Twist on Time

Detailed Design

Team 0336

Noah Boerner
Nikhil Bose
Peyton Howell
Farouk Marhaba
Fuad Youssef

Table of Contents

List of Figures

1. Introduction

Project Overview

The purpose of this project is to develop a timer plugin for that Android Team Awareness Kit (ATAK) application. ATAK is a plug-in based application currently in use by the military that allows plug-ins for different tools such as maps and camera feeds to be housed in the same application. Twist on Time will be implemented as a plug-in that interfaces with the rest of the tools in ATAK. Often in strategic missions and training exercises, commanders in the 75th Ranger Regiment of the Army need to manage many asynchronous timelines. Without an easy way to keep track of this, these commanders often build-up stress managing these timelines. By working with the 75th Ranger Regiment, we will create a customized timer plugin that will allow commanders to keep track of multiple timers at once. This will reduce the stress on the commanders and make it easier for them to focus on other important aspects of their mission.

Our Solution

In order to make managing multiple timelines easier for the 75th Ranger Regiment, we are developing a plugin to the ATAK specifically to manage multiple timelines. Our solution will allow commanders to have all the information they need to properly manage their teams without adding additional stress or hassle. Since commanders are already using the ATAK application to manage their other teams, integrating a timer plugin will allow them to manage their timelines without having to switch between applications or have another external device. In addition, we will be adding multiple features to enhance the speed in which timers can be set, along with enhanced customization of those said timers. To enhance the speed at which timers will be set, we will create a “Preset Timer” feature, so timers can be set with a single button. To allow for more customization, we will have a variety of notification settings and naming features to allow timers to be more recognizable and distinct from one another.

Our project is built within Android Studio, and the finished product will be a plugin within the ATAK application. We are using a CIVTAK SDK for plugin development, and it is being developed on version 21 of the Android SDK. Timers are created on a screen within the plugin, which creates a .json object with the parameters. A system call reads the data from the .json object, and a timer is then created using Android system resources. The main components for our plugins include notification banners and sounds, creation of synchronous timers, and simple and effective context switching.

Summary of Document

In this Detailed Design Document, we will introduce our overall system architecture and how our plug-in solution embeds with the current ATAK Android application. We will also discuss our data storage and transmission design, highlighting how our solution retrieves timer details and displays important information across the ATAK application as well as the Android Operating System. Our team will present a detailed component design that will demonstrate the static and dynamic components of our plug-in, in addition to a user-interface design highlighting the core screens and interfaces of our solution.

2. System Architecture

Our System Architecture

We are building our project as a plugin within the Android Tactical Assault Kit, so naturally much of our systems architecture will come from the confines of the preexisting app. With that in mind, our architecture will generally follow the Model-View-Controller (MVC) architecture, which is the current standard for Android development. The way MVC works is that there are separate files to handle UI, user input, and background logic. What this means for us is that we will have separate files to determine how the screens of our app look, how input into those screens is handled, and how objects like the timers work. Using the MVC architecture will allow us to easily find documentation online about best practices and will make it easy for future developers to understand and expand on our work.

Each of our screens will be represented by a view, which in our case are layout files written using XML. Controllers allow users to input data and navigate between these views, and in response to these will edit our models. These controllers will be based on ATAK's implementation DropdownReceiver class, which serves as the equivalent of an activity in the context of a plugin. These DropdownReceivers inflate an XML layout file to appear on the screen and accept user input.

Separate from DropDownReceivers, we will also have controllers to actively track our models using system threads, including when notifications need to be sent at certain times. Our models will represent our data, namely our timers. These timers will be stored can be stored in both system memory and system storage. They will be stored in system memory when running and will be stored in system memory when saved to be used later. Our timers will run separately from the DropdownReceivers as they will run on a system thread. This will allow the timer to run even when the ATAK application is the application open on the user's screen.

Static Elements

The Static Components behind our program are the Plugin Lifecycle, XML files, DropdownReceivers, and Timer Object. As our architecture follows the MVC pattern of Android, it can be understood that the XML files are the static views whose controller logic is located in the DropdownReceivers. All of which interact with the Timer Object which serves as a model. More detail for each component is provided below:

