Developer Manual

SleepFighter

Table of Contents

toxbee

Building

Requirements

Android API level 18 (4.3) SDK must be installed.

Steps

- Clone the project:
  `git clone https://github.com/Hasselmannen/sleepfighter.git`
- Import all projects in the following locations:
  - application/
  - application/lib-projects/
  - applicationTests-test/

Android Details

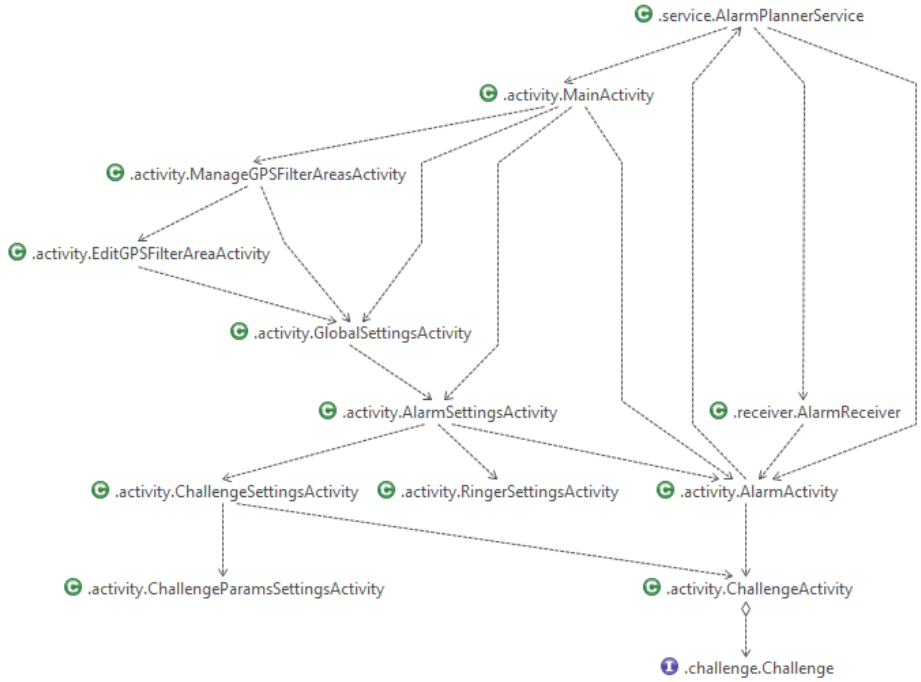
- Minimum supported API level is 9 (2.3). The reason for not using 8, which is the default and more often set as minimum, is that the library MBassador requires it. (It uses TimeUnit.MINUTES)
- The target of the application is the most current, level 18 (4.3), as of writing this.
- The Android UI design has mostly been done without using Fragments (Which is encouraged since API level 11), partly due to compatibility issues.
- UI-wise a dark Holo theme is used throughout the application

Architecture Overview

- Design principles
  - Object oriented, GRASP
  - DRY (don't repeat yourself) as opposed to CRY / WET.
- Goals
  - Modular, testable, loose coupling, etc.
- Design philosophy = active "MVC"
- Making STAN = happy.

