Android Solar Panel Documentation

Sean Hurley
Soo Woo
Tim Park
Aaron Bercovici
Jim Bright
Max Chang
Nikita Rathi
Mike Gonda
Mike Candido

Android Solar Panel Documentation

Description

This project aims to build an Android app capable of communicating with solar panel management systems via bluetooth. The key users of this app will be in the developing world where small solar panels are commonly used. These people need to know different things such as battery charge level, battery charge rate, and current settings for charge levels. They would also want to be able to set various settings for the battery charging. All of these features would allow them to get more out of their solar panels. The end goal is to have a finished product that can be released to these people so that they can get more out of their panels and hopefully improve their lives.

Development Method

Iterative Development

For our project using an iterative development process afforded us two major advantages. The first is that it allowed us to focus on a specific part of the project rather than having to deal with it as a whole. By decomposing the project into separate user stories, we were able to add new functionality at a rapid pace without needing to worry about any interactions with future work. This also allowed us to efficiently assign group members to tasks according to how much effort we thought that it would take to complete.

The second advantage an iterative development process gave us was the ability to easily modify previous code as needed. Because extra time was not spent adding unneeded functionality, we did not have to worry about breaking most of the existing code every time we made a change. Instead we could change the existing code in order to add the new functionality and then run regression tests to make sure nothing else was affected.

Pair Programming

Pair programming, or pairing, played an important role in our development. Pairing is a technique where two programmers work together using one machine for programming. This practice proved very beneficial over the course of our project.

We broke our project into several different components. These components include the UI, a mock bluetooth panel, and the communication code which links our app to the mock panel. By pairing we were able to divide these different aspects of our program among small teams. These teams then had a small portion to work on which was separate from the rest of the app. Pairing during these times of development also let people dive deeply into a particular aspect of the app. This ensured there were at least two people familiar with each part of the program.

Team Meetings

After doing pair programming throughout each iteration, we had to combine the work of each subgroup. For this reason, we had two meetings each week. These meetings all followed a similar structure. The first thing on the agenda was having each person talk about what they did during the iteration. This step encouraged everyone to be on the same page and helped ensure no one got left behind. After this step, we then broke up the next series of tasks. As mentioned before, we assign each new task to two people for the purposes of pair programming.

On top of our small team meetings, we also had client meetings every other week. These meetings were very important to us, as they allowed us to design our app to closely match our client's expectations. We started off these meetings by doing a short demo of the current state of the app for the professor. This step was vital since it allowed the professor to get an idea of the app's look and feel. After the demo, we got feedback from the professor and his ideas of how the app should be different. We left these meetings with a good idea of what our next steps in the project should be.

Version Control

We used Git for our version control, which was a better choice than SVN for a number of reasons. The most important of these is that the branching in Git is significantly nicer than in SVN. Our workflow of having several different groups working concurrently on our project required us to have good branching strategies to avoid conflicts. The branching strategy we use is outlined here. This model is successfully used throughout the industry. It allowed us to easily work on different components and combine them later on. Using github also had the advantage of providing issue tracking for us.

Refactoring

During the entire course of the project, it was important for us to constantly refactor our code for many different reasons. Working in teams of two, we were constantly switching roles as well as extending and maintaining the code, so it was important to be able to figure out code written by someone else. We were all working towards the goal of being able to hand off the code to the client and refactoring helped in getting rid of different code smells we found in our code making it flexible for further extensions.

Testing

Test cases actually played a somewhat minor role in our project. This is primarily because we spent the majority of the project learning how to develop for Android. Our interfaces are very simple, so our GUI testing did not need to be automated. However, we did have extensive testing for our Bluetooth communication. For this we wrote JUnit test cases that test the entire asynchronous lifecycle of the communication framework. This allowed us to test the heart of our app since it is what powers the data and sends it to the GUI.