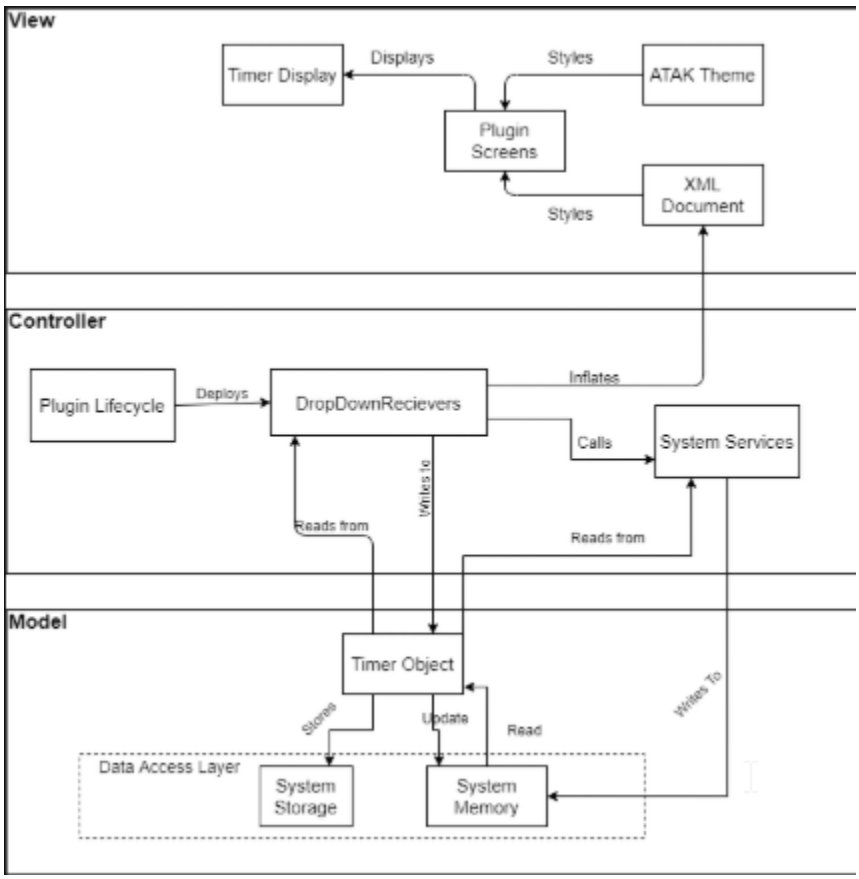

Figure 2.1 Model View Controller Diagram of Our System

A breakdown of our system architecture is listed below:

- Plugin Lifecycle
  - This is the memory/runtime environment in which all of the plugin operates
- XML Layout Files
  - These layout files serve as the views of our architecture and contain all of the UI elements of our plugin.
- DropdownReceivers
  - These DropdownReceivers contain all of the logic for handling user interaction with the XML files as well as passing data to our memory.

- Timer Object
    - The timer object is the one model we have in our architecture, and it contains all of the relevant information for a timer such as:
        - Timer ID
        - Timer Name
        - Duration
        - Sound
        - Reminder Notification
        - Whether or not the timer is a preset

Dynamic Elements

The Twist on Time Plugin is nested in the ATAK application, and through it users will request their timer through the system resources. The user navigates to the plugin through the ATAK toolbar and opens the plugin through there. The screen in inflated through a Dropdown Receiver that is standard within ATAK plugins, and the user is presented with the main view of the plugin. Any active timers are displayed on the home screen, and can be tracked, edited, and cancelled at any time. Users can also click a button that takes them to a page to create a new timer, or another button that takes them to a list of preset timers that can be conveniently used. When a user begins creating a new timer, a .json file is created that represents the different attributes of the timer. As the user changes the values in the fields on the screen, the changes are reflected in the .json Timer Object file. Once the .json file is finalized, the DropdownReceiver calls the System Service to read the file to create and run a timer based on it. This System Service continuously writes to system memory, which the Timer Object reads from. The Timer Object is referred back to the Dropdown Receiver which updates the view of the main screen, where it is visually represented for the user.

Figure 2.2 is a representation of a use case where a user navigates to the create timer screen and creates a new Timer object for use. First the user opens the plugin. From there the corresponding view sends a request to the main the dropdown receiver that corresponds to the Home Screen. This renders the Home Screen and sends it back to the view. The view then displays the Home Screen to the users. When the user wants to create a timer, the user clicks the create timer button, the view then sends a request for the create timer screen to the dropdown receiver. The receiver then returns the screen to the view which then displays it to the user. From there the user can input data to create the timer. The data is sent from the view to the dropdown receiver when then initializes a timer object that is stored in memory. This object is sent back to the receiver and is used to update the list of running timers. The view then displays these timers to the user.

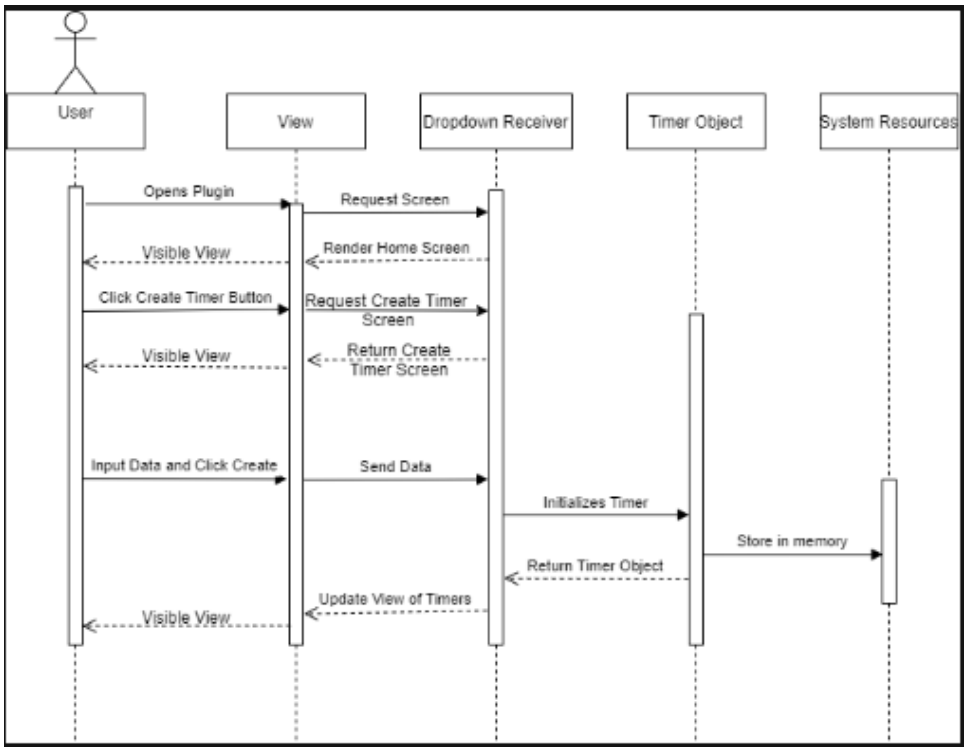

Figure 2.2 System Sequence Diagram for a User Creating a Timer.