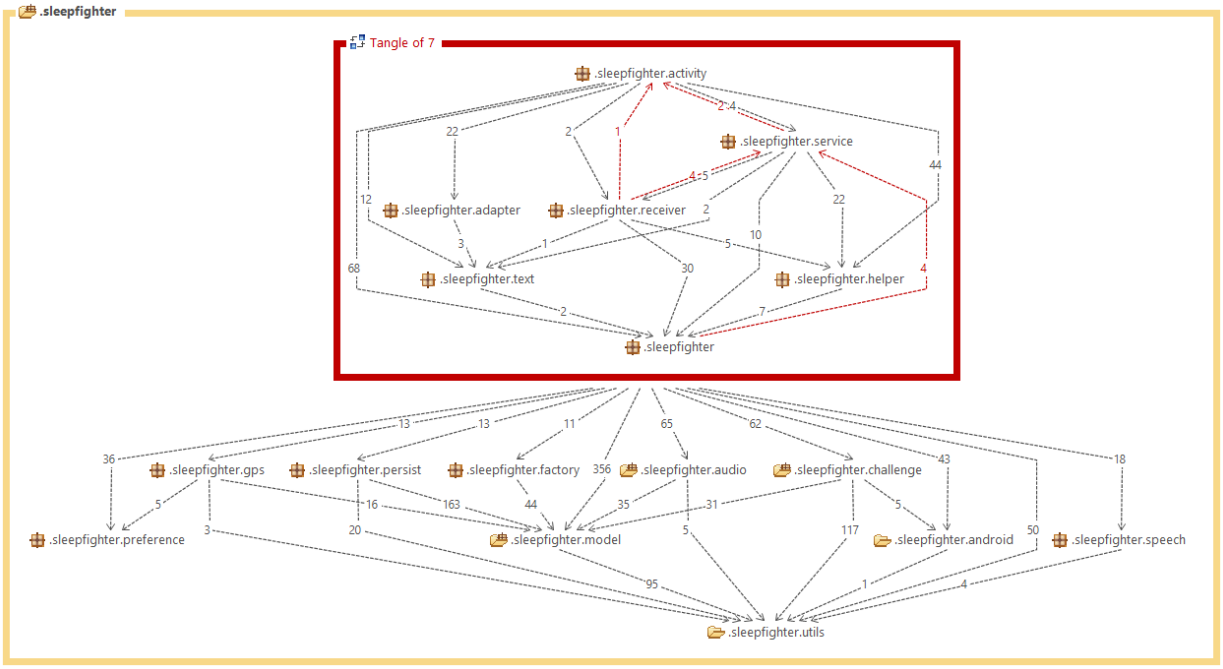
The application code resides in packages organized by area, such as activity (for Android Activities), challenge (for challenge logic and so on), model (for the basic app functionality) etc. Various helpers are in the utils package.

App Hierarchy



The App Hierarchy is a simplified version of the structure of entry-points and paths the user takes. The relationships between nodes is what's important.

Package Dependencies



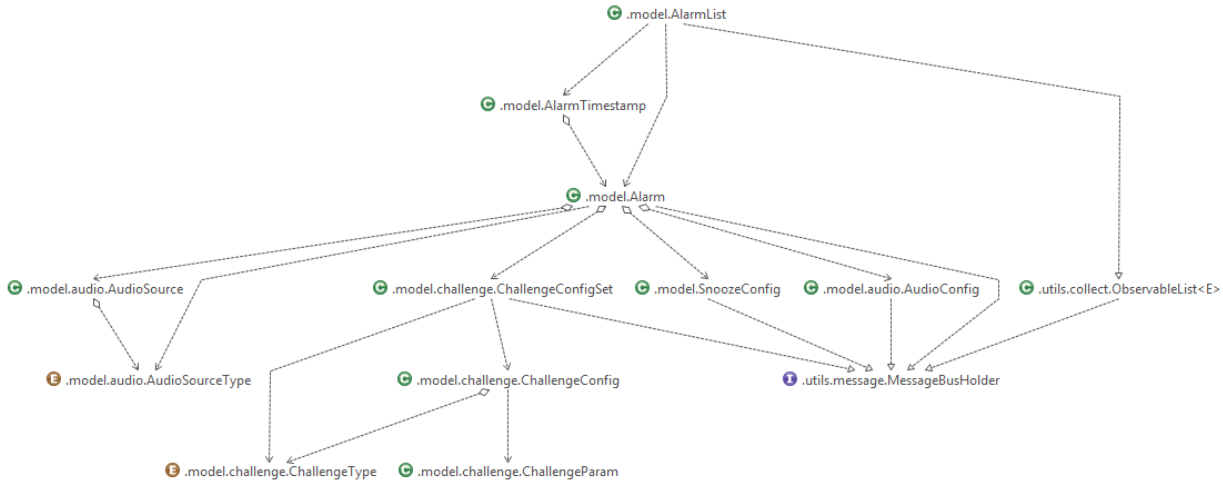
This is an overview of the entire se.toxbee.sleepfighter package.

