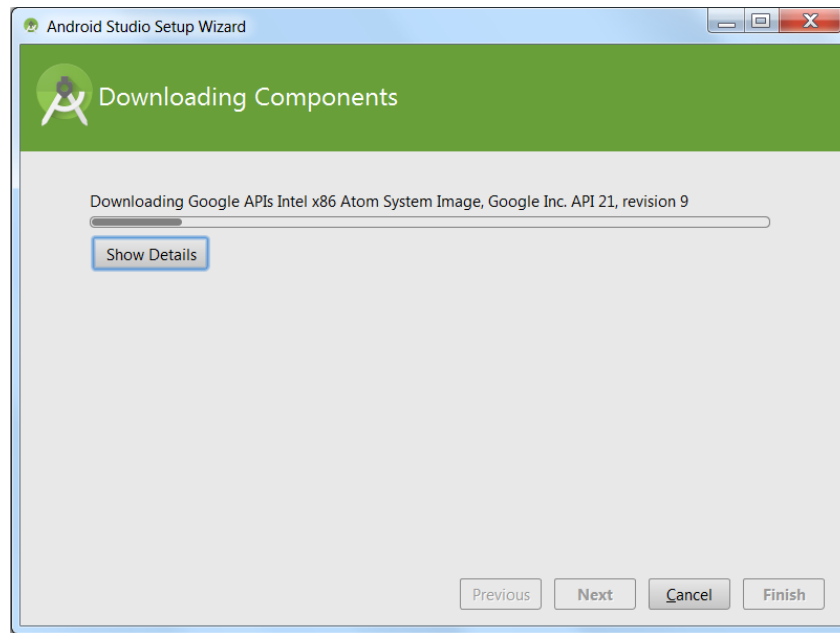


Figure 8.1.2.2: Downloading Components

8.1.2. Configuring the Project Workspace

This section will step through how to configure the workspace file structure. The EnvironmentApp project source files should have been resourced with this document, if so browse through the project folder directory to locate: ...\\Phone App\\Software\\EnvironmentApp

- Copy the EnvironmentApp folder and all its content to a local directory, for example C:\\EnvironmentApp\\
- Be careful, “Your workspace”, could be any folder, however it is really important to keep the directories, ‘EnvironmentApp/app’ and ‘EnvironmentApp/btsmart/’, in the workspace has the same hierarchy.
 - BTSmart Library v1.1 is used for the code development, it is CSR’s Reference library meant to be used will all CSR BLE devices
 - The app directory provides the EnvironmentApp source code

The Workspace should now be ready to be import into the Android Studio. In **Section 8.2**, will cover the procedure for bringing up the Android Studio environment.

If the EnvironmentApp project source files were not resourced with this document, an alternative approach would be to create a new Android Project in the same manner that was done originally for this project. The original EnvironmentApp was developed using CSR reference code (RunningSpeedDemo application) and BTSmart Library. The application was further developed in Android ADK using Eclipse IDE structure. Unfortunately, issues required upgrading to the modern Android Studio with IntelliJ. Information on this issue can be read in **Section 10** of this document. The remainder of this subsection will mention briefly the configuration steps to get started straight from CSR's website.

Since the EnvironmentApp project was developed using the Android Studio with IntelliJ, the project files structure is different. This means that there needs to be some minor tweaks to the source code files before importing the workspace directory into Android Studio.

Note: In retrospect it would have been better to have just started with the basic skeleton source code provided by CSR. There had to be many modifications to RunningSpeedDemo source code that using the skeleton reference source code would have sped up development.

Here are the steps to get started with a new reference design:

- Download the source_package.zip, CSR'S Android design reference applications, from:
<http://www.csrsupport.com/μEnergy/ExampleApplications>
 - This skeleton code can then be extended to create new BT Smart applications, such as the EnvironmentAPP in this project.
 - Along with this skeleton code, the zip file also contains other example CSR μEnergy apps designed to work with the on-chip applications included within the CSR μEnergy SDK 2.x. These other projects are: Heart Rate Collector, Cycling Speed, Running Speed
 - Be sure to delete the BTSmart Library that comes with the zip file. It is no longer useful.

BTSmart Library v1.1 was used for this code development, it has since been updated to BTSmart Library v1.2 with the old library no longer available. The latest reference library can be downloaded from CSR's Website:

https://www.csrsupport.com/download/54696/BtSmartLibrary_UseWithAndroidOTAUAppv1.2.zip

It is left to the reader, using this reference design, to become familiar with the files and the Android development environment. Information presented here is to provide an additional design option.

- Open the ZIP file and extract the folders. Delete the BTSmart Library folder.
- Open the ZIP and extract the BTSmart Library v1.2. Open the directory and copy the btsmart folder to the Reference demo folder. The \btsmart and \app should be in the same folder.
- Edit the RunningSpeedDemo\project.properties file lines 13 -17, reads as **Figure 8.1.2.3:**

```
# Project target.  
target=android-19  
android.library.reference.1=..\BtSmartLibrary
```

Figure 8.1.2.3: File Edit

Now it is time to launch the Android Studio program and import the Reference design as seen in **Figure 8.1.2.4**. Afterwards, proceed to **Section 8.2** to continue with the tutorial.

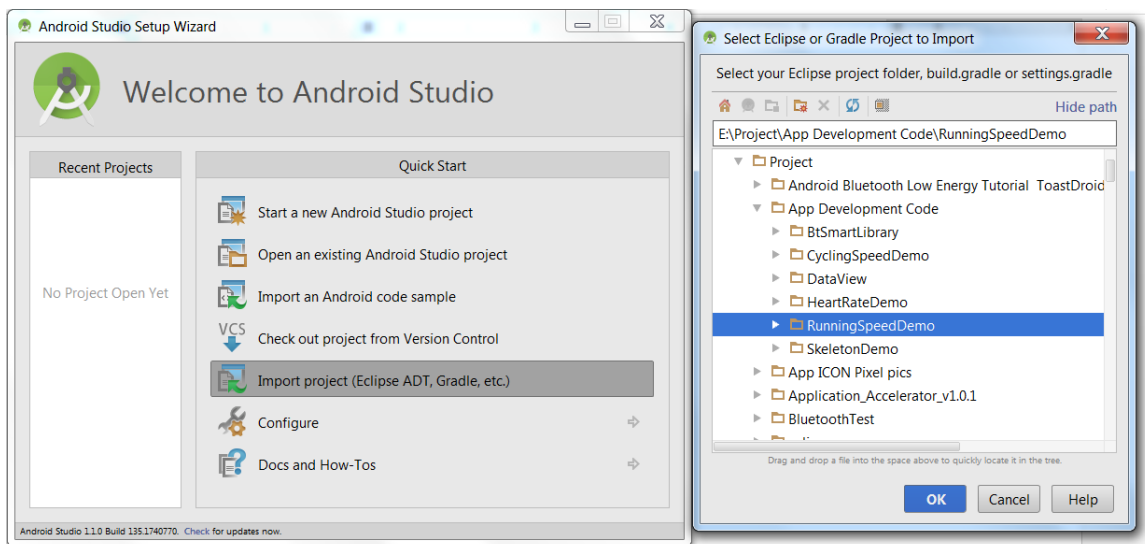


Figure 8.1.2.4: Importing Project

Note: Google also provides a good resource walk-through to get started:

<http://developer.android.com/training/basics/firstapp/creating-project.html>

8.1.3. Connecting the Hardware to the PC

The first thing, before connecting the hardware (Android capable Smartphone) to the PC, is to enable the USB Debugging mode under the Developer options menu on the phone.

The USB Debugging mode allows for the 'App' file to be transferred from the Android Studios IDE to the Smartphone. On Android 4.2 devices and higher, the Developer options screen is hidden by default. This enabled feature is required for the Android Studio IDE to link, communicate and download the application file to the smartphone. For convenience, a quick walk-through on configuring the Android phone is listed here. The same procedure follows for Nexus phones.

1. Go to the 'Settings' menu, it can be located by tapping the Apps folder icon on the main screen of the Phone display; from there select the '**Settings**' Icon.
2. In the '**Settings**' Menu, select the '**More**' tab, then scroll down to the bottom of the screen and tap the '**About device**' at the bottom.
3. Scroll down and tap '**Build number**' seven times until a message saying '**Developer mode has been enabled**'.
4. Press the back button then select the '**More**' tab again. The '**Developer options**', under System Manager, should now have appeared; press it.
5. In the '**Developer options**' menu, click on the box to enable USB debugging. Select okay after the prompt. The Developer settings should now been configured on the Galaxy S4.

Note: A detailed instruction on how to enable USB debugging is available through Samsung website: <http://www.samsung.com/au/support/skp/htg/16385#none>

The second thing, before connecting the hardware (Android capable Smartphone) to the PC, is to download the Samsung Android. The file can be downloaded from here:

<http://developer.samsung.com/technical-doc/view.do?v=T000000117>

Follow the prompts during the install, as seen in **Figure 8.1.3.1**.

