iOS Sensor Apps With Arduino

James Royal

Instructor: Janusz Zalewski, Ph.D
Programming Embedded Mobile Devices
COP 4908 Fall 2013
Final Draft: 12-2-2013
1. Introduction

Today, society depends on mobile technology to stay connected with the world. From getting the latest news to connecting with family and friends across the globe, people depend on smartphones, tablets, and computers to socially connect on a daily basis. Social network giants like Facebook, Twitter, and YouTube have revolutionized the way users interact online. With such leaps and bounds in technology in the last few years people are eager to use technology in new ways. The availability of technology and open source code has brought the future of mobile devices to the fingertips of modern day programmers. With this expanding market better ways to integrate our lives with these devices need to be discovered.

![Arduino Uno](image)

Fig. 1 Arduino Uno

It is not commonly recognized that society has been playing with these small devices for decades, although, it may not have been in the form society is familiar with today - such as a mobile device. All mobile devices are embedded systems and
embedded systems have been around for many years. These systems are in just about every technological product on the market from toys to household appliances. Only within the last ten years has the market for embedded systems exploded into the mobile devices that society has become so dependent on.

These devices, better known as mobile phones and tablets, have taken the economy by storm revolutionizing the interaction between consumers and businesses. Through the evolution of embedded systems almost everyone has a device that puts the world in their pockets.

Figure 1 details a typical embeddable computer, an Arduino Uno, released in 2005, which is a programmable microcontroller. This programmable board makes the process of engineering small embeddable devices much easier. Using a specific programming language, which has syntax much like C, one has control of the entire board as well as any device connected to the specific pins on the board. This board is particularly popular because it enables engineers and designers of embedded systems to bring products to the market much faster. The reason for the shortened time is due to the Arduino being an open source project as well as low price or the board costing about $29.95 according to Adafruit.com [1], which makes the Arduino a great choice for any project on a low budget.
2. Problem Description

The goal of this project is to use a tablet, specifically an iPad as seen in Figure 2, with an Arduino microcontroller to interact with the Xbee wireless network to provide real-time data on the Network. Using a Redpark Serial Cable for iOS, specifically the C2-DB9V model [5], one has the ability to connect the iPad's 30-pin dock connector to an Arduino microcontroller. This will allow for communication between the iPad and the Xbee wireless network with an Xbee controller residing on the Arduino board. This communication will enable the capability to turn on and off a wall outlet via an iPad between two Xbee radios.
Figure 3 shows a Redpark serial cable that has on one end (the left) has a standard serial DB9 connector and other (the right) is an Apple standard 30-pin dock connector. This cable is required for communication between an iOS mobile device to an Arduino board for use with personal projects. The reason for this requirement is that Apple does not allow uncontrolled use of their proprietary dock connector. Meaning one can't simply make a cable to connect to an iOS mobile device. Each Redpark serial has a tiny microchip in it that signs with each iOS device once the cable has been connected. If this microchip is not present, the iOS device will not communicate with the connected cable. If an individual/company wants to bring a product to the market that does not rely on the Redpark Serial cable they must register with Apple through Apple’s MFI developers program. Once accepted into the program the individual/company has the ability to hire one of Apples exclusive manufacturers in order to produce their own 30 pin dock connectors with
their own desired hardware as well. The kind people over at Redpark Industries have completed this step; therefore anyone wanting to tinker with their own DIY projects can do so without the need of being a MFI member, they simply must purchase the cable from Redpark Industries.

Figure 4 shows a RS232 conversion board for the Arduino. This board is the final component needed to make an iOS device communicate with an Arduino board, simply by connection directly to the Arduino. The Redpark serial cable connects to this piece directly. Without this piece there is no way of connecting to the Arduino board.
3. **Software Setup and Installation**

In order to write software for the Arduino, a programmer needs the proper development environment installed and setup on the host system. One can find the Arduino IDE from the arduino.cc website [3]. The IDE is not required although it is recommended as it helps for faster development. These instructions are expressly for Mac OSX, although the Arduino project has support for Windows and Linux operating systems as well.

*Fig. 5 Navigate to arduino.cc select download*
Fig. 6 Select your Operating System.