3. Data Storage Design

For our data storage, we only needed to save data locally as per the requirements of the project, so we are writing all necessary data to system memory. In Android this is typically done by creating a new file and writing to it when necessary and upon closing the application, or in this case the plugin. Data from this file can be read at any point the data is necessary and upon starting the plugin. Google provides a native JSON reading/writing API called GSON that greatly simplifies writing data to storage. GSON is therefore considered the standard for writing to memory on Android. Because of its widespread use and reliability, we thus opted to use GSON as the format for writing data to our files.

Our timer application is mostly self-contained. The only time we will need persistent data storage is when we save preset timers and running timer. In the case of preset timers, when a user creates a timer and indicates that the timer will be a preset, we will need to save the timer in the Android device's storage. To do this we will store timer as JSON objects and write these objects to the device. These JSON objects will be in the form shown in Figure 3.1. When a user navigates to the preset timer screen, we will populate the preset timers by reading from these JSON files. Doing this will allow the plugin to keep track of all the preset timers even when the device is closed. This will allow us to keep track of all the preset timers created – even across sessions. We are

also planning use the same JSON objects from Figure 3.1 to save information about currently running timers. This will be used in the case when a user closes the ATAK application when there are timers running. We plan to save these timers in the JSON so that when a user reopens the ATAK app the timers resume counting from where they were before.


```
[
 {
  "ID": 1,
  "Name": "Timer 1",
  "Duration": "1:30",
  "Notifications": [
   "At time of event",
   "1 minute before"
  ],
  "Preset": "true",
  "hours": 0,
  "miniutes": 1,
  "seconds": 30,
  "lastUpdateTimeStamp": "1:30"
 }
]
```


Figure 3.1 Timer JSON Sample for Presets and Running Timers

The primary JSON in Figure 3.1 we will be using stores the necessary data for a given timer. The ID field will not be presented to the user but will be used in the architecture to uniquely identify a timer since a name that a user gives a timer may not be unique. We then have duration, a string field that will be formatted to show the remaining time of a timer. This contains similar information to the hours, minutes, and seconds field, but differs in its use. The duration field is presented to users for display, but the integer hours, minutes, and seconds fields are used by the program to update the duration field as well as determining how much time is left. We also have notifications, which is a nested JSON of strings representing when a notification should be executed. These strings can be translated to integer time values for the system to use along with strings for the user to reach. The last field visible to the user in some manner is the preset field, which simply determines whether the timer can be re-used or not. Finally, we have lastUpdateTimestamp. This field exists in the event that the plugin is closed from memory by the user or by the Android operating system. This field will automatically populate in this event so the running timers can be restored when the plugin is reopened.

4. Component Detailed Design

We opted to use Java for Android to write our plugin because of the basic requirement for ATAK that all plugins be native Android apps. As a result of this requirement, alternatives such as React Native would not work for our project. We specifically chose to use Java over Kotlin because using Kotlin would require us to import Kotlin language support for our plugin on top of the Java support necessary for the rest of the ATAK app, increasing the size necessary in memory for our plugin. In addition, considering the rest of the ATAK app and the provided plugin formats were written in Java, we opted to be consistent with our plugin rather than adopt a new standard. Because we are creating a plugin rather than a standalone app, each screen has its functionality determined by a DropdownReceiver class which functions similarly to an Activity in a standalone app. These DropdownReceivers are responsible for transmitting data to other screens via intents, displaying Views containing the necessary UI elements for each screen, receiving data from other screens, and handling the logic of user interaction. The proper way to use timers in Android is to use CountdownTimers which run on the system resources to accurately keep time. Finally, we created an object (Timer) responsible for containing the information related to our timer and another object (ActiveTimer) whose job it is to bind that information to a CountdownTimer and to handle the necessary custom functionality behind each timer such as displaying a notification.

Static Component Design

The static structure diagram of our Timer plugin can be seen in Figure 4.1. We took an extremely basic and intuitive approach to the static layout of our plugin, having a total of only 5 different screens: Home, Preset Timers, Create New Timer, Select Custom Notification Times, and Select Custom Sounds pages.