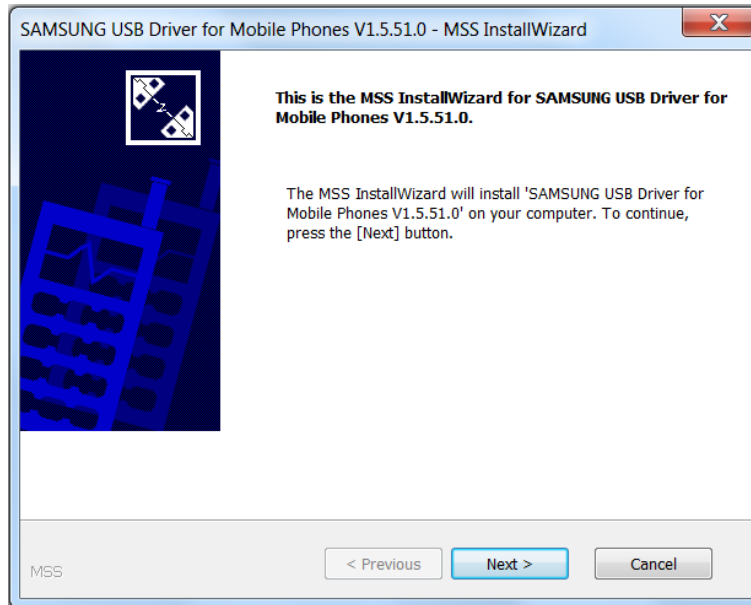


Figure 8.1.3.1: USB Driver for Windows

Once the phone is setup to be in USB Debugging mode and the Samsung USB Drivers for Windows has been installed, it is now the time when the Smartphone can be connected and tethered to the PC through the USB cable. Later, this method will be used by the Android Studios to download the application file to the phone. **Figure 8.1.3.2** is a picture of the hardware.



Figure 8.1.3.2: Configure Phone for Android Development Environment

Connect the Samsung Galaxy S4 or equivalent Smartphone to the PC with a USB cable.

8.2. Launch Android Studio IDE

This section shows how to use Android Studio to build the application envapp.apk file.

Reopen the Android Studio Program from the Start Menu, same as earlier. This time, select Import project (Eclipse ADT, Gradle, etc.). A pop-up window will appear, browse through the Windows directory and select the top level folder to import into the directory; as shown in **Figure 8.2.1.**

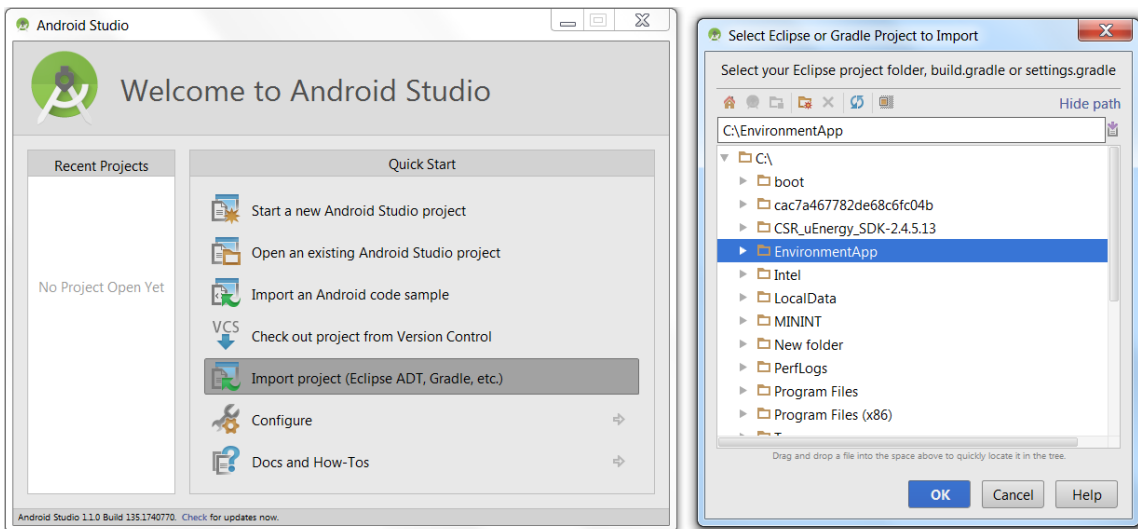


Figure 8.2.1: Importing Project

The Android Studio program will attempt to build the project as it opens the development environment, a window should appear as in **Figure 8.2.2.**

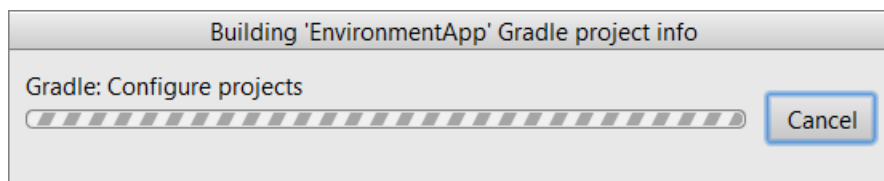


Figure 8.2.2: Building Project

Once the project is on the computer, open the Android Studio and import the project.

- From the Windows **Start** menu, chose the Android Studio folder, and then click on the Android Studio to open the development environment.
- A window should pop open that looks just like **Figure 8.3**.
- From the wizard "Welcome to the Android Studio": select "Import project (Eclipse ADT, Gradle, etc.)".
- From "

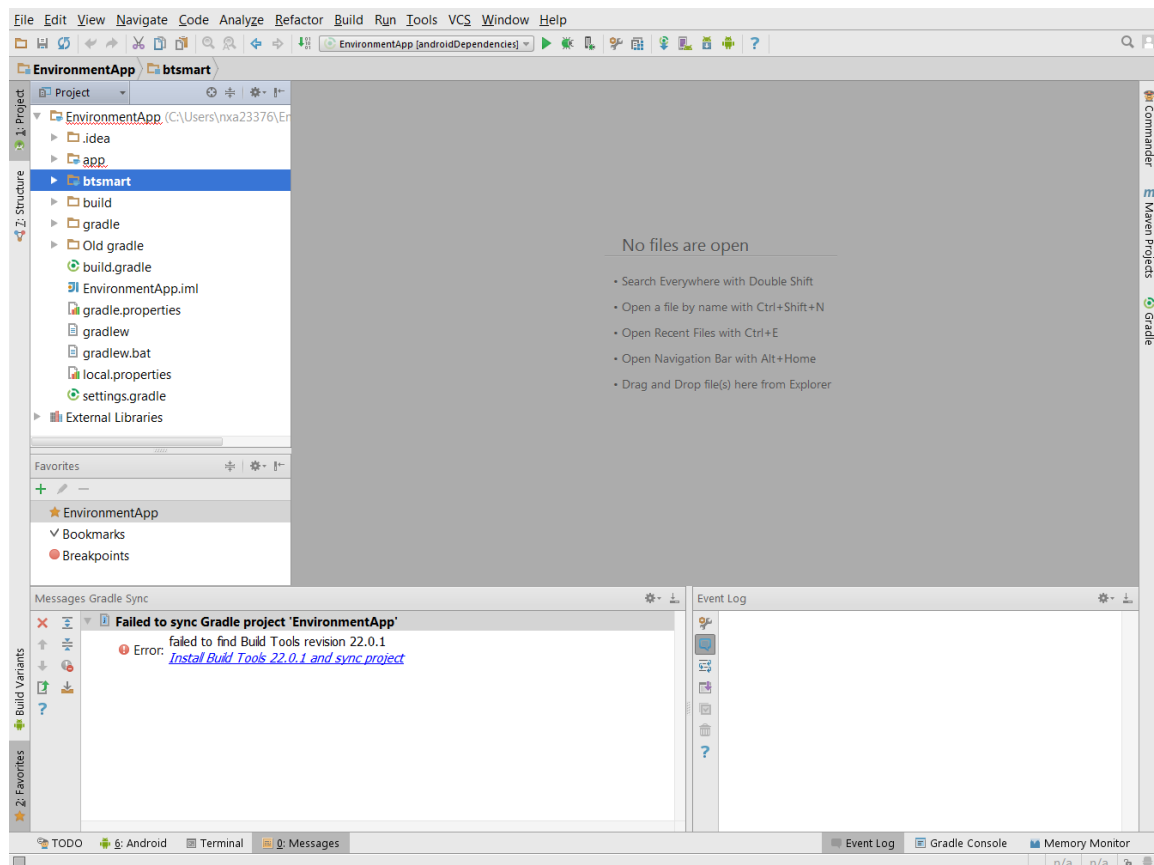



Figure 8.2.3: Install Missing Components

The application is compatible from Android 4.4.2 (API 19) thru to the latest. Run the Android SDK Applications and load the Project demo the Thumb drive location:

//Phone App/Software/EnvironmentApp/app

It is recommended to use the latest version of Android Studio along with the latest SDK from Android – Android 5.1 (API 22). From Android Studio: Tools > Android > SDK Manager or select the icon  on the top toolbar.

Once the project is on the computer, open Android Studio and import the project.

- From the wizard “Welcome to Android Studio”: select “Import project (Eclipse ADT, Gradle, etc.)”.
- From “File > Import Project...”

Then select the top root of the project which is the “EnvironmentApp” folder – the icon could be the same file named “build.gradle”:

After the Smartphone Device has been configured for deployment using the Android Device Monitor in the Android Studio IDE, move onto Section 8.4.1 – Building the Application.