Fig 5 and 6 detail how to download the Arduino IDE from the Arduino website, specifically for Mac OSX. In order to acquire the Arduino IDE one simply needs to navigate to the Arduino.cc main website from any web browser on their personal computer. Once there, look for the link that says download. This is detailed in figure 5. The download link will take the individual to another page that is detailed in figure 6. Once on that page, about half way down there are several links for the different Operating Systems. Select the link for Mac OSX. The file comes in the standard dmg format and is ready to run once opened and copied to the applications folder. Depending on the Operating System the exact process will vary. Once installed and verified by opening the application, one should see something very similar to Figure 7.
Fig. 7 Arduino IDE
4. *Arduino IDE Basics*

The IDE is very simple for an advanced programmer that is familiar with IDE’s such as Eclipse, Visual Studio, or the Xcode toolset while a novice programmer that is not acquainted with coding could be quickly overwhelmed.

![Arduino IDE Explained](image)

**Fig. 8 Arduino IDE Explained**

From figure 8 one should notice there is not much to the UI of the IDE. This is due to the Arduino IDE having the ability to be broken into eight parts. The parts are as follows and correlate with the numbers in figure 8 respectively:

1. This button allows one to validate their code written in the window. It checks for errors.
2. This button is to flash the program from the Arduino IDE to the Arduino Microcontroller.
3. This button allows for the creation of a new file/program.
4. This button allows one to open existing programs.
5. This button allows one to save their current program.

6. This is the editing window and all code should be placed here.

7. This will open a console window. It allows for log messages from the Arduino once the program is running on the board.

8. This is where any build/compile error will be displayed as well as any coding mistakes.
5. Hardware Setup

Once the IDE is setup and installed it is time to set up the Arduino board. The RS232 Shifter SMD adapter from Spark Fun comes without the ability to connect directly to the Arduino board. In order to make it connect and interact with the Arduino one needs to solder headers onto the board. This is not required but recommended so that it is easier to connect and disconnect cables to and from the RS232 Shifter board, in Fig. 9.

![Solder 4 headers here.](image)

Fig. 9 Four connector holes on the RS232 Shifter SMD
Fig. 10 Universal Headers

Pinch/break off 4 Headers

Fig. 11 Pinch of the desired number of headers

Fig. 10 and Fig. 11 detail a set of universal headers designed specifically for electronics projects. These headers can be purchased from Radio Shack. In this project there is only a need for four of the headers to fit in the RS232 Shifter SMD.
connector holes. Specifically Fig 11 displays how to remove the four desired headers. To do so, simply pinch between the fourth and fifth header and bend to break.

![Image of header details](image)

Fig. 12 Note the different lengths.

Make sure that when soldering the headers to the board to take into consideration the lengths of each side of the headers. Fig 12 shows that one side of the header is just a bit longer than the other side. The longer side is used for a quick connect and disconnect of a jumper wire. The shorter side is the side that should be soldered to a board. In this case the shorter side will be soldered to the RS232 Shifter SMD board, in Fig 4.
Before soldering take a look at Fig 13. Make sure the plastic piece between the headers is on the top side of the RS232 Shifter SMD board, in Fig 4. The only thing that should be seen from the bottom of the board is a very small piece of the shorter side of the headers. Also when soldering be sure that the solder does not make a connection between two of the pins. This will create an undesired outcome when supplying the board with power. This could also deem the board unusable by burning the circuit.

Fig. 13 Headers preparing for solder.
Once the soldering process is complete, the RS232 Shifter SMD board should look very similar to Fig. 14. Note the placement of the plastic header separator. It is on the topside of the board.

The major motivating factor that one should take these steps is so that they would be capable of using jumper cables between the RS232 Shifter SMD board and other boards and devices, and so they do not have to solder and unsolder wires to the four holes on the board when deciding to change or disconnect the board.

As stated previously one can now use jumper cables for connection to the RS232 Shifter SMD board. Fig. 15 and Fig. 16 display the necessary jumper wire for the headers that were soldered to the board. Note the female and male end. Using four male to female jumper cables slide the female ends over the header prongs. This process is shown in detail in Fig. 16.
Fig. 15 Male to female jumper cable.

Fig 16. Connect female end to Header.
Once complete the board should look very similar to figure 17 with four wires connected to the RS232 Shifter SMD board ready to be connected to the Arduino board. At this point it would be very wise to add a label to each wire so that each pin can be identified very easily when plugging and unplugging the jumper wires. There are four pins on the RS232 Shifter SMD board:

1. VCC (Power specifically 5 volts DC, no more.)
2. GND (Ground)
3. TX (Transmit)
4. RX (Receive)

A few small pieces of masking tape should suffice in labeling. This step is just so there is a reference to the pins later.