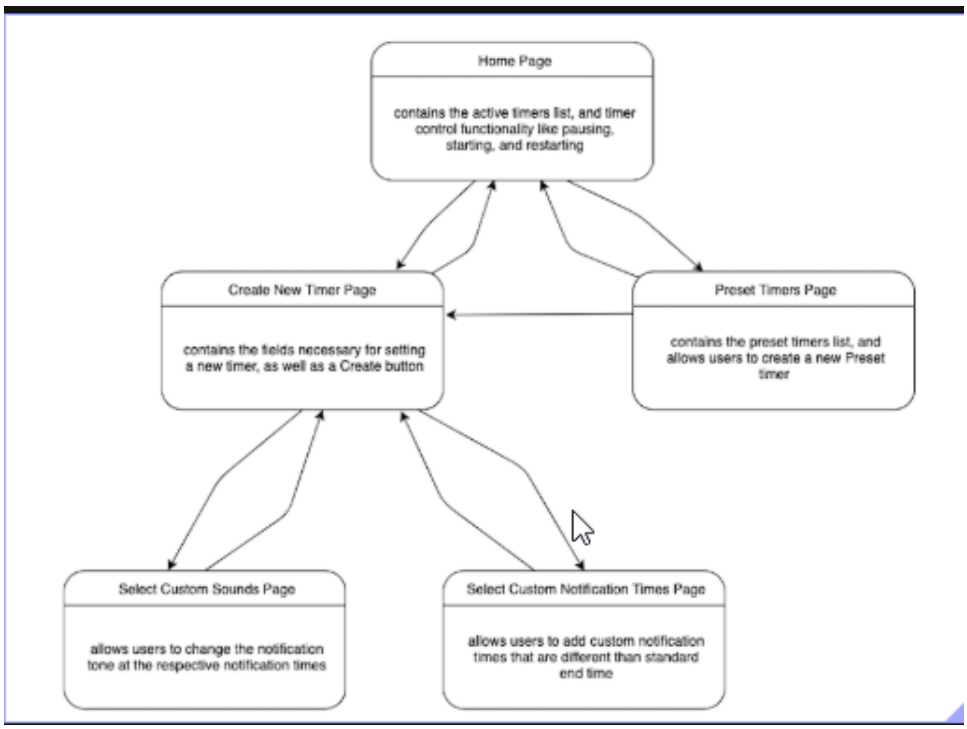

Figure 4.1 Static Structural Diagram of ATAK Timer Plugin

Dynamic Component Design

The dynamic structure diagram of our Timer plugin can be seen in Figure 4.2. In order to prevent unnecessary time-intensive actions, the ATAK Timer Plugin has a set of minimal yet effective actions that users can take at any given time. Actions that users of the timer plugin can take can be observed in Figure 4.2. When a user opens the application, the begin at the Home Page. From there, they can click the “Create New Timer” button, taking them to the create new timer screen. From this screen, a user can access the custom sounds page and custom notifications page. After returning to the create new timer screen, the user can then finish creating a timer, which will cause the timer to be written to JSON if it was selected to be a preset, or simply pass the timer object back to the home screen. The user can also access the preset timers page from the home screen to access the saved preset timers, and activate them, returning the user to the home screen.

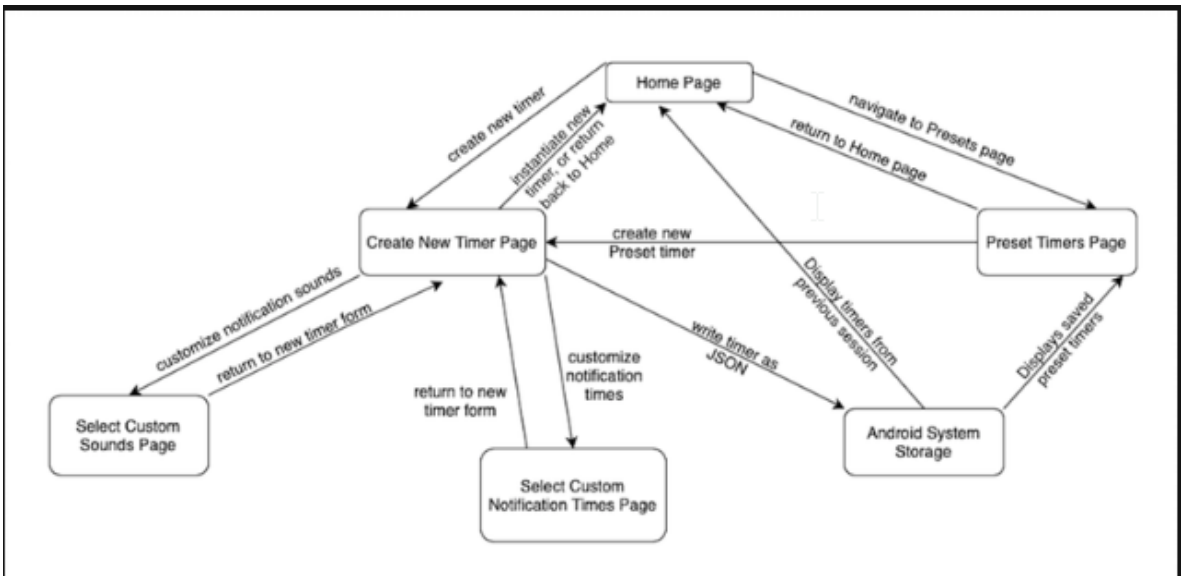

Figure 4.2 Dynamic Component Structural Diagram of ATAK Timer Plugin

5. UI Design

Introduction:

Our app is composed of 6 major screens. The Home and Preset screens are composed of a list of timers as well as relevant timer controls for each timer and buttons for navigating the app. The Create/Edit Timer, Change Sound, Change Notification, and Create Custom Notification screens are primarily responsible for accepting user input as it relates to modifying/creating new timers. In addition to these screens, we also have a notification displaying information about ongoing timers. As is the Android standard, we created each of these screens using Android UI elements in XML. Each XML document corresponds to a DropdownReceiver that handles any logic for user interaction. For all of our UI, we tried to incorporate the CRAP design principles. To do this we made sure to use a similar theme throughout our plugin, made our elements were aligned properly and related elements were close together. We choose not to increase the contrast beyond what is seen in the images below to keep the UI consistent with the ATAK application.