Model

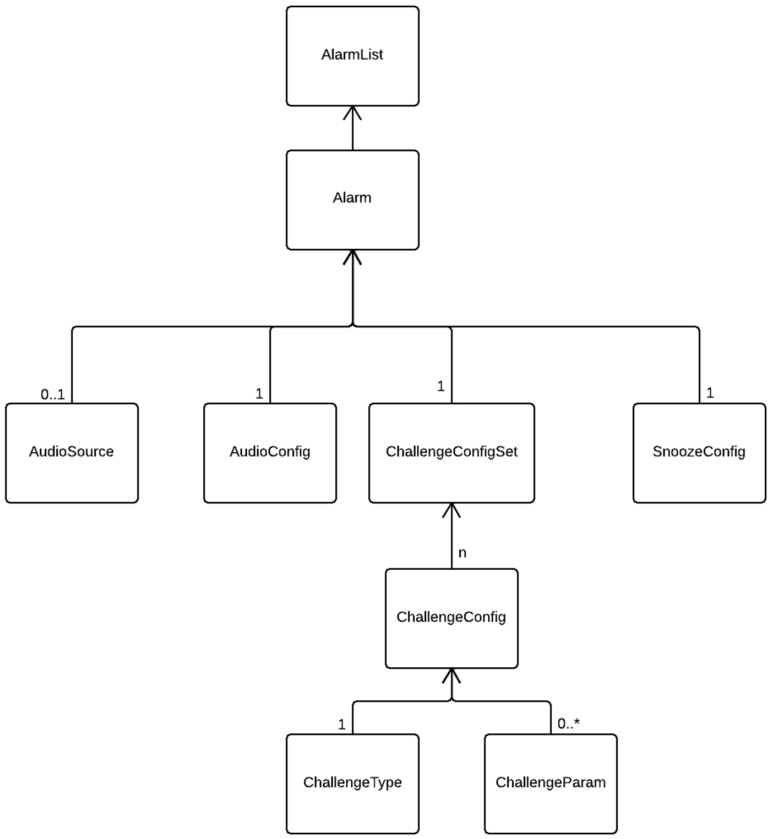
- Alarm is the primary model class.
- Most classes has message bus to publish class specific events through.
  This is how the active model in MVC is implemented.

  The bus is dependency injected (DI) into aggregate objects of Alarm when they are set to Alarm or when Alarm gets a message bus.

  This also applies to Alarm itself - when an alarm is added to an AlarmList the list will inject its message bus.
- Please do note that the location-filter package (gpsfilter) is NOT a part of the model package, it is completely isolated.



Overview of the model package according to STAN, Location Filter packages not included.



UML of model package.

Message Bus

- MBassador
- All subscribers doesn't get all events, only "instanceof" what's in handler.
- Model updates trigger GUI update through bus and handler
- Less coupling between View and Model

For non-UI control related event handling, a message bus system is used throughout the application. Using an event bus lets us have looser coupling between the View and Model.

The library used for the functionality is MBassador (https://github.com/bennidi/mbassador), read more about it in the External Dependency appendix.

Persistence

- To SQLite database through ORMLite
- ORMLite wrapper classes
- Most fields automatically stored?

For persistence of the various model objects in the application a SQLite database is used. To bind the Java model to the database, the library ORMLite is used. **All** of the code regarding this is available in the .persist package. Most outside interaction is done through `PersistenceManager` available in that package.

The .persist module has nearly no bindings to other modules - this is thanks to using the global message bus to listen for changes to models.

The tables and fields in the database is defined by OrmLite annotations in the different model classes. This way, persistence has been implemented without having to do as much manual queries as we had worked directly with the SQLite database.

Whenever anything related to the fields and table in the database has been changed in the source code, such as addition of an annotated field, a "database version" integer has to be increased, as well upgrade code has to be written for change. This is done in OrmHelper.