8.3. Application Files

This section provides only list of files that make up the EnvironmentAPP project and should be contained within the workspace on the PC, these files are provided as is. It is up to the Engineer referencing this design to become familiar with the structure and function of the files and how to implement them into other designs.

8.3.1. Java Files

The Java files provide the behind the scenes processes where the action happens. These files work with with the XML files to deliver to the display screen the User interface.

File Name	Purpose
BatteryActivity.java	Provides Battery Data to update on the display, extended from the LinkedActy.java. Parallels BatteryActivity.xml file.
BtSmartLink.java	Creates one connection to the BTSmart device, requests all the available services & characteristics.
BtSmartLinkListener.java	Allows the BTSmart link class to communicate with the different activities which implement this interface.
BtSmartUuid.java	Enumerated type for Bluetooth Smart UUIDs.
CircleImageView.java	Sets the icon that will be shown on the button to scan for available BTSmart devices.
DevicesListAdapter.java	This class extends the ArrayAdapter to the ScannedDevice
EnvironmentActivity.java	This class allows to display the information for characteristics from the Environmental Sensing service. Parallels activity_environment.xml file.
FabButton.java	Sets the progress to indeterminate or not, shows the animation ring
FabUtil.java	Creates the starting angle animator for the circleview
InformationActivity.java	This class displays the Device information, an extension of the LinkActivity.java. Parallels the activity_information.xml file
LinkActivity.java	This class is the BTSmart Library abstract activity to extend this application. This class implements all instances needed on each activity and manages their life cycle depending on what is active.
MainActivity.java	Provides three button action controls, parallels the MainActivity.xml
ProgressRingView.java	States the progress of the progress bar.
Utils.java	Contains all useful methods for this application, like signal strength

Table 8.3.1: App Java Files

8.3.2. XML Files

The XML files handle the graphical user interface (GUI). These files determine the look and feel of the User experience and where to place the data on the screen when the 'App' is running on the Smartphone. See **Table 8.3.2** below for the list of XML file used in this project.

File Name	Purpose
activity_battery.xml	Shows the battery level
activity_connection.xml	Shows a list of connected devices
activity_environment.xml	Provides a window to place the environment sensor data on the display.
activity_information.xml	This class allows to display the information for characteristics
activity_main.xml	Contains the Layout for the FragmentActivity Values
AndroidManifest.xml	Presents essential information about the app to the Android system
Attrs.xml	Defines the attributes throughout the different elements
Colors.xml	Resource that carries a color value
values\dimens.xml	Displays part of the GUI graphics
values-sw600dp\dimens.xml	Sets the values for 600dp size
values-w820dp\dimens.xml	Sets the values for 820dp size
flat_button_background.xml	Gives the graphic for a button
flat_button_text_color.xml	Sets the color for the text on the button
list_devices_item.xml	Creates the emulated phone body size
selector_flat_buttons.xml	Button movement animation
Strings.xml	Used to control what is printed on the device
styles.xml	Specifies properties such as height, padding, font color, font size, background color. Default
tile_battery.xml	Creates the battery backdrop color. Default
tile_device.xml	Creates the device backdrop color. Default
tile_environmental_sensing.xml	Creates the environmental sensing backdrop color. default
widget_fab_button.xml	Used to fabricate buttons

Table 8.3.2: App XML Files

8.3.3. BTSmart Library Files

The BTSmart Library is provided by CSR to be used with the CSR1010 hardware. It handles the communication process and simplifies the overall development effort. **Table 8.3.3.**

File Name	Purpose
ApplicationTest.java	Tests the application using the java script
BluetoothStateBroadcastReceiver.java	Used to inform about Bluetooth connected/disconnected state
BtSmartConnectedActivity.java	Provides the list of available devices
BtSmartConnectionActivity.java	Extends BtSmartConnectionActivity from the BTSmart Library, starts the Bluetooth LE scan and provides the list of available devices.
BtSmartRequest.java	A simple data structure to be used in the request queue.
BtSmartService.java	Service for communication with a Bluetooth Smart device.
CharacteristicHandlersContainer.java	A class to hold handlers that want to receive notifications about characteristics.
ParcelCharacteristic.java	An interface whose characteristics can be written to and stored for parcel services
ParcelService.java	Provides an array that stores information in a parcel
ScannedDevice.java	Data object used to manipulated BTSmart devices found on Scan
ScanResultsActivity.java	Activity used to scan for remote Bluetooth Smart devices, show the results in a list, and perform an action
ServiceRegistrationFailedException.java	Looks for a serialVersionUID and creates a fail exception
UUIDHelper.java	List of available Service names
activity_scan_results.xml	Shows results of the scan
AndroidManifest.xml	Presents essential information about the app to the Android system
colors.xml	Resource that carries a color value
dimens.xml	Sets the dimensions for the GUI graphics
list_row.xml	Used by the activity scan to list the available devices seen by the Bluetooth
scan_button_style.xml	Detects the button style
values\styles.xml	Specify properties such as height, padding, font color, font size, background color. Default

Table 8.3.3: BTSmart Library Files

8.4. Compiling and Deploying the Application

By following the previous instructions, the EnvironmentApp Project should have been loaded into the Android Studio IDE and modified to the designer's specifications (optional). The next step is to compile and deploy the Android application to an Android OS capable Smartphone, such as, a Samsung Galaxy S4 or equivalent. In the following two sections will give instructions on how to configure and compile the project, and then deploy the Android application to the Smartphone through the USB cable. After these processes have been completed, the Android Smartphone should be ready to communicate with the CSR1010 Module with the attached MEMs Sensors board that was developed in **Section 6**.

8.4.1. Rebuilding the Application

In **Section 8.2**, the EnvironmentApp project will automatically be built after it is imported into the Android Studio. If there are no issues or modifications to the project then there is no need to 'Rebuild Project' and can move onto **Section 8.4.2**. If changes were made then select from the Toolbar Menu → Build → Make Project or use shortcut key (ALT B) as seen in **Figure 8.4.1**.

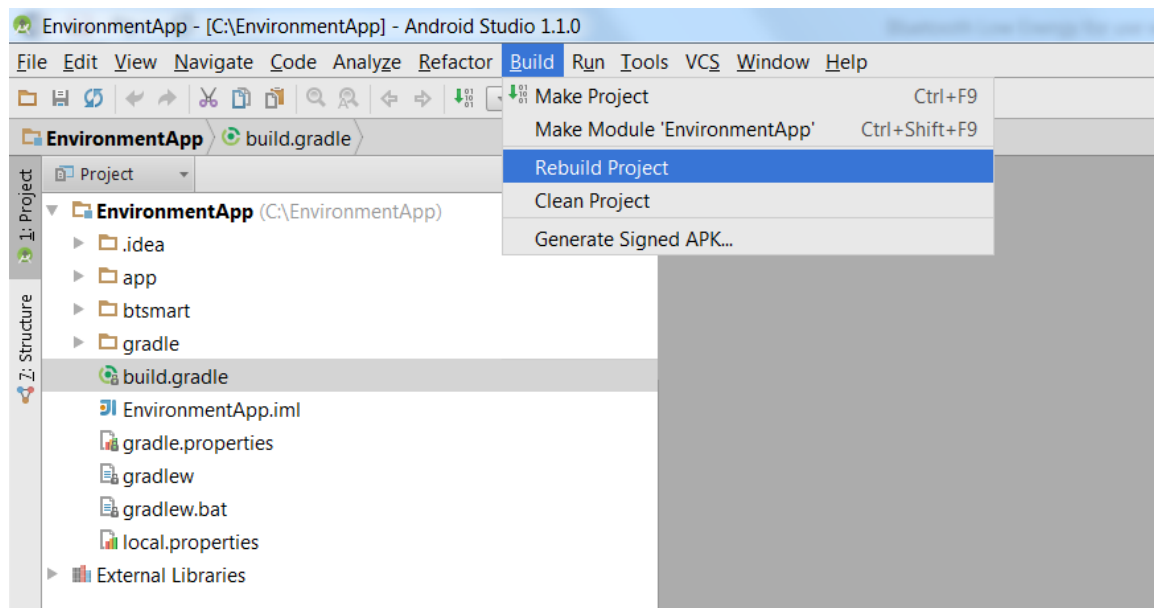


Figure 8.4.1: Building the 'App'

If the compilation was successful or an error occurred, it would be displayed in the 'Event Log, display window at the bottom right location of the Android Studio, as shown in **Figure 8.4.2**.

8.4.2. Deploying the Application

Make sure the Smartphone is running and logged into with the main screen visible.

Select Run → Run 'app' from the toolbar menu or shortcut keys (Shift 10) to begin the downloading process of the envapp.apk file, that was created in **Section 8.2** and again in **Section 8.4.1**, deploying the file to the Android phone.