In the section below, we go into more detail about each screen and our design choices for that screen.

Major Screens

Home Screen

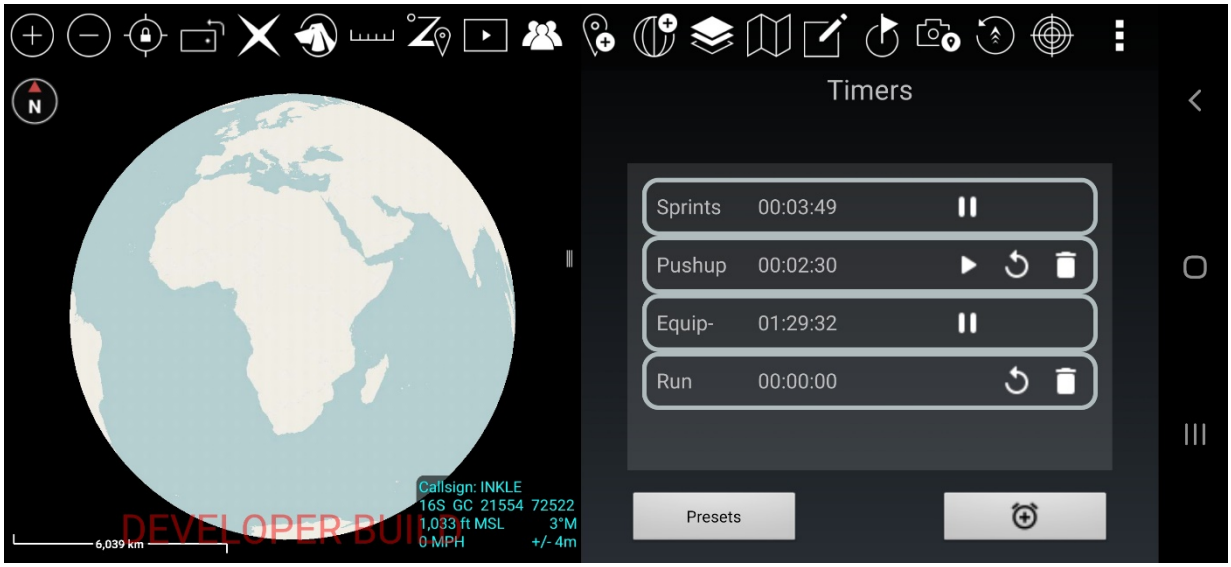

Figure 5.1 Home Screen

The minimalistic Home screen is depicted in Figure 5.1. In the center, there is a viewing window housing all the currently ready, running, or finished timers within the plugin. Each timer can be played, paused, reset, or deleted through the respective buttons within its timer cell. Users can also navigate to the Preset Timers screen as well the Create New Timer screen through the buttons on the bottom of the plugin. In this screen we tried to emphasize the alignment principle by making sure all the information about a timer, such as the timer's name, the timer's duration, and the buttons to control the timer were always in the same spot.