Read more information about OrmLite in the External Dependencies appendix and find examples here.

Challenges

- Entry in enum and factory for clean instantiation and enumeration

A challenge in the application is defined by an interface, with several methods. When designing a way to define different challenges, we needed a way that gave much freedom to the challenge itself, to be able to have varied and advanced challenges. UI-wise, a challenge should be able to render itself in different ways, either by setting its own Android UI elements, or rendering itself using OpenGL. The way we chose to do it is not making the challenge itself extend something like a Fragment, or an Activity, but it's instead given an empty activity, through a method, when it's started. With that, the challenge can modify the UI and set listeners. One of the reasons for not extending an Android component is that we wanted to easily create instances through factories, which would have been more difficult.

Every challenge must itself signal when the user has completed it. That's done by having it send an event to marks its completion through an event bus it has been given access to.

Math Problems

A special category of challenges is the math problems.To create a new math problem, implement the interface MathProblem. Since a WebView utilizing the javascript library

jqmath is used to render the problems, you are easily able to create professional-looking math problems with little effort. See the documentation in MathProblem.java for more details. And to add a new math problem, please make the appropriate changes in the method runChallenge() of the class MathChallenge. MathChallenge already handles rotation of the device for you, so this takes very little effort to do.

Tests

Unit tests have been made for most of the core model as of yet. Upon implementing new features to the model, even just adding fields to different classes, some tests should be updated, and new ones created, if appropriate. For example, any field with restricted contents (such as a bounded integer), should have tests that makes sure the model can't be put into an invalid state.

Test cases with clearly defined steps have been made for some of the higher level features of the application, such as creating alarms and clearing challenges. When adding new features of similar magnitude, new test cases should be written, and previous test cases should be run again when applicable.

Release Procedure

Release branch

1. `git checkout -b release-<version-identifier>`

Stability Check

1. Run unit tests, fix if anything is broken.
2. Perform tests in Test Cases document and update the spreadsheet.
3. If database version has changed, make sure to test that upgrading from the previously released version works as it should. It would be very bad if the app crashed upon launch after a user has updated.

Version Code

1. Update `AndroidManifest.xml` in manifest, increase versionCode by one.
2. Increase `versionName` by 0.1 if feature release (0.01 if bugfix?).
3. Enter the new name in `strings_version.xml`, for version name in UI.

Documents

1. Make pdf version of Release notes and move to `<git root>/build/<versionName>/release-notes.pdf`
2. Sprint Documents
    a. Check that the release notes document contains what it should from the Sprint backlog
    b. Make pdf version of Project backlog and replace the old one on git `<git root>/documents/project-backlog.pdf`
    c. Make pdf version of Sprint backlog and add to git `<git root>/documents/sprint-<num>backlog.pdf`
    d. Commit document updates

Build

1. Proguard
    a. Need changes in proguard-project.txt! For example with Play Services (application/lib-projects/google-play-services_lib/proguard.txt)
    b. Test it, other libs might require proguard configuration!
2. Export signed APK, use the se.toxbee.sleepfighter private release keystore.
3. Install on device and try out if Google Maps work (correctly signed needed)
4. Commit as build release
5. Push before proceeding!

Merging and tagging

1. Merge into master

    a. `git checkout master`
    b. From release: `git merge --no-ff release-<version-identifier>`
       From develop: `git merge --no-ff development`
    c. (`gitk --all` to visually check if merge is correct)
    d. `git push`
2. Tag merge commit, tag needs to be pushed as well!
    a. `git tag -a <versionName>`
    b. Enter tag message.
       Format: "SleepFighter version <versionName> (Sprint <num>)"
    c. `git push origin <versionName>`

Pretty GitHub release (not done previously)

1. Use GitHub's online tools for prettier release, separate apk download, and release notes on GitHub: https://github.com/blog/1547-release-your-software
   Using this, we can attach an APK to releases without having it be in the repo, which isn't where git works best. If changed, build folder could perhaps be removed

Release to Google Play Store (not done prev)

1. Go to the developer console and follow instructions there using the .apk you've built previously.

Appendix: External Dependencies

OrmLite

http://ormlite.com/
License: MIT

OrmLite is an ORM, Object-Relational-Mapping(s) library for Java which works well with android.