Once the Fig. 4 RS232 Shifter SMD is finished being soldered it is time to setup the Xbee Module for use with the Arduino board. This process is very similar
to the setup of the Fig. 4 RS232 Shifter SMD in that it requires headers to be mounted to a shield that can simply plug into the top of the Arduino board. This time though it will be a set of female headers not the male headers used previously. Theses headers are shown in Fig 18.

![Fig. 18 Female Headers](image)

To make this work properly one needs two sets of eight headers shown in Fig 18 and also two sets of six headers.
Fig 19 details precisely where to place the headers on the Xbee wireless shield purchased from Sparkfun.com. Simple solder all 28 headers to the Fig 19 Xbee wireless shield following the same steps that were taken when soldering the RS232 Shifter SMD in Fig. 4. Make sure the black plastic is on the topside of the board. To ensure this simply insert the headers into the board from the topside down. Once complete the Xbee wireless shield should look like Fig 20 and Fig 21 with all headers attached.
The reason for the use of the Xbee wireless shield in Fig 19 is so that connecting the Xbee wireless sensor to the Arduino is easier and not so many wires running around. If this step were not taken one would need to solder and unsolder wires all the time for different applications.
Now that the Xbee shield in Fig 21 has its headers mounted correctly one can simply plug it in to the top of the Arduino Board. This project will require two Arduino boards with two Xbee shields so the previous steps need to be taken twice.
6. Connecting Arduino Xbee and the RS232 Shifter SMD board

Now that the majority of the soldering is complete for the initial setup of the all the parts it is time to connect the parts together to acquire communication between them. Since the headers were soldered to the Xbee wireless shield in Fig 21 it is now stackable for use with the Arduino. Simply line the bottom of the headers up with the Arduino and plug it in. This is detailed in Fig 22 and Fig 23.

Fig 22 Xbee Shield Stacked on Arduino
One can’t really mess up this part, as there is only one way to plug the shield into the Arduino. The key benefit to this is that now all of the Arduino pins are exposed on the top of the board and the Xbee sensor once plugged in will be connected to the circuit.

At this point one should test the device to make sure the connection and all soldering was done appropriately. One can do this by plugging the Arduino into a power source via USB or external battery. If all goes well one should see a red LED light above the reset pin labeled “PWR” for power. This is shown in Fig 24.
Fig 24 Power on Test
7. Building the Project

7.1 Introduction

As stated earlier, this project will be using an iPad to control an Xbee wireless network. At this point, one may ask what exactly will the Xbee sensors control? The sensors will be used to control the power to a 120-volt wall outlet. There is a need for two Arduino boards and two Xbee sensors in order to do this. The iPad will be connected to one Arduino with an Xbee sensor on board. This Xbee will send a signal across the Xbee wireless network to the other Arduino board connected to the second Xbee sensor at which time the second Arduino will flip a relay to turn on power to the socket. This is described in Fig 25.

![Diagram](image)

**Fig. 25 Design Diagram**

To get started creating the project one will need to obtain the following parts; a wall outlet, a box of some kind to house the Arduino and the wall outlet, and
an extension cord preferably one with a ground (typically the bright orange ones have the desired wiring). Figures 26, 27, and 28 show the parts that are recommended.

Fig. 26 Outlet box

Fig. 27 Outlet
Fig. 28 Orange extension cord with the female end cut off

Once all the parts are acquired, it is time to build. The blue box has some small structures in the bottom that prevent the Arduino Board from fitting in properly. Simply cut them out with any tool that is sufficient for the job.

Fig. 29 Structures
Once the structures are trimmed down enough insert the Arduino in the bottom, measure two holes for the Arduino’s USB and power connector. Drill or cut the measured holes out and place the Arduino in the bottom of the box. This is shown in figure 30, 31, and 32.

Fig. 30 Measured Holes

Fig. 31 Arduino place in blue box
Fig. 32 Cutting complete
7.2 Configuring the AC Circuit

Now that the cutting is complete and the Arduino fits nicely in the bottom of the box, wiring up the 120 volts AC circuit is next. Pay close attention here 120 Volts can and will kill the user if not handled appropriately. The user should handle with extreme caution as electricity is very dangerous and can have disastrous effects if mismanaged.