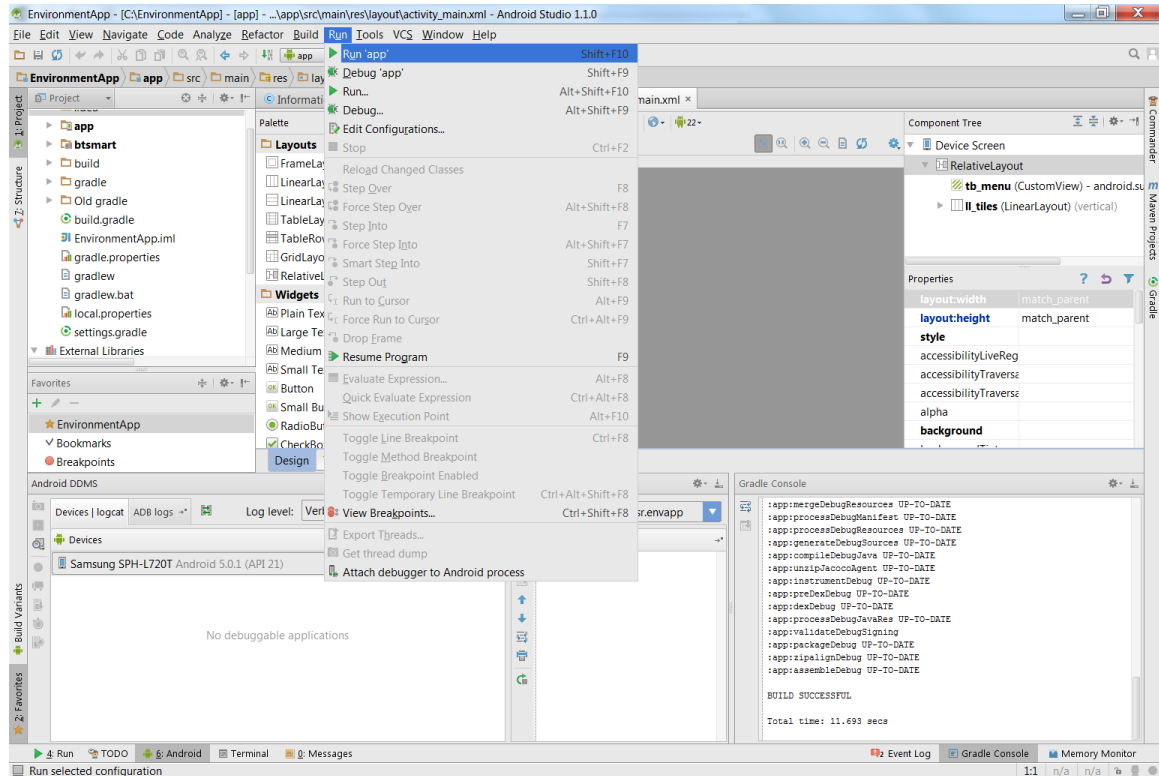


Figure 8.4.2: Running of the 'App'

Android Studios will begin deploying the envapp.apk code to the Smartphone after the Run 'app' was selected. If the procedures from **Section 8.1.2** was successful and the phone name appears in the device window (bottom left in **Figure 8.4.2**) of the Android development environment then a pop-up window, **Figure 8.4.3**, will appear and ask to choose an available device or give an option to just run an emulation of a predefined phone model to run simulations.

Select the available Device and press okay to begin the download.

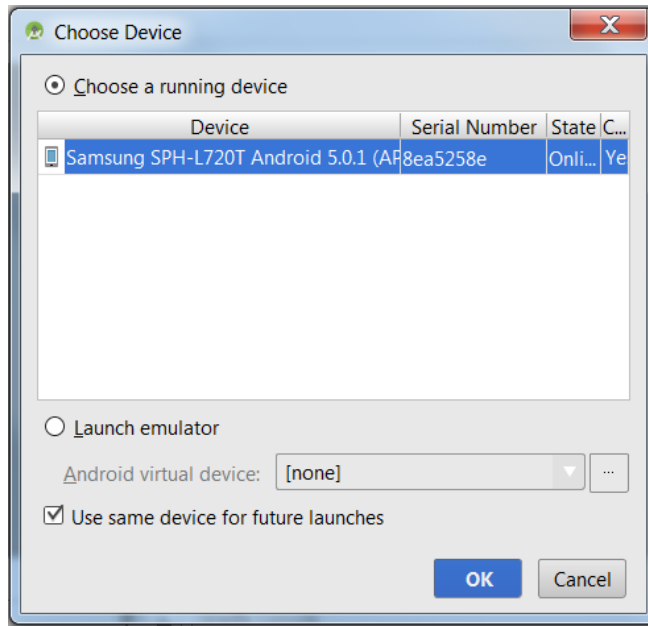


Figure 8.4.3: Choose Device Pop-up Window

If all went well, the window console should say built successful and that the Session 'app' running as in Figure 8.4.4

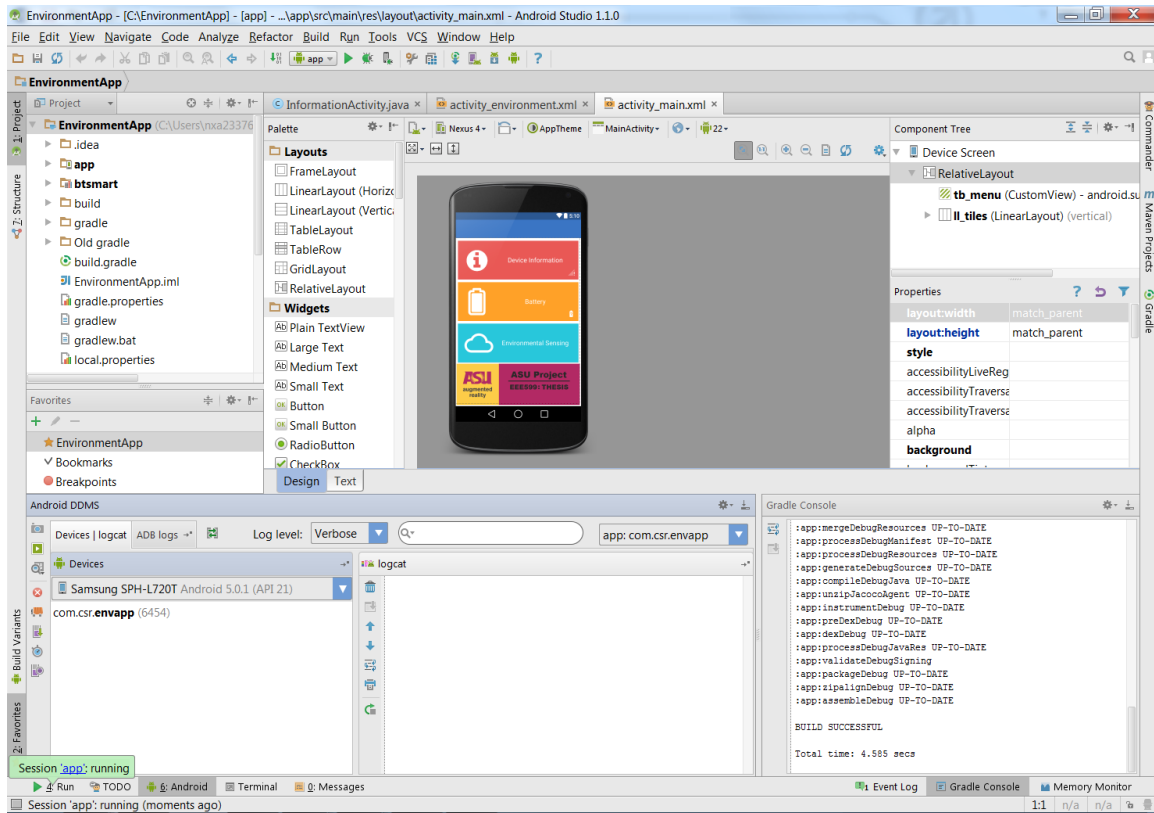


Figure 8.4.4: Session 'app' running

In the previous figure, a green bubble pops up in the lower left corner of the Android Studio saying Session 'app' running. By clicking the '4 Run' tab or pressing (Alt 4) with the keyboard, the running display console window will appear, as seen in this Figure. This will notify if there are any issues while the 'APP' is running on the Smartphone.