Requirements & Specifications

User Stories

View Real-Time Data on Voltage and Current

The user uses the app to ask the control box for the current voltage and current data, and the control box sends back this data. The app then displays these numbers.

View Historical Data on Voltage and Current

The user uses the app to ask the control box for historical information on voltage and current, and the control box sends back this information. The app then presents a line graph of the data over hourly, daily, and weekly scales.

View List of Scheduled Events

The user uses the app to ask the control box for the daily events it has scheduled, and the control box sends back this information. The app then shows a list of the start times and durations of scheduled events, sorted based on start time.

Schedule a New Event

From the list of scheduled events the user attempts to add a new event. The user chooses the start time and duration of the event. If the event doesn't conflict with already scheduled events the app schedules it on the control box, and updates the list.

Remove a Scheduled Event

From the list of scheduled events the user selects an event to remove. The app then removes the event from the control box's schedule, and updates the list.

Set Minimum and Maximum Charge Levels

The user chooses a minimum and maximum charge level percentage. If these constraints are valid, the app sends them to the control box. The control box then attempts to keep the charge level of the battery between the specified levels.

Set a Password for a Device

The user chooses a password for a particular control box. The app then sends this password to the control box. The control box will reject subsequent attempts to communicate if the messages do not include this password.

Architecture & Design

Bluetooth Communication API

General Message Format

```
{
        type: <string>,
        pin: <string>,
        ...
}
```

Responses will always contain a result code indicating errors. A code of 200 indicates everything went alright, 404 means request not understood, and 403 means forbidden. Most requests carry a pin field for authentication. The initial pin is built into the controller and can be changed by the pin-update request.

Handshake

Verify that the device is a compatible control box device.

Request

```
{
        type: “handshake”
}
```

Response

```
{
        type: “handshake-response”,
        result: <int>
}
```

PIN Update

Set a new PIN for the control box.

Request

```
{
        type: “pin-update”,
        pin: <string>,
        new_pin: <String>
}
```

Response

{
type: “pin-update-response”,
result: <int>,
message: “OK”
}

Time/Date Update

Synchronizes the time and date on the controller by updating it with the provided timestamp in milliseconds from the UNIX epoch.

Request
{
type: "time-update",
pin: <string>,
timestamp: <long>
}
Response
{
type: “time-update-response”,
result: <integer>,
message: “OK”
}

Location Update

Updates the location data stored on the controller with the provided latitude and longitude values.

Request
{
type: "location-update",
pin: <string>,
latitude: <float>,
longitude: <float>
}
Response
{
type: “location-update-response”,
result: <integer>,
message: “OK”
}

Device Status Snapshot

Sends a snapshot of the current state of the installation to the phone. “timestamp” is the time on the phone when the request was received. “battery-voltage” is the voltage reading of the battery

in volts, “battery-percent” is an integral number of percentage points that represents the current percent of the battery filled, “panel-current” is the current reading of the solar panel in amps, and “panel-voltage” is the voltage reading of the solar panel in volts. The request is initiated by the phone. “intake” is x, “outtake” is y, “min” is the lower limit on the battery percentage, and “max” is the upper limit.

Request

```
{
        type: “snapshot”,
        pin: <string>
}
```

Response

```
{
        type: "snapshot-response",
        result: <integer>,
        message: “OK”,
        timestamp: <long>,
        battery-voltage: <float>,
        battery-current: <float>,
        battery-percent: <int>
        panel-current: <float>,
        panel-voltage: <float>,
        intake: <float>,
        outtake: <float>,
        min: <int>,
        max: <int>
}
```

Update History

Retrieves all historical data currently stored on the controller as a list of snapshot objects (see above).

