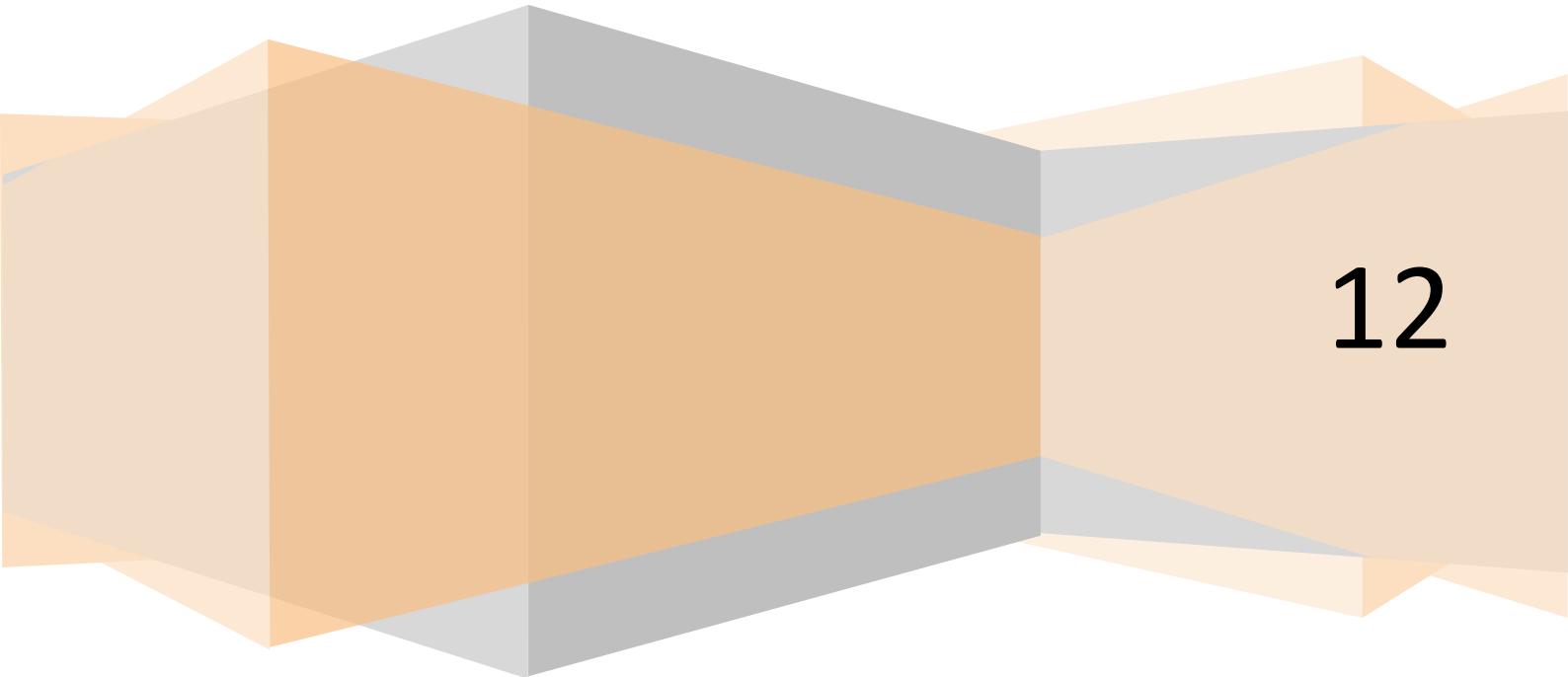


Group 26

Developer manual

Claudio

Marcus Parkkinen, Aki Käkelä



12

TABLE OF CONTENTS

1. Developer introduction

- 1.1 Setup
- 1.2 Project layout

2. Build procedure

- 2.1 Building the application
- 2.2 Automated builds
- 2.2 Libraries and dependencies

3. Release procedure

- 3.1 Preparing a release
- 3.2 Publishing a release
- 3.3 Post-release

1. Developer introduction

1.1 Setup

1. In order to run the application an emulator or Android device is needed. The application also requires API level 10 (Android 2.3.3) or above. In installing the Android SDK, make sure to create a virtual device with these requirements in mind. In addition to the Android SDK, Java SE 6 or later is needed to set up the development environment. An example of a recommended IDE is Eclipse combined with the ADT-plugin.
2. Fetch the current version of the application from the official git repository at:
<https://github.com/MarcusParkkinen/AudioBook.git>
3. Import the project into your development environment. The project contains a separate test project, *AudioBookTest*, which may require modification to the `.classpath` file to add it as a source folder.

1.2 Project layout

The source code of the project is divided into separate packages based on which module the classes belong to. This means that all code related to Android *activities* and GUI-elements reside in the *view* package, and all code that represents the *model*, or state, of the application is located in the *model* package and so forth. A more detailed description of the roles of the core classes in the application can be found in the **Architecture Specification**. Classes whose functionality is not related to the three main packages of the **MVC** pattern (Model, View, Controller) are grouped into the packages *interfaces*, *constants* and *util*. As the naming implies, the interfaces package contains interfaces that are implemented by classes from different modules within the application and thus do not fit into one specific module. Finally, the util package contains utility methods and the constants package contains hard-coded values used by the other modules.

As previously stated, the test classes of the project are all contained in an inner project. The layout matches the package structure of the application project, meaning tests for model classes are contained in the model package within the test project and so forth.

2. Build procedure

2.1 Building the application

To generate an .apk-file of the application, the project may be built directly within the development environment granted that the development environment is supplied with the ADT-plugin. In other cases, the project is also buildable through the supplied Ant file "build.xml" that can be found in the root folder of the application. The build.xml file can be run using a terminal on a system that has Apache Ant installed by simply navigating into the project folder and executing the command "ant build -f build.xml" or simply "ant build".

2.2 Automated builds

The provided Ant file build.xml can be configured to run within continuous-integration tools such as Jenkins, Gradle or Cruise-Control for automated builds. Currently, Claudio does not include automated builds in its development procedure.

2.3 Libraries and dependencies

All library dependencies are included in the libs folder by default and entries containing references to these are included in the .classpath file. These are therefore always included when the project is built.

In case other libraries are to be included and bundled within the application, the .classpath file must be updated with entries to these.

3. Release procedure

3.1 Preparing a release

As the features of the application are developed separately in different branches in the version control system, the initial step of forming a new release of the application is to assess which features are ready to be included. This step is usually carried out in a meeting where each team member describes the functional state of the feature(s) they are developing. This meeting is usually held in conjunction with the weekly sprint retrospective and planning. If any new features have been assessed and verified to be adequately functional, these are ready to be merged into either the main branch or the development branch as a release candidate based on its completeness.

The merging of new features is usually done by a member of the team who possesses knowledge of the areas of the application that are affected by the merge. A goal in the distribution of tasks during each sprint is to prevent several team members from working with the same classes, which usually makes merging a simple task.

When the merging is complete, regression testing is done to assert that features have not been broken during the merge. The contributions of these features are then added to the changelog for the new release along with known limitations and bugs. Finally, the new release is marked with a version number as a tag.

3.2 Publishing a release

When all preparations for a new release are completed, the new version of the application is pushed to the master branch in the application's repository along with a version tag. The push also includes a new version of the .apk for the application in the /apk/ folder that can be used to test the new release directly on Android devices.

3.2 Post-release tasks

In case the introduction of new features into a release has introduced new (minor) bugs, the recommended approach is to utilize the hotfix branch to quickly fix these. All major issues and bugs are to be reported through the issue tracker along with detailed descriptions, including triggers and conditions.