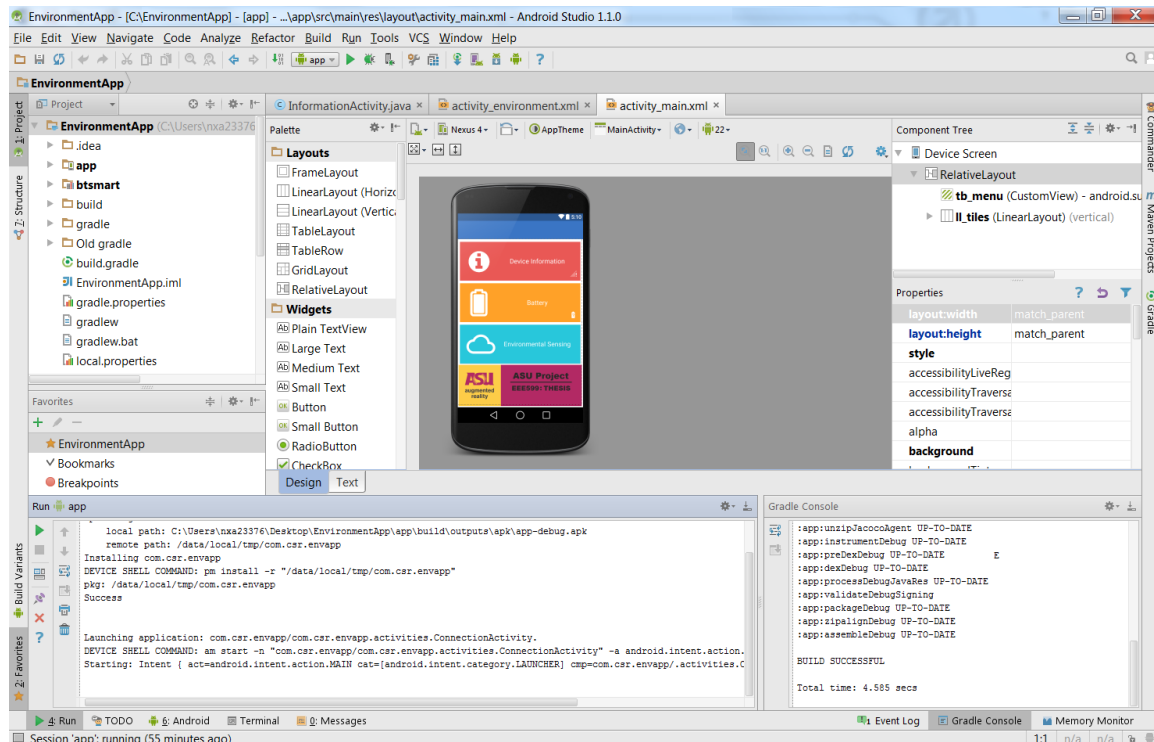


Figure 8.4.5: Running App Verified

The screen on the Smartphone should now be displaying the running application that was just deployed. Power cycle the CSR1010 Development Environment Kit from **Section 6.5.3** to put the device in discovery mode. The Smartphone should see in the Bluetooth available devices 'CSR Env Sensor' or some other device name given in **Section 6.4**.

The results should be the same as the Demonstration seen in **Section 9.2**. The next **Section 9.1** will go through a brief tour of the xml files that make up the Graphical User Interface (GUI) as seen from the Android Studio program.

9. Phone App Overview

This section discusses a top level connection, pairing and data transfer summary of how the Smartphone and hardware interact as demonstrated in the following flowchart. The remaining portions of this section described the Graphical User Interface (GUI) from the software that was deployed on the Android Phone, known as the 'App'.

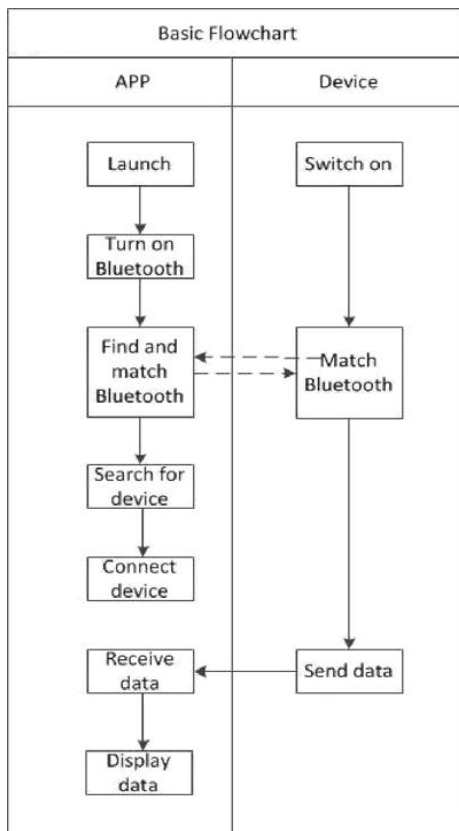


Figure 9.1 Basic Flowchart

Basic Flowchart

Description :

- 1、 Bluetooth in mobile phones and devices must be turned on and matched successfully to be ready for data exchange.
- 2、 Here the "Device" refers to all of CSR's Bluetooth-enabled devices. Specifically for this design, CSR1010 module with the Sensors: Temperature, Humidity, Accelerometer, and etc, but can also be used for other sensors like heart rate monitor, blood pressure meter, blood glucose meter, safety equipment, thermometer, counter and so on.
- 3、 The transmitted data received from the device includes: the device information, battery information, the sensor data and so on.
- 4、 APP can monitor, get and display device parameters and associated measurement data.

The following screenshots are from the emulator provided by the Android SDK. If the code is properly loaded into the Android SDK, with the IntelliJ Platform, then the results should be identical. The following pages are the App display screens as it should look like after the software has been deployed on to the Smartphone. The images and GUI Interfaces are defined in the XML files at this directory location:

//Phone App/Software/EnvironmentApp/app/src/main/res/layout

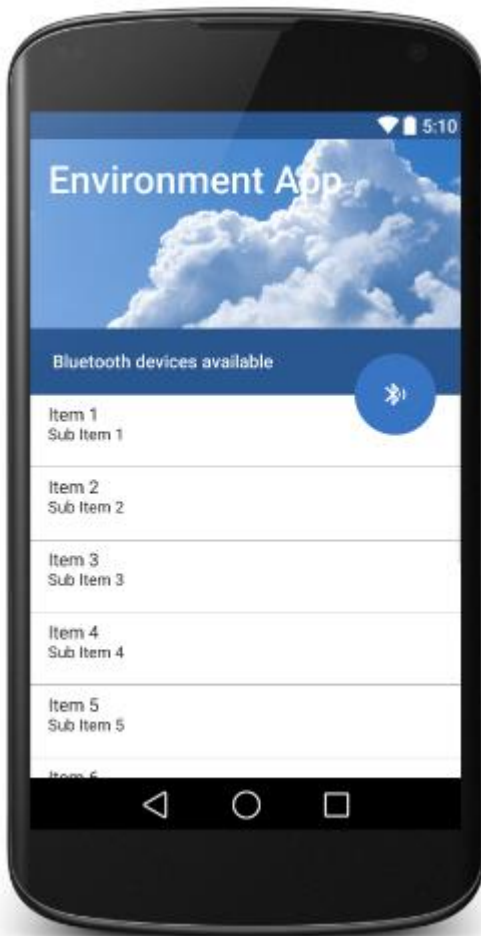


Figure 9.2 Activity Connection

activity_connection.xml

This image is the emulated Splash Screen as seen by the user during a pairing / connecting event. It displays the Title of the Android Application and all Bluetooth devices in range.

During this event, if not already active, the Smartphone will request for Bluetooth activation; afterwards it will attempt to locate the Bluetooth capable device.

By selecting the item (device of interest) the phone will attempt to mate with that device, in this case will be the CSR Env Sensor (BLE device) as labelled in the firmware (CSR1010 module) of the device.

If pairing is successful, then the 'activity_main' Screen will be displayed. See next page.

Here, the Smartphone Client was able to obtain a direct connection with the Bluetooth Low Energy Server and thus went onto the main user interface screen called 'main'.

The ASU Logo and label at the bottom of the Splash Screen is from the original ASU design project from EEE590: Reading and Conference. This Thesis on 'Bluetooth Low Energy for use with MEM Sensors' is a derivative of that project.

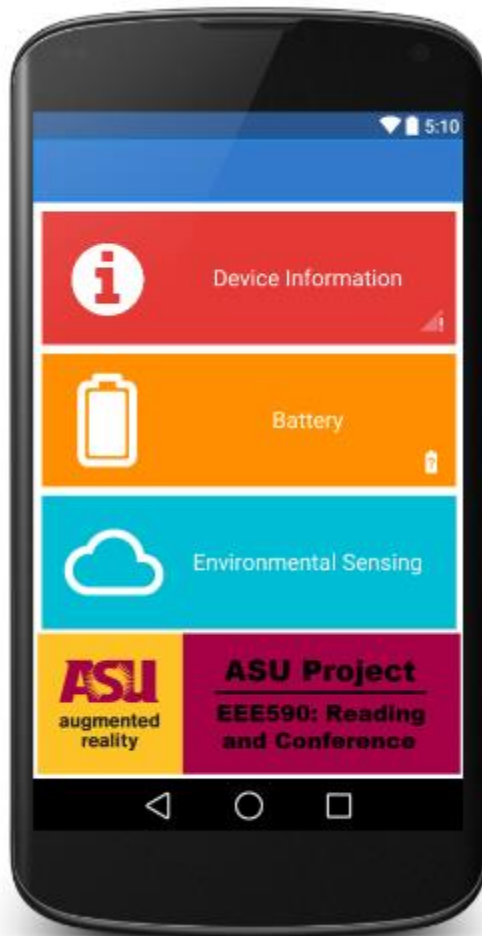


Figure 9.3 Activity Main

activity_main.xml

This image is the emulated Splash Screen as seen by the user after pairing / connection has been established.

Three buttons are available that gives the user the choice to read from the hardware CSR1010 BLE module and the MEM Sensors.

- *'Device Information' as read back from the firmware (CSR1010 module).*
- *'Battery' as read back from the CSR1010 module battery monitor.*
- *'Environment Sensing' as read back from the MEM Sensors connected to the CSR1010 module.*

Here the Device information is displayed. This information or data is retrieved from the GATT Service through the BTSmart Library files and manipulated or controlled through the corresponding Activity Java files.