Request

```
{
        type: “history”,
        pin: <string>,
}
```

Response

```
{
        type: “history-response”,
        result: <integer>,
        message: “OK”,
```

history-data: [snapshot]
}

Update Schedule

Modifies the schedule of the controller. To schedule a new event (at this time, an event just means turning on the power), use “schedule-event”. “name” is a tag for the event, “first-run” is an absolute timestamp in milliseconds from the Unix epoch that identifies the first time this event should occur. “duration” is the duration in milliseconds until the event is over and the power turns off. “interval” is the duration in milliseconds until the event repeats (from the last time it began). “unschedule-event” simply deletes the event associated with the given identifier.

Request

{
type: "schedule-event",
name: <string>,
pin: <string>,
first-run: <long>,
duration: <long>,
interval: <long>
}

Response

{
type: “schedule-event-response”,
result: <integer>,
message: <string> (the assigned id)
}

Request

{
type: "unschedule-event",
pin: <string>,
id: <string>
}

Response

{
type: “unschedule-event-response”,
result: <integer>,
message: “OK”
}

View Events

Returns a list of all currently scheduled events

Request

{
type: “events”,
pin: <string>
}

Response

{
type: “events-response”,
result: <int>,
message: “OK”,
events-data: [event]
}

Battery Charge Levels

Set and get battery charge constraints

Request

{
type: “set-charge-constraints”,
pin: <string>,
max: <int>,
min: <int>
}

Response

{
type: “set-charge-constraints-response”,
message: “OK”,
result: <int>
}

Request

{
type: “view-charge-constraints”,
pin: <string>
}

Response

{
type: “view-charge-constraints-response”,
result: <int>,

```
        message: “OK”,
        max: <int>,
        min: <int>
}
```