Run the orange extension cord through the bottom of the blue box so that it is nice and clean. Figure 33 shows this.

![Fig 33. Inserting Extension Cord.](image)

There are three wires that come from the extension cord. The three are typically colored following electrical wire standards, although this is not always true. The Green wire is for the ground, the white wire is the neutral, and the black wire is the hot wire. It needs to be reiterated that one can easily be electrocuted by any of the
above wires no matter the color or purpose. Don’t go grabbing the white wire just because it is called the neutral. If there is a potential difference connected to the other end of the extension cord one should consider each wire as a live wire. A live wire is simply a wire that can be considered to have electricity flowing through it and has the ability to electrocute. Notice the different wires in Fig 34.

![Fig. 34 Different wires and their purpose](image)

Once all three wires have been identified the user needs to understand where they are to be connected to the outlet. There are three different screws on the outlet. One is green and this is where the connection to the ground will be placed. One screw is silver, which is the neutral, and this is where one should connect the white wire. The last screw is gold this is the hot wire and the black wire will connect to it. For details please view figure 35.
Before connecting all the wires please make reference to figure 25. Notice that there needs to be a relay installed between the power source and the outlet. The relay that was used here was purchased at a local radio shack on clearance. Although any relay will suffice. Just make sure that it is rated for 120v AC at 15amps as this is the maximum current from the wall outlet, and that it has a coil resistance of 12v DC or less. The 12v DC current will be used to flip the relay back and forth.

The Arduino will be used to flip this relay once a signal is sent to the Xbee radio. When installing the relay take a look at the pin layout. With this one included, there are five pins. First we have the commonpin that typically makes connection where the current is coming from. In this project the extension cord plugged into a wall outlet is where the current is coming from. Second the normally open pin is the pin that will allow current flow at rest without being flipped. Third the normally closed pin is the one that will allow current flow when the relay is flipped. Last we
have the two pins that flip the relay. These pins are called the coil pins and are
where the 12v DC charge should be applied. There is no positive or negative side to
these two pins. To flip the relay place a positive 12v DC charge to one side and a
ground to the other side allowing a current to flow through the coil. To install the
relay use the black wire, cut the black wire about midway allowing enough space on
each side for movement. Solder the black wire coming from the extension cord to
the common pin on the relay. Then solder the side of the black wire that was cut to
the normally closed pin. The finished result can be seen in figure 36.

![Fig 36 Relay Connection](image-url)
Once all of the connections have been made one should see something very similar to figure 37. This completes the circuit for the 120v AC current.

Once the 120V AC circuit is completed it is time to move onto the 12V DC circuit. The 12V DC circuit will be used to flip the relay and allow power to the outlet. This circuit will be connected to the Arduino. The flipping mechanism will be programed using the Arduino IDE. Lets test the relay now to see if it works. 9V is enough to test the relay and make sure it is flipping appropriately. Make sure the extension cord is unplugged from an outlet before attempting the next step. Grab two wires it doesn't matter the color. Connect both of them from the 9V battery to the coil pins on the relay. The common pin typically sits between the coil pins. Figure 38 details the pins. Once the connection is made a noise from the relay should be very prominent signifying that the relay has been flipped.
Fig 38 Coil pins
7.3 Configuring the DC circuit.

Before taking on the challenge of wiring up the DC circuit one needs to understand one of the basic principles of electricity, which is known as Ohm’s Law. Ohm's law states that there is a direct relationship between the voltage, current, and resistance. The best way to depict this relationship is by using Ohm’s triangle, which is shown in figure 39.

![Ohm’s Triangle](image)

From figure 39 if one desires to know how many volts are needed then they must know the current denoted by (I) and the resistances denoted by (R). Voltage is measured in volts, current is measured in ampere or amps, and resistance is measured in ohms. Also from the figure if one needs to know how much current will be needed they would divide volts by the resistance, and for deciding on resistance one would simply divide volts by the current. These equations are for the standard unit, which means if you are given a unit in milliamps or millivolts then this value
must be converted to amps or volts before the calculation in order to obtain the right value.

Now that the basics are out of the way the following is the design of 12v DC circuit. When designing the circuit one should keep in mind that the relay needs 12 volts in order to flip on, not only 12v but also 30 milliamps of current. These values are detailed on the datasheet of each component. This causes an issue because the Arduino only outputs 5v from any one of its programmable pins. With that said, there has to be a way to step the 5v of power output from the Arduino to 12v required by the relay. There are many different ways to solve this, though this project will be using a Bipolar Junction Transistor along with an external power source to get the job done.