- Device information displayed through the activity_information.xml is from the file InformationActivity.java

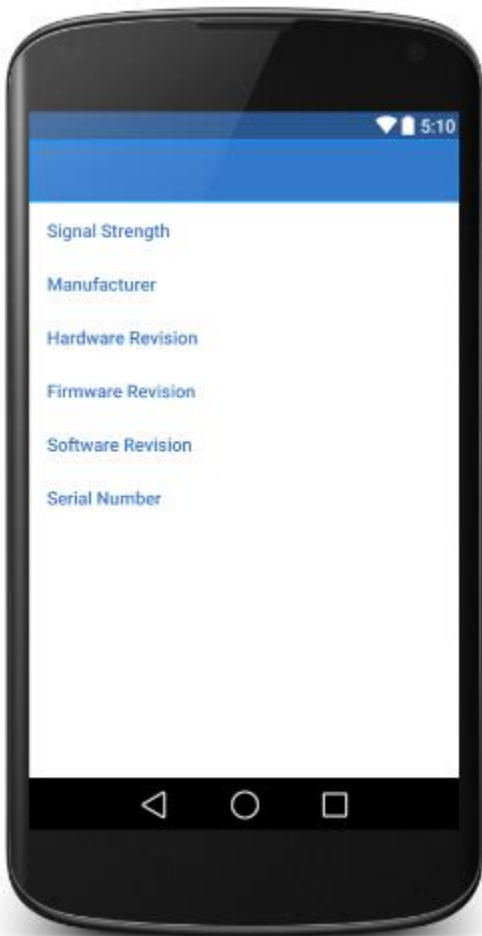


Figure 9.4 Activity Information

activity_information.xml

- Emulated Splash Screen after 'Device Information' button is selected.
- The fields are populated with the data that was read from the running service, 'BtSmartService'.
- The data displayed is the info stored in the firmware upon programming, except for the Signal Strength.

Here the Battery information is displayed. This information or data is retrieved from the GATT Service through the BTSmart Library files and manipulated or controlled through the corresponding Activity Java files.

- Battery information displayed through the activity_battery.xml is from the file BatteryActivity.java

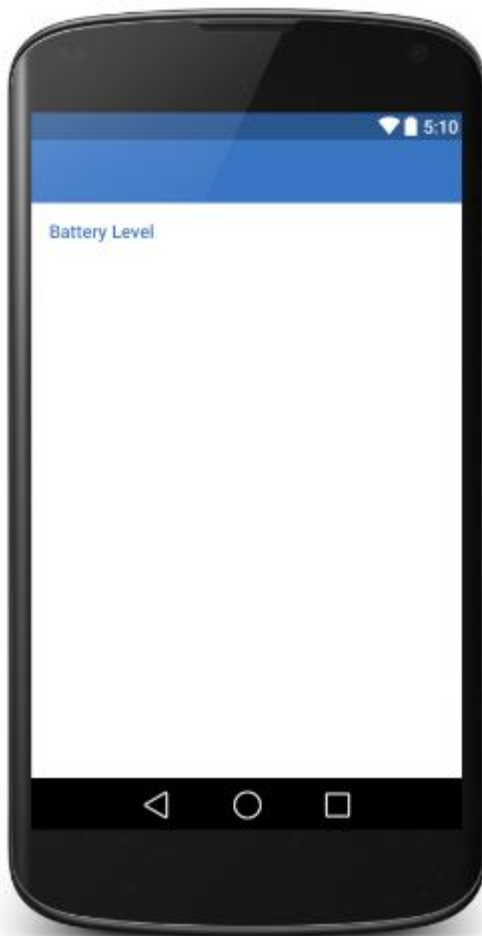


Figure 9.5 Activity Battery

activity_battery.xml

- Emulated Splash Screen after 'Battery' Button is selected.
- The field is populated with the data as read from the CSR1010 battery Monitor through the 'BtSmartService'.
- The data displayed is the measured Voltage level of the onboard Coin Battery.
- It will display 100% if either the hardware jumper is configured to USB mode with port externally powered or while the battery is fully charged in Battery mode.

This is the main Environment Sensing Display screen where the data from the sensors are shown and updated at a predetermined interval rate, configured in the EnvironmentActivity.java file. This file controls how the data is utilized once received from server (hardware). The file can be located in this directory:

```
//Phone App/Software/EnvironmentApp/app/src/main/java/com/csr/envapp/activities
```

To change the data polling rate, in milliseconds, modify this portion of the routine:

```
private static final int TIMER_REQUEST_CHARACTERISTICS = 1000;
```

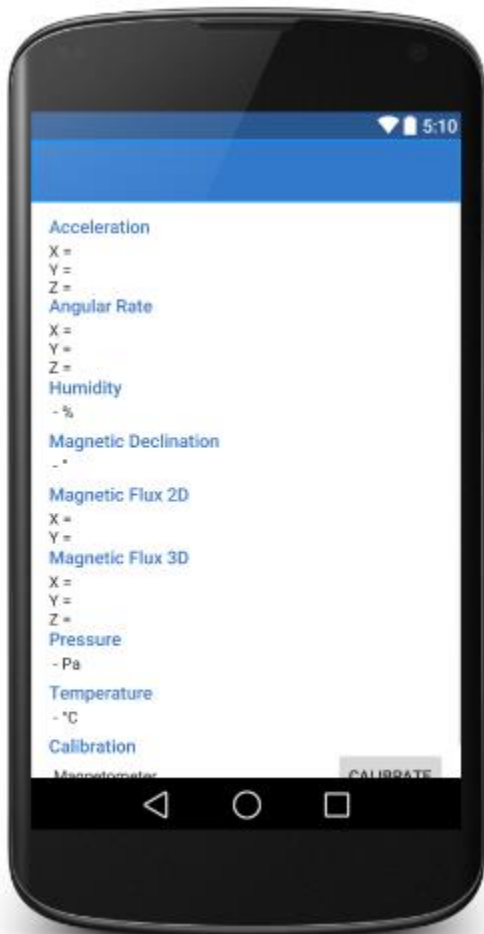


Figure 9.6 Activity Environment

activity_environment.xml

Emulated Splash Screen after the 'Environment Sensing' button is selected. Here five MEM Sensors are updated every second through the BtSmartLink notifier:

- Accelerometer: displays in m/s^2 on the X, Y, Z
- Gyrometer: displays in $^{\circ}/s$ on the X, Y, Z axes
- Humidity: displays in % relative moisture
- Temperature is displayed in $^{\circ}C$
- Magnetometer data is displayed in both 2D and 3D and is relative to magnetic north, calculated as Magnetic Declination with 360° of rotation
 - A calibration button allows for recalibration to magnetic north, if necessary
- Pressure data is displayed in PA = Pascal

If the End-user or Developer adhered to the instructions as described in previous sections, they should get the same results, look and feel as displayed in this section.

9.1. Results of Prototype Implementation and Demo

The Prototype development and Implementation were successful, with the expected End results. Please see the following sections for a personalized account:

Section 9.1: Are the demonstrations

- This section provides links to locations for some operational demos of the BLE Environmental MEM Sensors designed for this project.

Section 9.2: covers the Phone App

- The screen emulations support the Graphical User Interface on the Android (19) App on the Samsung Galaxy S4, as seen in the video.

Section 9.3 and 9.4: discuss bug fixes, workarounds and issues

- Some issues did arise during development but through workaround were able to get a functional end product.

9.2. Environmental Sensor Board demonstration

The goal of the demo is to prove to the audience that this BLE design is real and performs as advertised in this document. There are two demos in this section that demonstrate the performance of the Environmental Sensor Board design and implementation.

The **first Demo** runs a CSR Project Demonstrator program local to the PC and sends commands through the USB cable to read back the Sensor data. The video is a USB tethered demonstration only. As of July 9, 2015, Simon Finch, our CSR Market Consultant, placed a video up on www.youtube.com giving an Environmental Sensor Board demonstration on the platform developed for this project.

- Option 1: can Watch the Video at CSR: <http://www.csr.com/products/bluetooth-smart-environmental-sensor-board>
- Option 2: watch video on YouTube: https://www.youtube.com/watch?v=_CL8re2f0-A

The **second Demo** was developed for this project to demonstrate the wireless interface through an Android Smartphone. Currently working on getting this software and a user guide up on the CSR support domain / site for future referencing, but may take quite some time depending on scheduling.

Demonstration:

Search the directory for the '[BLE_Environment_Sensors_Demo.mp4](#)' that accompanied this document. The Video demonstrates that all requirements have been satisfied, according to the design:

- Remote MEM Sensors data is received by Android S4 device and displayed on screen
- The Phone's Bluetooth connects/pairs with the BLE CSR1010 device
- A run through of each button and corresponding displayed info as sent from the BLE CSR1010
- Full manipulation along the MEMs Sensor board axes to demonstrate that the Environment MEM sensors are reporting on the Android App display as expected.
 - **Accelerometer**
 - Data changes with orientation of x, y, z axis
 - **(Gyro) Angular rates**
 - Data changes with orientation of x, y, z axis
 - **Temperature and Humidity**
 - Data changes as a breath is applied
 - **Pressure**
 - Data displayed is the Atmospheric pressure
 - **Magnetometer**
 - Data will change as the Environment Sensor is rotated 360° with respect to Magnetic North
 - Calibration feature allows to reposition the sensor to magnetic North

9.3. List of Code Debugs and Fixes

Some issues or otherwise inconveniences were encountered during the development of the Phone App software. A list is provided here for completeness.