Preset Timer Screen

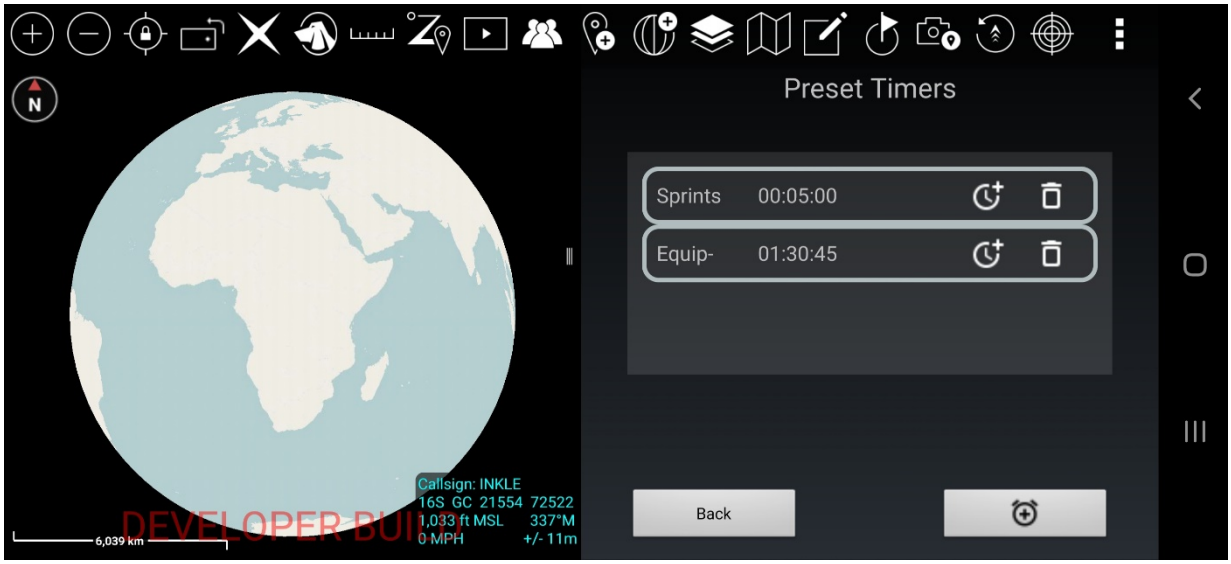

Figure 5.2 Preset Timers Screen

The Preset Timers screen in Figure 5.2 is similar in format to the Home screen; however, it houses the user's preset timers. Preset timers on this screen can be set using the add timer button within the appropriate preset timer cell, and they can be permanently removed as well via the delete button. On the bottom of the screen, users can navigate back to the home page via the back button, or they can add a new timer via the create timer button. In this screen, we took advantage of the CRAP principle of repetition by making this screen very similar to the home screen. We did this to make it easy for users to work with the plugin.