An ORM is an API/framework which is used for automatically mapping (and querying, fetching, deleting, inserting and updating) stored data in a RDBMS, relational database management system, rather than manually writing SQL statements to a database and then parsing, which can be a cumbersome task.

The use of OrmLite in our project is limited to persisting and fetching data and doing simple queries.

MBassador

https://github.com/bennidi/mbassador
License: MIT

MBassador is a lightweight, performant and easily configurable message/event bus implementation that is used instead of a implementation with a whole host of listeners all-over-the-place which can easily get out of hand and provide sources of threading problems with null-pointer problems everywhere.

A message bus is a concept where instead of having an observer and an observable which both known of each other, you have an intermediary, the bus, which the observers can subscribe to and the observables can publish to. The observers and the observables have no knowledge of each other, they only know of the bus. Note the use of “observables”, it is plural - you can have many objects publishing to the same bus!

In our project, this is used to e.g. notify the GUI of change when the time of an alarm is changed.

Find more information about how it MBassador works on their git repository.

Joda Time

http://www.joda.org/joda-time/
License: Apache v 2.0

Our project is a project that at its core deals with time and date calculations above everything else. The occurrence of an alarm is after all entirely based on time.
Joda Time is a library used for date & time calculations which is faster, more comprehensive and most importantly conceptually and codewise better designed than java.util.Calendar.
It provides concepts such as “Period” which is optimal for our use.
java.util.Calendar was tried early on, but was dropped for precisely this reason.

Joda Time is a stable library which is used by e.g. Google.

Guava

http://code.google.com/p/guava-libraries/
License: Apache License 2.0

Guava is a basic and comprehensive utilities and collections library that provides many things that JDK 1.6 does not provide. Its use is very baseline and could be considered an “addition” to the JDK in that it should be included in most projects. Explaining guavas use is a difficult task since its use is possible in most fields, but a few examples are: Preconditions (a more elegant way of checking for e.g. null for method parameters).

Commons Math: The Apache Commons Mathematics Library

https://commons.apache.org/proper/commons-math/
License: Apache License, Version 2.0

Used for implementing the more complex math challenges. In the complex math challenges you do things like matrix multiplications, which is one of the things this library is used for in this project.

Google Play Services

https://developer.android.com/google/play-services
License: Same as rest of Android? Apache 2.0? Considered a system library?

Used for having the user define areas on a map where an alarm will or will not go off.

Spheres

http://geospatialmethods.org/spheres/
License: GNU General Public License

Mathematics library used for spherical trigonometry calculations, which is used in determining if the user is in an area defined by a location filter.

JQMath

http://mathscribe.com/author/jqmath.html
Copyright 2013, Mathscribe, Inc. Dual licensed under the MIT or GPL Version 2 licenses.

For the more complex math challenges, it was necessary to draw more complex math formulas. There is no android library for this, so we had no choice but to use a javascript library(jqmath) and a WebView to achieve this. The main reason we choose jqmath is that it is one of the smallest javascript libraries for rendering math. Small size is important for android apps, in our opinion.

JQuery

http://jquery.com/
License: MIT

JQuery is a javascript library which JQMath requires.

JSCurry

http://mathscribe.com/
Copyright 2012, Mathscribe, Inc. Dual licensed under the MIT or GPL Version 2 licenses.

JSCurryis a javascript library which JQMath requires.

LibGDX

http://libgdx.badlogicgames.com/
License: Apache 2.0

Only utility class `com.badlogic.gdx.utils.IntArray` is used. Added to optimize some algorithms?

Android Numberpicker

https://github.com/SimonVT/android-numberpicker
License: Apache 2.0

We couldn't use the android NumberPicker in the API level we used.

Forecast API

https://developer.forecast.io/

Terms of Use

Used in the application for getting a weather forecast to be read out to the user.