# TensionCamApp: Developer Manual

**Latest revision:** 2013-05-20

A quick-start guide and overlook of the TensionCamApp Android application designed for G-coder Systems AB.

## Getting started

git clone git:// github.com/lisarythenlarsson/TensionCamApp

## Dependencies

* Android SDK
* An actual (preferred) or virtual Android device
* Java XX XX (6 SE) development environment

## Android SDK targets

* Minimum SDK: **16**
* Target SDK: **16**

(SDK targets are motivated in *Project Plan*)

## License

## Building and installing

A build.xml is included in the root directory which may be used for building the project. The default output directory is bin in the project root. By default, Eclipse is set to build automatically. To build the project manually, go to *Project* in the menu bar.

To install the TensionCamApp on an Android device, the project has to be run as an Android Application in Eclipse. In order for this to work, the device has to be connected via Android Debug Bridge (ADB).

To uninstall the TensionCamApp on an Android device, standard procedures should be followed. I.e. using the device’s default tool for app-removal is recommended.

## Release procedure

This section describes the steps taken before every release of the TensionCam Application.

### Requirements

To build an application package in release mode, it needs to be signed with a certificate. Refer to this Android guide on signing applications for release: <http://developer.android.com/tools/publishing/app-signing.html>. Releases will be built, signed and compiled Eclipse ADT. Access to a keystore is necessary for signing application. The keystore path is TensionCamApp/certs/realese.keystore

### Building a release package

Use Eclipse ADT to compile and sign an apk-file for release:

1. Select the project in the Package Explorer and select **File > Export**.
2. Open the Android folder, select Export Android Application, and click **Next**.

The Export Android Application wizard now starts, which will guide you through the process of signing your application. When signing you will need to provide the key alias and password stored in keystore.

1. Complete the Export Wizard and your application will be compiled, signed, aligned, and ready for distribution.

### Organizing the distribution directory

After having built a release package, it should be organized in the distribution directory: **dist** in root direvtory.

1. Create a new directory in dist named with the version number. Examples:

a. v0.1

b. v0.2

.

.

.

2. Move the TensionCamApp.apk package from the bin directory to the newly created release directory

3. Rename the application package to TensionCamApp-<version>.apk. Examples:

a. TensionCamApp-v0.1.apk

b. TensionCamApp-v0.2.apk

### Release requirements

Every release’s directory include the following:

* An application package (see above).
* A release notes document with the following headings (if applicable):
  + New features
  + Changed features
  + Removed features
  + Known bugs
  + Coming features
* A test report document.

## Tests

Automatic tests are included in a separate project embedded within the application project which is called TensionCamTesterTest. To run the included tests

## Architecture

The functionality of the application is built up by three different parts. A web server together with a STS (Spring Tool Suite) project, external software for picture analysis and the android project. Kan vi motivera val här? The reasons for using a web server are several. The main reasons though are the following:

* Less redundancy since there is no need for each device (cell phone) to carry the analyze program. This minimizes the storage need for the application. It also enables the analyzing program to be run on a more powerful machine.

The application code resides in packages organized by area, such as database (for data provider

interactions), net (for HTTP and IMDb connections), activity (for Android Activities), etc. Various

helpers are in the utils package.

The domain model consists of two classes: Movie and Tag.

**Content Provider**

**Package:**

The application code

**External libraries:**

HttpClient 4.2.5

HttpCore 4.2.4

**Motivation for libraries:**

These two libraries enables connecting to and communicating with the web server through POST request.

database

The data source is backed by an SQLite database which a Content Provider is using to interface with

the application code.

**Web server Package**

**External libraries:**

Commons FileUpload 1.3

**Motivation for libraries:**

Enables the client to send picture (file) to the web server through a POST request whit multipart/form-data, in this case the picture.

**HTTP and the IMDb API connection**

**Package**

net

**API endpoint:**

http://api.themoviedb.org/2.1/

**Response format:**

JSON

The API calls to the IMDb API is handled by a middle layer class called HttpRetriever which

basically performs an HTTP GET request for an arbitrary URL (using the Apache HTTP library). The

top layer is the IMDBHandler class which uses a HttpRetriever to make calls to the IMDb API

service.

**Notifications and Services**

**Package**

notifications

The movie notification feature is handled by a system of alarm tasks, clients and Android Services.

The main task for the system is to schedule Android notifications on a certain date (e.g. a movie’s

release date, set by the user).

DAT255

To use notifications with the Movie model, create an instance of the NotificationClient

class, connect to the underlying notification service with connectToService(), and use

setMovieNotification(Movie movie) to set a notification for a movie object.