There are two different types of Bipolar Junction Transistors NPN and PNP. Bipolar Junction Transistors consist of three terminals identified by the collector, base, and emitter. With NPN, the gate between the collector and the emitter terminals is shut or at rest and when a positive charge is placed onto the base terminal it opens the gate and allows current to flow between the collector and the emitter. Likewise with the PNP transistor the gate is open at rest allowing current to flow between the collector and the emitter, and once a charge is place on the base terminal it pulls the gate shut. Figure 40 details these two bipolar transistors.
This project will utilize the NPN transistor to act as a gate for an external power source of 12V. Remember this is needed due to the fact that the Arduino only outputs 5V maximum from any programmable pin. Another part that will be needed is a rectifier diode. The purpose of the rectifier diode is so that current is only allowed to flow one way within the circuit. Electricity has the capability of moving backwards so if the circuit is not prepared for this one could easily burn up some components. Last, we have a resistor its responsibility is simple; it must protect the Arduino microcontroller. Since there is the use of 12V, this 12V power source has the ability to travel back across the Arduino and potentially fry the microcontroller thus we must use a resistor to prevent this from happening. Figure 41 details the overall circuit design and current flow.
From figure 41 one should get a clear idea of how the 12V DC circuit will work. Once the Xbee sensor receives a signal, the Arduino will output a 5V signal on any desired pin. This 5V signal will flip the transistor allowing the larger 12V power source to flow through the collector terminal of the transistor to the emitter terminal then exiting to the ground. This flip will allow the required 12 volts to flow through the Relay flipping it on. When the relay is flipped on it allows the 120 volts to flow through thus powering the outlet.
7.4 Configuring the Xbee Sensors

Now that both AC and DC circuits have been established its time to configure the Xbee sensors so that they can speak to one another. This project uses two Xbee series 1 sensors. The benefits to this series of Xbee sensors are that they are essentially plug and play. Although, they are configurable so make sure the two that are in use are configured to communicate with each other. When in doubt, to save a lot of headache and pain, it would be wise to reset them even if they are brand new.

The Xbee sensors are like any other networked device in which have several parameters that must match so that they can communicate over the airways. These parameters help to distinguish the network in which the sensors can communicate. The parameters are stored on each Xbee sensor and in order to be changed one must access the sensor over a serial port. This project does not require the change of any of these parameters as long as the two Xbee sensors have been left untouched. Even though the sensors are left unchanged one should know how to reset them. The following is how to do so using the Arduino IDE and Xbee explorer dongle from spark fun. This board can be viewed in figure 42.
This dongle makes changing settings to an Xbee sensor a breeze. Plug the Xbee that needs to be configured/reset into the top of the dongle. This step is shown in Figure 43.
Once the Xbee has been inserted into the dongle insert the dongle into one USB port, and open the Arduino IDE. Once open go to tools -> serial port from the toolbar and select the serial port that corresponds to the dongle. View figure 44 for details.

Fig 44 Select the appropriate serial port

Once the proper serial port has been selected open the serial monitor in the upper right hand corner of the IDE. See figure 45 for details.
Fig 45. Serial monitor.

The serial monitor should look very similar to fig 46.

Fig 45 Serial Monitor

The following steps are very crucial to resetting an Xbee sensor. There are three commands to do this. The First command will put the Xbee into configuration mode.
Once in this mode the end user only has 10 seconds to finish changing settings or the Xbee leaves config mode, so be quick. The second command will be issuing a reset. The final command will tell the Xbee to write the reset to its flash space. There is a tricky part though to enter into configuration mode the command must be sent with no line endings. Once in before issuing the reset and write commands one must change the line endings to carriage return. The commands in complete sequence are as follows:

1. Change line endings to No Line ending
2. +++ (Command to enter into configuration mode. Should receive an ok)
3. Change line endings to carriage return
4. ATRE (Command to call reset)
5. ATWR (Command to write the changes)

The three commands should be followed with an ok from the Arduino. If everything went well one should receive three oks back from the Xbee displayed in the serial monitor. Simply issue the above commands into the serial monitor from figure 45 when an Xbee reset is needed. It is recommend to do this step before use with Series 1 Xbee sensors as there is no telling their current configuration. Once both Xbees are reset and configured to communicate simply plug them into the Xbee shields. It’s now time to start programming the Arduino as well as the iPad.
7.5 Building both Arduino programs

In this project there is a need for two Arduino programs. Both will be implemented from the Arduino IDE. The first program is the server program. This program is responsible for receiving the on/off command from the iPad and sending it out across the Xbee network. The second program is the client and is responsible for receiving the command from the server and turning the wall outlet on or off.