- A connection issue, as it concerns the application, when the device is already paired
 - Spent considerable amount of time debugging this issue. Upon disconnect the Android OS Stack would not unpair with the Hardware, therefore if additional connections were made between the App and hardware then not all Characteristic data is transmitted and displayed to the Android screen. Determined it was a bug with the Android OS and not the Phone. See bug in next subsection for additional details.
- Removed the automatic disconnection when a characteristic fails. This toast appears when the application is trying to register for an available characteristic and fails. This was a carryover from the previous application, and it was decided at the time to disconnect from the device when that happens. It is removed by commenting out the line 567 from the BtSmartLink.java file which is into com.csr/envapp/model. The line is “parentBtSmartLink.disconnect();”. See bug in next subsection for additional details.

The code was updated with fixes to the following:

- Magnetometer features
 - For the Magnetic calibration, added a button and pop up descriptor for calibration by the user.
 - Magnetometer calibration: there was a problem with the on/off switch working from the application side. The calibration values are to be sent from the client to the server (CSR1010) but errors in the code prevented transmission. Checked with a sniffer to verify that the Phone was sending the expected information.
 - In order to get the Magnetic Direction (Declination) to be functional had to do some Math from the Magnetic Flux characteristic values (2D and 3D) since there is no Magnetic Declination characteristic sent from the Environment sensors software for the chip. Also, changed Tesla to Gauss.

- Issues displaying the CSR custom defined characteristics on retrieving values.
 - Angular rate (CSR custom defined characteristics): the values the application receives are signed and not unsigned as formerly thought versus the signed value coming from the Acceleration: (CSR custom defined characteristics).
 - Pressure: Was using the wrong type, “int” instead of “long” on Java.
 - All database handles needed to be reverse, the bytes that are being received – as it works generally with Bluetooth.
 - For all other characteristics, see the [Developer Bluetooth official website](#) [20]

9.4. Issues seen during Design / Roadblock & Workaround

This section provides information regarding issues and / or bugs that were observed during the development of this project.

Bugs: Bugs observed with the Samsung Galaxy S3 and S4 and equivalent Google Nexus 4 / 5

1. After the Android App connects and pairs with the CSR1010 Module (GATT Service), an error occurs that causes the running Phone Application to disconnect and exit. The workaround was to tell the Android App to ignore the error to prevent the APP from closing (default Android response).
 - All UUID characteristics that are used for the MEM sensors request an encryption. When Android requests the first characteristic to the Bluetooth Device, it automatically fails (timeout event) before Android can ask for the encryption. Normally, at this moment the Android stack should keep this first characteristic as “has to request it again”, but it doesn’t. That’s why it’s not working here. By ignoring the error and not exiting, the Android can request the encryption again with no further issues. The issue with this bug was originally seen on Samsung Galaxy S4, but was not seen when using a Nexus 7. The bug was then also verified as present on the Nexus 5.

2. Under normal conditions, after the Application has started, the connection and pairing process are engaged, requesting the characteristics, fails, encryption, and then the remaining custom characteristics. The very first time, the application runs as expected. The devices are paired and displaying all data as received by the GATT Client. Afterwards the application is closed by the user. It is expected that the device will unpair upon exit, however, the devices remains paired. Upon running the application again, the device is already paired. The (Nexus 5) Android stack looks lost or not working: it is never able to come back to the application to say what happened. So the application is blocked on waiting information from the Android stack which never arrives. On the Galaxy S4, the Android stack reads back only partial characteristic data. Not all MEM Sensor information would have the data updated and displayed; such as, Gyroscope, Accelerometer and Magnetometer data. Data would just display '0'. It was determined that the CSR custom UUID for these MEMs do not transfer to the GATT Client under this condition.

- This is a limitation with the Android OS Firmware for these two models of phone. Was told that this bug does not exist in Android models S5 and later, verified on an S6.

Solution: There is nothing that can be done for the Android stack as it belongs to the Android OS. The only solutions at the moment are:

- Use a Galaxy S6 (Nexus 7) or newer model
- Galaxy S4 and Nexus 5, unpair the device every time one would want to restart the application (go to Bluetooth settings and “forget” the device) should fix the characteristics request issue for that connection.

Issue: There was an issue with the original Android Development Environment (Eclipse) that was used for this project, it was no longer supported by Android / Google and therefore prevented further development of the “APP” causing a roadblock condition. The “App” design referenced the CSR skeleton code that was written in Eclipse; but, with that platform no longer supported had to start over with the Phone Application Design.

- After speaking with Bluetooth SIG, in an attempt to find a workaround, determined that the Eclipse IDE Platform and the Android Development Tool (ADT) plugin were no longer supported by Google / Android and were not active.
 - Had to switch over to the new Android Studio with the IntelliJ IDE Platform for the code development, resulting in a redesign of the software code from the original approach. The new environment was easier to program with, more stable and did not require workarounds or hacks to get the app to function as expected
- All other issues- see the *SDK Release Notes*.

10. Summary and Future works

For over 15 years, it was envisioned that people would become connected wirelessly with everyday objects that surround their environment; making life more convenient as they try to squeeze every last second into their lives as possible. This movement has become known today as the 'Internet of Everything' and the market is taking full advantage of this fact by continuously driving up the demand for these wireless connectivity technologies [14].

It is so easy today to incorporate a wireless solution into any project. The theme of this paper was to devise a reference platform that could be readily utilized by others to minimize cost and development time with a wireless solution. This platform described in this document should give the developer / end user a method to quickly develop a prototype strategy for their remote sensors requirement.

The goal was to instill in the reader the philosophy of Rapid Prototyping by using reference material already available on the market as a baseline to get started with their design.

Key features:

- Android Phone App and Source code that is associated with this project could be used as a reference to get the end-user up and running in the shortest amount of time. As a baseline reference, the software is fully functional with all the bugs either fixed or documented with a workaround.

- Hardware and Firmware (Embedded Software) could be purchased off the shelf, such the development kits and modules (Digikey or CSR), along with the use of firmware as a reference point to get started with the prototype. The COTS have already be proven, adhere to the FCC regulations and have been qualified by Bluetooth SIG, improving the developers odds that the prototype design will work as expected the first time.

The Developers and College Students needing to create the next generation of sensors/MEMs-enabled smart devices now have a simple solution to enable them to develop in the minimum time possible.

Future Work: It is of great interest to apply what was learned during this project and apply these skillsets to work with bioelectronics and other sensors that can be worn on or in the body. This is where the technology is headed in the near future.

11. Documentation and Other Collateral

This section identifies supporting documents and evaluation kits that will be provided to support end-user engagement and integration and does not attempt to capture all of the internal deliverables, such as lab boards, compliance matrices, and the like; which are requirements of the development process.

11.1. Documentation

There are additional documentation that were not specified directly in this document that could provide additional insight into further development with Bluetooth Low Energy , such as Application Notes, User documentation, and supporting software for the ADK/SDK and Tools. These files were useful in development of this project and should accompany this document.

11.2. Hardware and Development Kit

Please note that the hardware and supporting items (the package) that are submitted along with this report are 'as is'. If it is desired for further development in Bluetooth Wireless, visit www.csr.com for additional reference designs and documentation. It is expected that customer development boards and ADKs/SDKs will continue to be updated with newer revisions. These follow-on activities fall outside of this project.