UML Diagrams

Diagram for connecting to a device and syncing snapshots and history

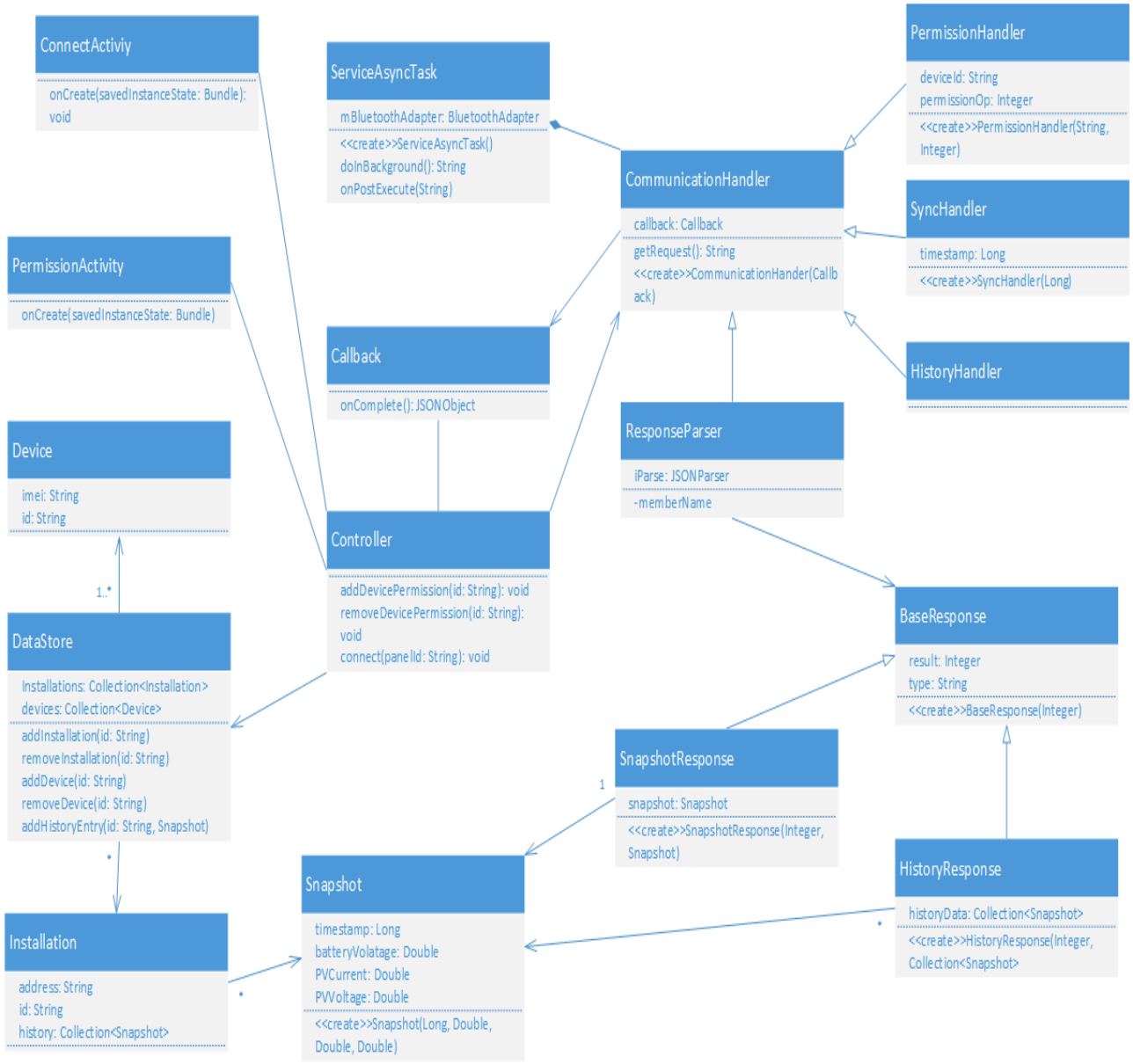
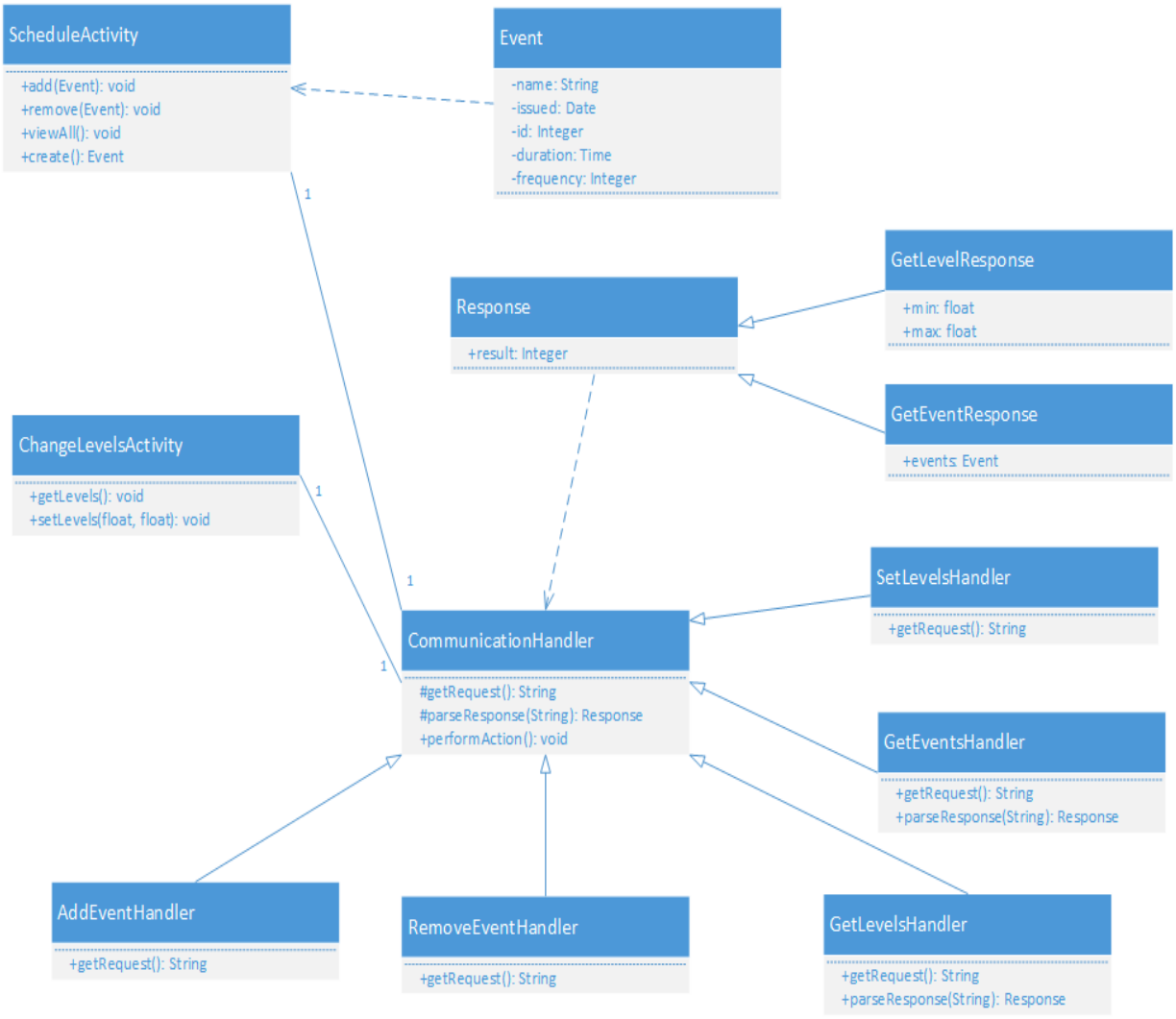
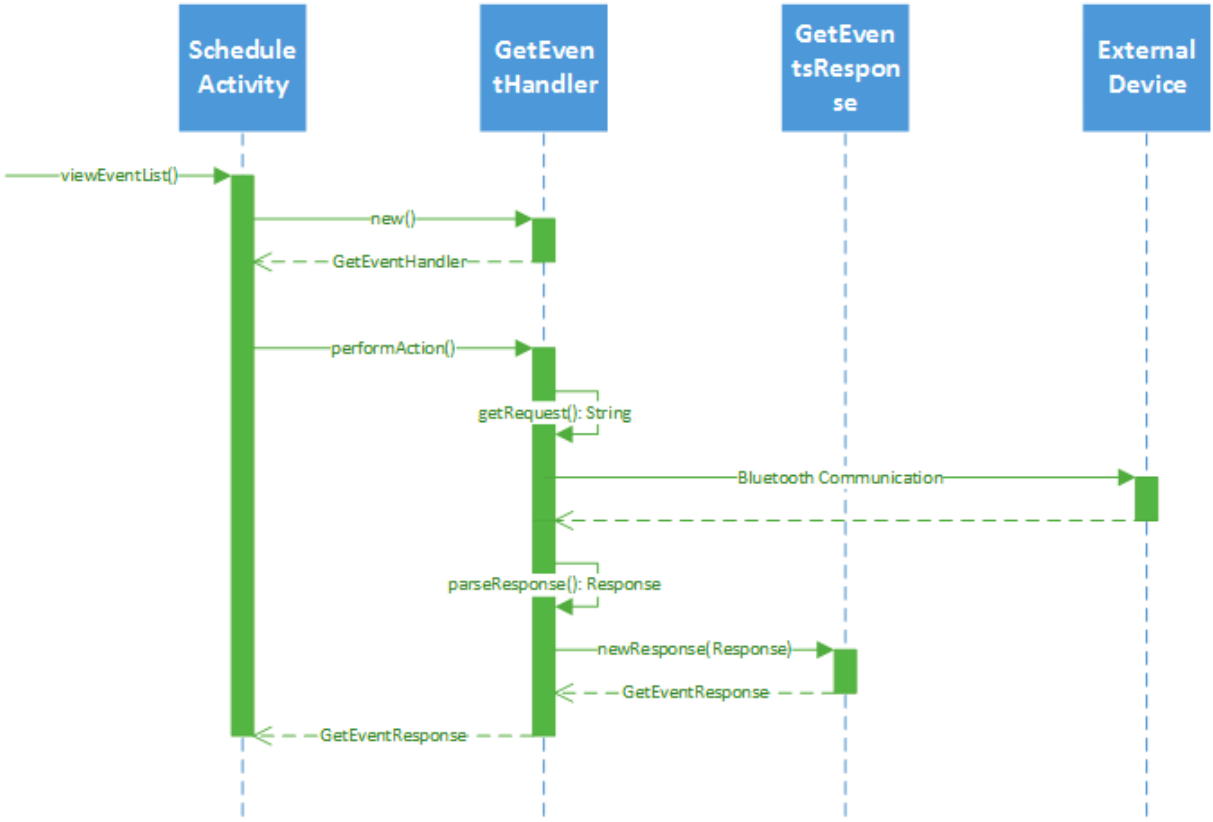