Create and Edit Timer Screens

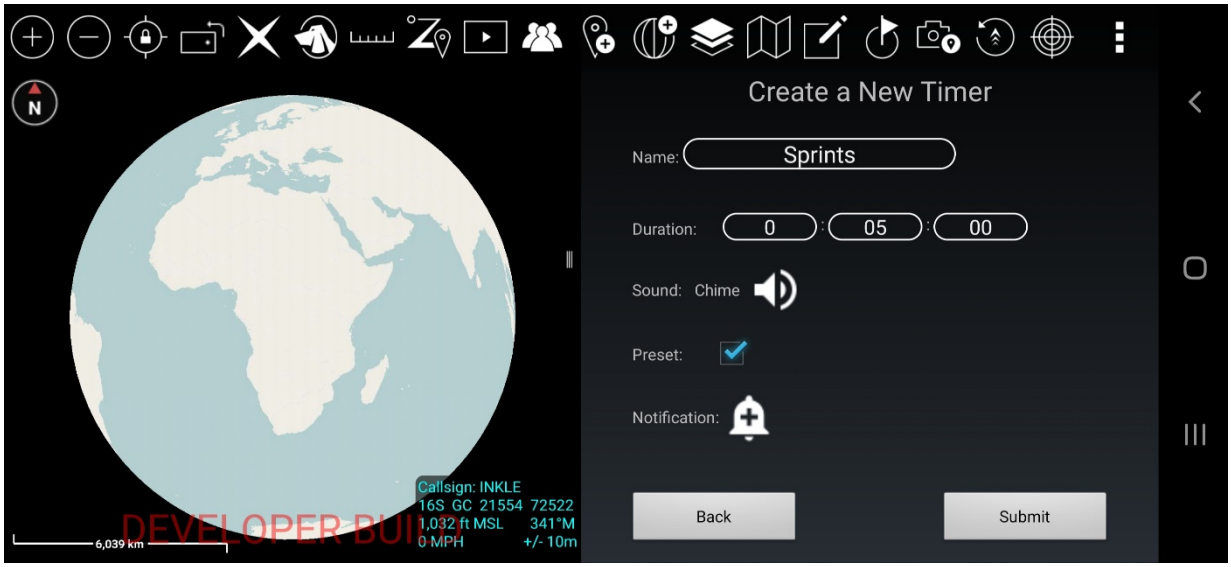

Figure 5.3a Create New Timer Screen

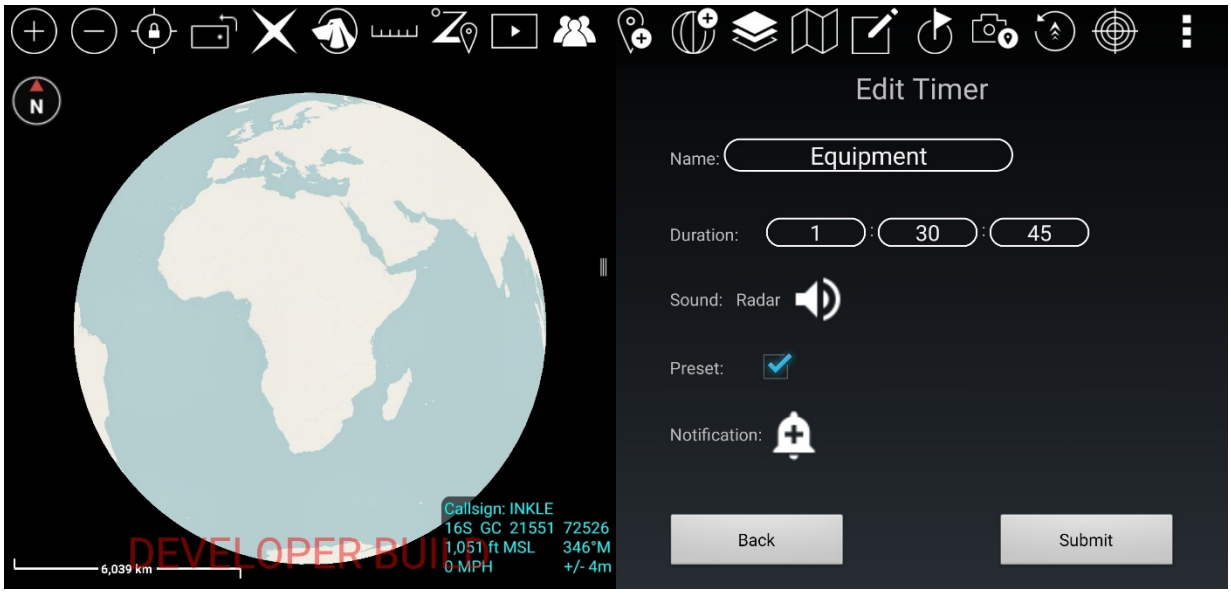

Figure 5.3b Edit Timer Screen

The screen for creating a timer, shown in figure 5.3a, comes up when the user clicks the new timer button on the home screen. This screen is designed to make it easy for a user to create and customize a timer. To accomplish this, we use big text fields and appropriate buttons. We also allow the user to edit preset timers. This is shown in figure 5.3b. For this screen, we tried to keep the UI as similar to the create timer screen as possible. The use of repeated elements across screens that sever similar purposes will make it easier for users to get familiar with and use our app.

Change Sound Screen

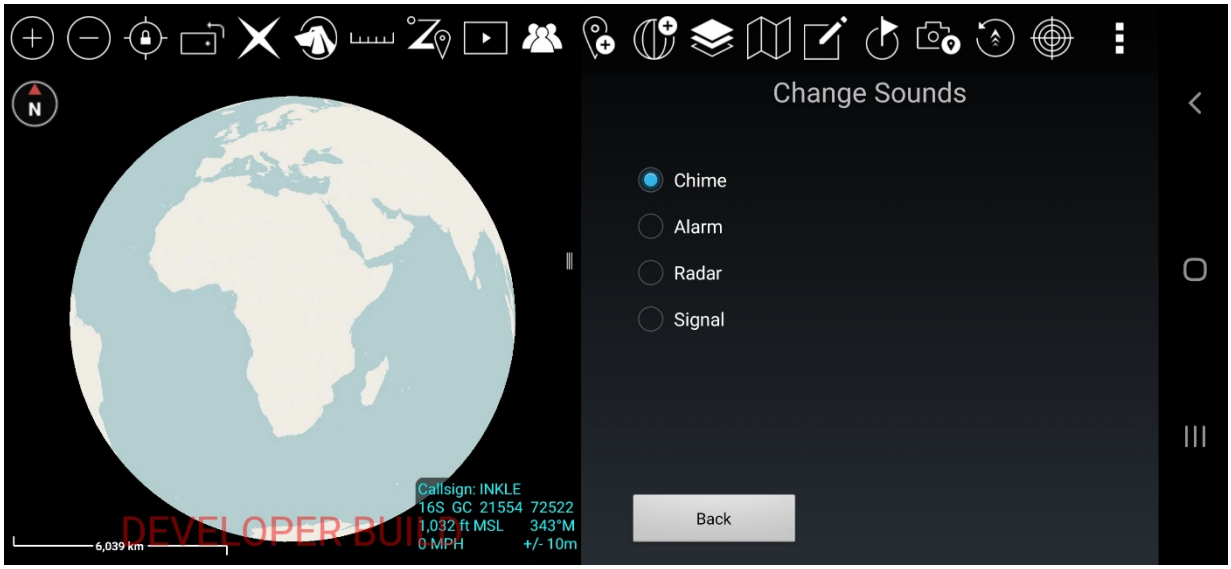

Figure 5.4 Customize Sounds Screen

The customize sounds screen, shown in figure 5.4, is used when the user wants to change the sounds of a given timer. On this page, the user is shown the different options of sounds the timer can make. We use radio buttons to ensure that the user only selects one sound. We have also designed the screen so that when the user selects a sound, that sound plays for the user. We tried to keep our design consistent with similar clock and alarm apps to make sure this UI is user-friendly and easy to understand.