Fig. 46 Arduino server program
Fig. 47 Arduino client program

From figures 46 and 47 one should be able to see that Arduino applications are not very difficult to write, the difficulty is introduced with the circuit design in figure 41. Once both applications are written simply install them to their respective Arduino Microcontroller. If the install is not very clear please view figure 8 specifically number 2 from that figure. The server program should go to the external microcontroller, which will be connected to the iPad, and the client program should go to the microcontroller inside the blue outlet box. Now that both Arduino applications have been written time to move on to the iPad application.
7.6 Building the iPad Application

Start the Xcode IDE and remember Xcode is only available on Mac's. When Xcode starts the user is presented with a series of menus to create a new application. Within the first menu select a new Xcode project. Figure 48 shows this.

Fig. 48 Create a new project

Once a new project is selected the user is presented with another screen allowing for project type selection and weather the application. This can be seen in figure 49.
Fig. 49 Select the type of project

After the project type has been selected the next menu asks for one to provide a project name and whether the project should be built for iPhone or iPad. Figure 50 details this.

Fig. 50 Name the project.

The last menu, one will see, allows the user to select a save destination for the project. Please see figure 51.
At last one should have a brand new iPad project within Xcode ready to go. There are three more things that need to be added to the project in order to build the application so that it is capable of communicating over the 30-pin dock connector and through the serial cable. The first thing that the application needs reference to is Apple’s external accessory framework. This framework was specifically designed by Apple and holds all of the API’s and logic to communicate across the dock connector. To add this framework to the project select the project file in the left side bar -&gt;build phases-&gt; then under the link binary with libraries select the + button. Once done a window should slide down from the menu bar showing all frameworks/libraries that can be added to the project. In the search bar start typing external accessory framework and the list should sort it-self revealing the framework select it and click add. This process is shown in figures 52 and 53.
Fig. 52 Adding the external accessory framework.

Fig. 53 Adding the external accessory framework

Once the external accessory framework has been added one can verify its success in the drop down menu in the left side bar of Xcode name frameworks. Once verified, it is time to take care of the second addition, the Redpark SDK. To get a copy of the SDK one must navigate to http://redpark.com/developers/#sdk from any web
browser and fill out the form to download it. The SDK will be included in the files for this project in case there are future issues with the web site. Once downloaded simply drag and drop the files into Xcode’s left side bar. A menu will slide in from the top just select ok. The Redpark SDK requires the external accessory framework, which was added in the previous step. Now for the last step to get our iPad talking over the serial cable, Apple requires a code signature so that the iPad knows what cable it is communicating with. This prevents anyone from making a random cable and plugging it into the iPad. To provide these code signatures one must alter the projectname-info.plist file located in the left sidebar of Xcode. Be very careful with this file as it tells Xcode build instructions and if something is missing Xcode will not build the project. Add one additional entry at the end of the file of type arrayto make it work. The key field of the entry should be named Supported external accessory protocols. The array should hold two strings the first is com.redpark.hobdb9v and the second should be com.redpark.hobdb9. These two strings represent both cables produced by Redpark. This entire process can be viewed in figure 54.
Fig 54. Adding the code signatures.

Once all is set up the project can be built for the iPad and the iPad will be able to use the Redpark serial cable. The Xcode project is attached so if desired one can view the Objective C code.
8. Conclusion

The objective of the project was to make the iPad successfully send on and off commands wirelessly. This project has covered simple electricity principles for AC and DC circuits along with a basic introduction to iOS accessory development. An Arduino board with XBEE shield for wireless transmission is used at both ends.

To recreate this project it is of the utmost importance that the progression of this document be followed. After following this detailed guideline, one should have a comprehensive understanding of how to develop projects with the Arduino in combination with the iPad. Beforehand, a clear understanding should be obtained of the inner workings of the Arduino and how to develop electronic circuits for this device.
References


[6.] https://www.sparkfun.com/tutorials/192#defaults