Diagram for getting/setting events



Sequence Diagrams

Sequence Diagrams for getting events. This shows how the overall communication flows throughout the app.



The Big Picture and Impacts of the Android Framework

The Android framework played a huge role in our overall project structure. This is the case whenever you write an app on some sort of framework, and Android is no exception. The most noticeable impact is how the files within our project are laid out. Android is an MVVM (Model View View-Model) architecture which has a general structure which often applies to apps built in it. The first aspect of this structure is that the GUI is kept in an entirely separate folder, and is also written in XML. This is done so that the interface is kept separate from the control code which makes changes easy. The XML files which include the GUI and different assets and resources would be the “View” portion of the app.

Following the MVVM style we also have several different classes which model the data that our app must display. This is the “Model” layer. The communication framework we designed ties in heavily to the model and is the only place we ever create the actual instances of the model.

Our Bluetooth communication framework is the meat of our app. We also put a great deal of work into the design of this framework. The app will be sending and receiving many different messages that will all have a similar flow of control, and similar message contents. When we realized just how similar the functionality would be, we realized that it would be a perfect use for the template method. The template method is a design pattern where a programmer defines a skeleton algorithm which contains the majority of the code and overall flow, but it calls abstract methods wherever the behaviour differs. For our app, the behavior only differed in two places:

what data to send and how to parse the data received. This allowed us the put all of the logic for sending arbitrary data and waiting to receive more data over the Bluetooth protocol into the skeleton template method. Then whenever we needed to add a new type of message, it was as simple as adding two short methods within a class and the rest of the logic was taken care of for us. This allowed us to reuse a lot of code and have the core logic in one central location.

The one layer of MVVM we haven't talked about is the "View-Model" layer. This is referred to as "Activities" in Android. Activities would often be considered controllers in the more common Model View Controller paradigm. These classes are where we tied together the XML GUI into our actual model and the Bluetooth communication framework. They worked as the glue which holds the app together.

Overall, our app is structured the way it is solely because of the Android framework. The high level architecture is a little strange and confusing at first, but this is how every Android app is structured and is the norm for these projects. After learning how the MVVM paradigm works, it actually makes a lot of sense since the separation of the different components encourages better code overall.

Major Design Components

Communication Protocol

As mentioned previously, the communication framework is an important part of our codebase. The main class responsible for this is `CommunicationHandler`. This class is what contains the template method which has the core logic of the Bluetooth protocols. In this class there is an `ASyncTask` which contains the actual template method. An `ASyncTask` is a class in Android which is for doing asynchronous work and having it complete back in the main UI thread. We define the method `doInBackground` within that `ASyncTask` to have the core logic of the template method. The template method first calls the abstract method `getRequest` which will return the properly formatted `JSONObject` that holds the contents of the message to send. The template method sends this string and then waits for a response. After receiving a response it then calls the abstract method `parseResponse` which will properly parse whatever response is returned for this type of message. After the response is parsed, the `CommunicationHandler` then sends the result back the main UI thread via the interface we created called `Callback`. The `Callback` interface only has one method, `onComplete` which has a parameter containing the proper response object. In order to use a particular `CommunicationHandler`, all you need to do is construct it with the proper parameters (including the `Callback`), and then call the method `performAction` which will run everything in the background so that it does not block the main UI.