Change Notification Screen

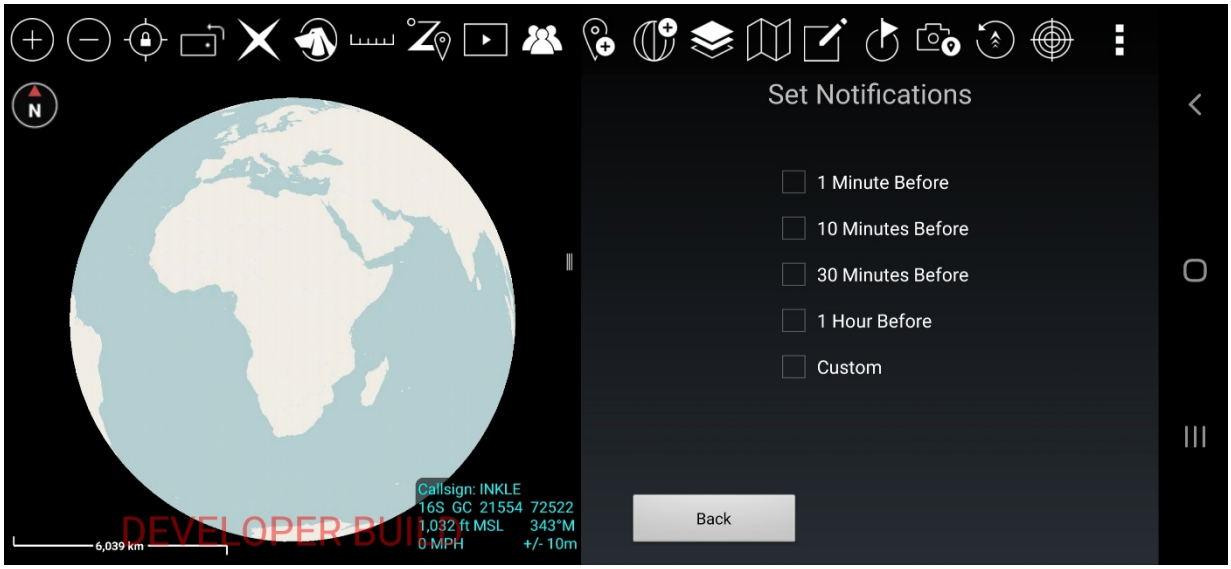

Figure 5.5 Change Notifications Screen

Figure 5.5 shows the screen to change notifications. This menu can be accessed through the create or edit timer screens. On this screen, the user is presented with some default options for notifications. ranging from 1 minute to 1 hour before the event. This screen is designed to resemble the change sound screen for a consistent experience throughout the process of creating a timer. A user can also click the “Custom” option at the bottom of the screen to go to the create custom notification screen, where they will be able to make a different time for the screen. This screen is designed to be consistent with other popular timer and calendar applications which should make it easy to use.

Create Custom Notification Screen

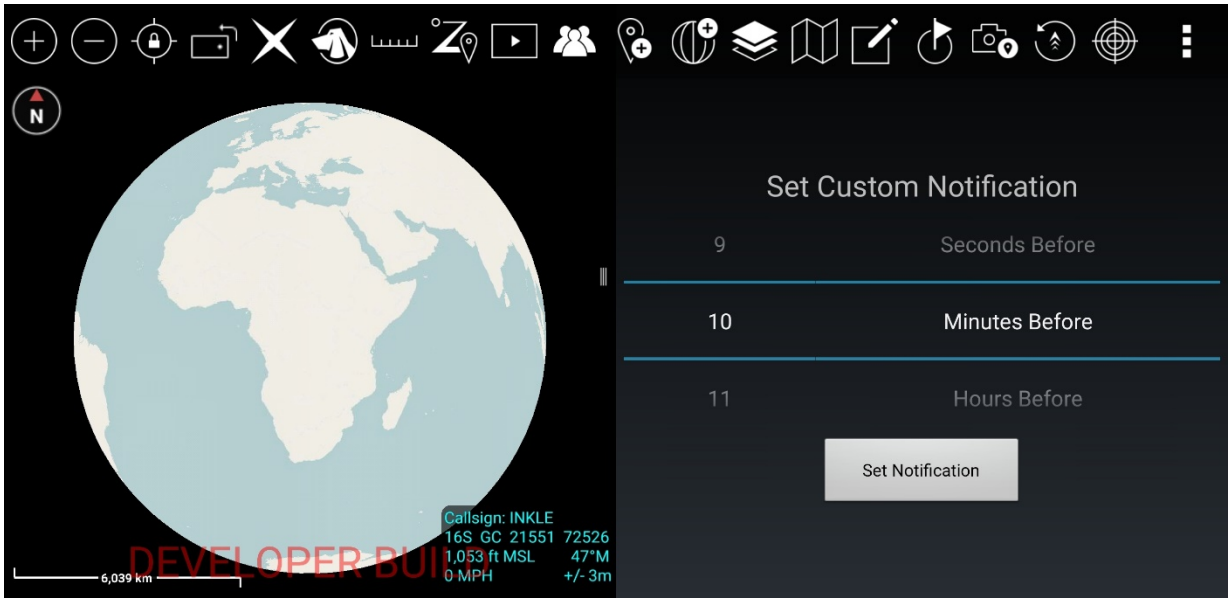

Figure 5.6 Create a Custom Notification Screen

The custom notifications screen shown in Figure 5.6 can be accessed from the change notifications screen. Upon opening this screen, a user can select the seconds, minutes, or hours before that they would like a notification for a given timer. Once the user is satisfied with their selection, they can press the “Set Notification” button which will create the custom notification, as well as redirect them back to the change notification screen. In this screen, we used the high contrast blue lines to make it clear what notification will be created when the user hits the set notification button.

Notification

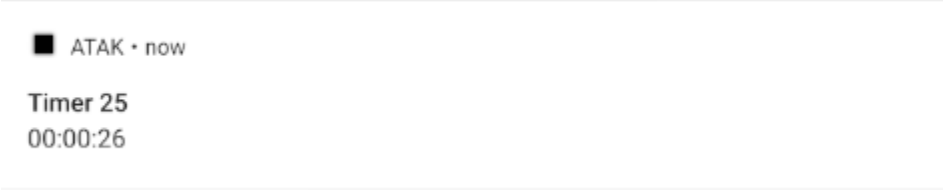

Figure 5.7 Notification

The notification in Figure 5.7 appears for each timer at the times specified when creating that timer, in addition to the sound playing that the user selected. The notification includes the title of the timer, in this case “Timer 25” along with the remaining duration. The notification will also indicate whether the timer is paused and will be automatically dismissed if a timer is restarted or dismissed. Most of the design was within the constraints of the Android notification API, which does not readily allow for much UI changes by the developer. For this reason, we decided to keep the notifications easy to read and consistent to a standard notification for ease of use.