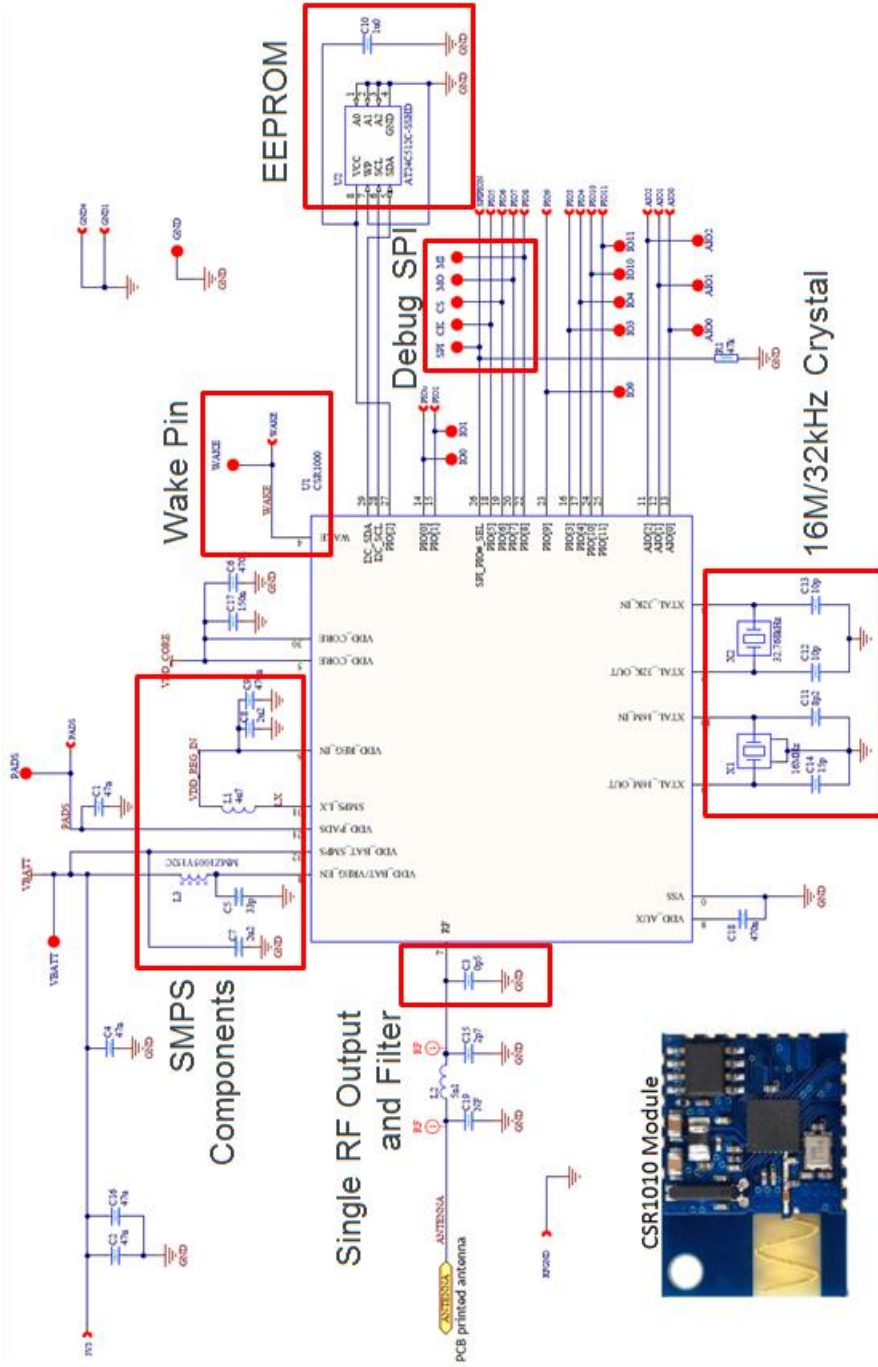
REFERENCES

- [1] Bluetooth Smart Environmental Sensor Board:
<http://www.csr.com/products/bluetooth-smart-environmental-sensor-board>
- [2] Bluetooth Smart Starter Development Kit
<http://www.csr.com/products/bluetooth-smart-starter-development-kit>
- [3] Bluetooth 4.0: Low Energy
<http://chapters.comsoc.org/vancouver/BTLER3.pdf>
- [4] RF Radio Frequency
<http://www.homtrol.com/show.php?id=582&newsid=54>
- [5] Industrial Wireless - Selecting a Wireless Technology
<http://www.bb-elec.com/Learning-Center/All-White-Papers/Wireless-Cellular/Industrial-Wireless-Selecting-a-Wireless-Technolog.aspx>
- [6] What is Bluetooth Technology?
<http://www.bluetooth.com/what-is-bluetooth-technology>
- [7] Understanding Bluetooth Technology
<http://blog.abrfid.com/understanding-bluetooth-technology/>
- [8] Bluetooth Smart Technology: Powering the Internet of Things
<http://bluesoleil.com/life/402.html>
- [9] Low Energy
<http://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy>
- [10] Introduction to Bluetooth Low Energy
<https://www.yumpu.com/en/document/view/38246955/introduction-to-bluetooth-low-energy/3>
- [11] CSR1010 μ Energy Development Kits
<http://www.digikey.com/catalog/en/partgroup/csr1010-%CE%BCenergy-development-kits/43200>
- [12] CS-212742-UG μ Energy xIDE User Guide:
<https://www.csrsupport.com>
- [13] CS-314507-AN_CSR μ EnergyEnvironmentSensorApplicationNote:
<https://www.csrsupport.com>
- [14] The “Internet of Things” Brings Sensors, Wi-Fi Connectivity to Everyday Objects
<http://www.broadcom.com/blog/wireless-technology/the-internet-of-things-brings-sensors-wi-fi-connectivity-to-everyday-objects/>
- [15] Bluetooth Core Specification Version 4.1:
<https://www.bluetooth.org/Technical/Specifications/adopted.htm>
- [16] Environmental Sensing Service Specification Version 1.0:
<https://www.bluetooth.org/Technical/Specifications/adopted.htm>

- [17] Battery Service Specification Version 1.0:
<https://www.bluetooth.org/Technical/Specifications/adopted.htm>
- [18] Device Information Service Specification Version 1.1:
<https://www.bluetooth.org/Technical/Specifications/adopted.htm>
- [19] CS-218270-DD CSR1010 Hardware Design Review Template:
<https://www.csrsupport.com/document.php?did=39334>
- [20] Developer Bluetooth official website:
https://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.environmental_sensing.xml

APPENDIX A

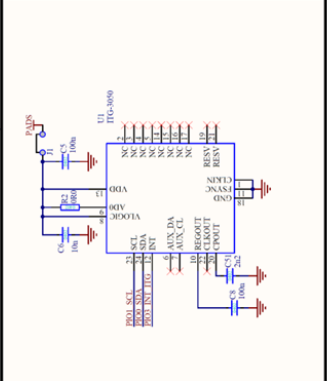
CSR1010 MODULE SCHEMATIC



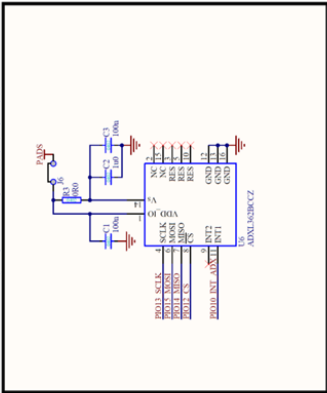
APPENDIX B

ENVIRONMENTAL MEMS SENSOR BOARD SCHEMATIC

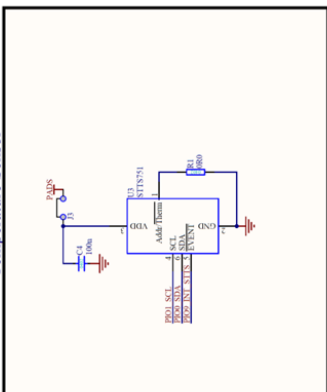
3-axis MEMS Gyro



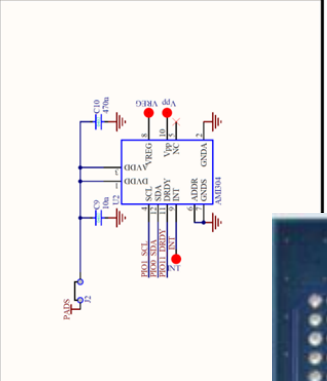
3-axis MEMS Accelerometer



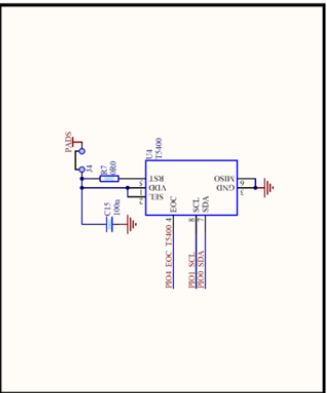
Temperature Sensor



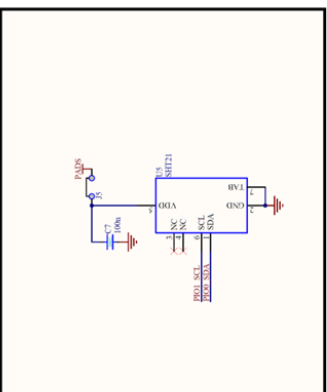
3-axis Magnetometer



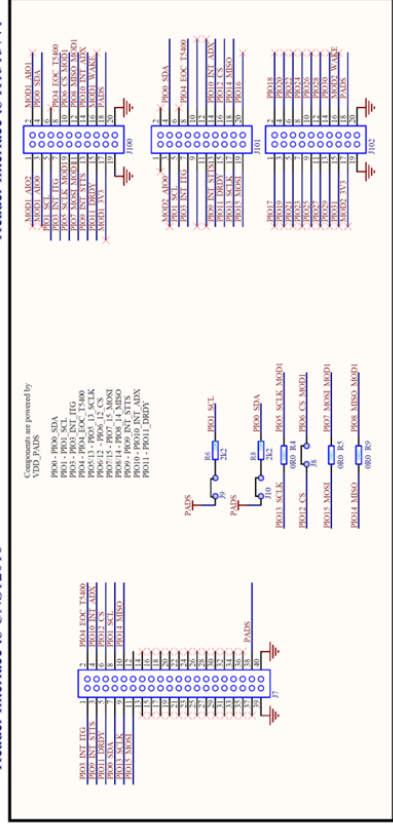
Barometric Pressure Sensor



Humidity and Temperature Sensor IC



Header Interface to CNS12016



Header Interface to HI3137v1