Graphing Library

For the graphing feature of this app, we chose to use the popular Android graphing library called "AChartEngine". Additional documentation can be found here and additional tutorials can be

found on YouTube. Please note that this library is used within the `getView` method of the `LineGraph` class.

A reason we chose to use this library is because it supports various chart types, including line graphs. However, if the client wishes to change the chart type, there are many others to choose from (line, area, scatter, time, bar, pie, bubble, range, dial, etc). Another reason we chose this library is because it is well optimized and can handle and display numerous values. Both the versatility and efficiency of the app makes for a reliable graphing library.

Future plans

Future of the Code

This project was sponsored by a Professor in the ECE department. So after we are done with this semester we will be handing it off to him. We spent the last week of the project focussing on refactoring and cleaning up the code so it would be ready for the handoff. In order to do this handoff, we will have a handoff meeting, strong documentation, and instructions for the professor in the future. The handoff meeting will be the key. This is where we will give him access to all of the code and anything related to this project. During this time we will sit down and walk through the code with him to make sure that he has a solid understanding of how the app is structured. This documentation will also be very valuable to him as well since he can refer to it for the instructions of how to do anything he will need.

After this semester, our entire team will no longer be around, so we will not be able to continue working on the project. However, the professor will have contact information to reach us. We will use this to keep an open communication channel with the professor as the project continues to develop. This will allow us to be filled in as our work proceeds as well as let the professor ask us any questions he might have.

Personal Reflections

Sean Hurley

Overall this project was a good real-life experience of working in a larger group for software development. It was very useful to see how things like version control, pair programming, code reviews, and weekly meetings can have a positive impact on software development. The most useful aspect to me was definitely coordinating between everyone to split up the work. After splitting up the work, bringing everyone back together and merging code as well as keeping everyone on the same page was a good learning experience.

Soo Woo

Working alongside other talented software developers has changed my perspective on large group software development. I have accumulated many software development skills that will effectively transfer to my real-world job. The following has proven invaluable: pair programming, code reviews, refactoring, version control, documentation, testing, and so on. Among the many

skills which I have come to appreciate, code reviewing and learning to use Git have been very beneficial. Git allowed us to work concurrently on different branches and also made merging a breeze.

Tim Park

In contrast to all of the other classes we have had this project taught me the what all is involved when working with a larger group. Having to coordinate with eight other programmers about what needs to be accomplished and how best to go about doing so meant that a lot more time and effort had to go into group communication. Being able to use Git so that everyone is able to work independently and then easily combine work saved us a lot of trouble even over other version control software such as SVN. The ability for everyone to concurrently edit Google Docs also allowed us to quickly reach consensus on issues and then modify them as changes are needed.

Mike Candido

This project has changed my perspective on what it means to work on a larger software project. Working on a team requires an entirely different process than developing software on your own, and this semester has provided invaluable experience working with that process. Furthermore, I got experience with areas of software development like version control, documentation, testing, code maintenance, and refactoring. On top of it all, I had the opportunity to work on a meaningful software project that has the potential to help people in developing countries and deploy new technology developed here at the University of Illinois. Overall, it has been a rewarding and fulfilling experience that I am thankful to take with me as I begin my career.

Mike Gonda

Working in the largest group I ever have has taught me a lot about software development on a grander scale.I also had the chance to learn a lot more about Android, Git, and working to implement a client's idea. This project was an excellent experience, applying the skills I've learned here at UIUC to something more purposeful than typical class assignments. It was great to work on something that, hopefully, will have a positive impact in people's lives. I hope our client perseveres in realizing his vision, and I was happy to play a part in its beginnings.

Aaron Bercovici

Like last semester, our project was an interesting experience in working in larger groups. This time around it was more enjoyable thanks to a more interesting problem domain, and competent leadership from Sean. Process-wise we did a decent job of planning out task and pulling things together at the end. It would have been advantageous to employ more rigorous code reviews, as we had to spend much of the last milestone refactoring and writing comments. Over the course of the project we could have communicated more with our client. Nonetheless, working on a real-world problem with an actual customer was quite exciting.

Max Chang

The most useful thing this project taught me is how to interact people in a large group.I enjoy the working experience with other eight talent software developers. We discuss the problem together and figure out how to solve it like a real software team. Besides the teamwork process,

the project itself is also interesting and challenging - it is the first time I touched with a project dealing with real world problem. I hope I can do a better job and help my teammates more next time.

Nikita Rathi

Overall I felt like this project was a taste of what it would be like in the real-world working with a large team having divided up all the tasks. This was the first time for me working with a client towards reaching a consensus regarding what user stories are feasible and what way to implement them. Overall, it was a great experience and I am going to be taking away a lot from this project.

Jim Bright

By far, this was the most successful group project I have ever worked on. From the very beginning, the group meshed extremely well. We all had past experience and unique knowledge that we shared with everyone. Without Sean's guidance and familiarity with Android and Git, I don't think this project would have been nearly as successful. I was excited to work on a project that would see life beyond the end of the semester. For the first time, I felt that the quality of my work would need to last beyond the grading deadline. I feel I have gained invaluable experience with XP and working with large groups to accomplish seemingly insurmountable tasks.

Appendix

Setting Up the Program

First you need to set up the Android development environment. This will vary based on your operating system. Follow the guide from Google here for more detailed instructions. Be sure to have API levels 8 and 16 installed.

The next step in setting up our app is getting the code and setting it up in the development environment. Follow these steps:

1. The code is available on github via this URL. To checkout the project do `git clone https://github.com/SeanHurley/AndroidSolarPanelManager`
2. Now to initialize the Action Bar Sherlock submodule, `cd AndroidSolarPanelManager`. Then run command `git submodule update --init`.
3. Now import Action Bar Sherlock, in eclipse `Import -> Android -> Existing Android Code Into Workspace -> Select ActionBarSherlock/library`.
4. The above command will checkout the code into the current working directory. Now, we need to import the projects into Eclipse. In order to do this, in eclipse `Right Click -> Import -> Existing Projects into Workspace -> Select BTTest and SolarPanelApp`
5. Now both of the projects should show up in the Eclipse workspace and build properly.

Setting up the Android app portion of our project is very straightforward. The steps are as follows:

1. Enable debug mode on your device. This varies with different versions of Android, but is easy to find online.
2. Connect the device to your computer and it should recognize the device.
3. Now in Eclipse `Right Click -> Run as -> Android app`, and select the device you just connected and it should install no problem.

If you would like to run the mock bluetooth panel, things get a little more difficult. If you are running a version of linux, it should work just fine out of the box. This of course depends upon having the proper bluetooth drivers installed. If you are running Mac OSX, you will need to start eclipse with a special script. Do the following:

1. In the `extrafiles` folder there is `bteclipse.sh`. Edit the last line of the file to point to where ever you have Eclipse installed.
2. Run `./bteclipse.sh` and eclipse should start.
3. In `BTTest`, you will need to create a special run configuration. `Right Click BTTest -> Run as -> Run configurations -> Click the new icon -> add -d32 to the VM arguments`.
4. Now you can run this run configuration and the app should work properly.

You should now be able to run both the mock panel and the Android app together.