# System Design Document (SDD) for the Hedgehog Photo Project

Contents

Version: 1.0
Date: 2012-05-23
Author: Hedgehog Photo
This version overrides all previous versions.

# 1 Introduction

## 1.1 Design goals

The design must be very modular and flexible, parts should be isolatable for ie testing. The core (model) should be separated from the view and the controller so that the application should easily be ported to other platforms like Android.

## 1.2 Definitions, acronyms and abbreviations

MVC, an architectural pattern that splits and separates the interaction into three separate roles Model-View-Controller.
API, a specification intended to write plugins compatible with Hedgehog Photo.

# 2 System design

The application will heavily rely on MVC to achieve modularity. The supplied plugins will also use MVC architecture. User written plugins can use any system architecture as long as they use Hedgehog Photos custom annotations.

## 2.1 Overview

### 2.1.1 Event handling

There will be no central event handling in our application. Every component will have its own unique controller.

### 2.1.2 Plugin handling

Hedgehog Photo will have support for plugins. This makes the application very customisable and modular. Plugins will be loaded from a folder called plugin from the user home directory. Writing plugins should be an easy task and we want the devoloper to be in control as much as possible and thats why we have decided to have a very simple “API”.  We aid plugin writing by only having two recquired custom annotations; one which initialises the plugin and one which gets the view. The plugin loader also compiles the plugins if its necessary (if the .class file is nonexistent or outdated).

### 2.1.3 Stock plugins

Some stock plugins will be supplied with this applications. The stock plugins are: TagCloud, Map and Calendar. These plugins were made as they provide a richer experience while they at the same time were designed to stay to our core philosophy, simple and intuitive to use. These plugins are open for modifications. Users that want to write custom plugins can look at the source code to maybe get an even better understanding on how to write modifications to this program.

### 2.1.4 Database handling

Hedgehog Photo will include a database. The database will  save all necessary information-all information that will be shown in the program plus the paths- about the photo. You can get all information by using the DatabaseHandler class.

### 2.1.5 Import Photos

You can import photos from your computer to HedgehogPhoto by clicking on “Import photos” buttton. The database will save all necessary information.

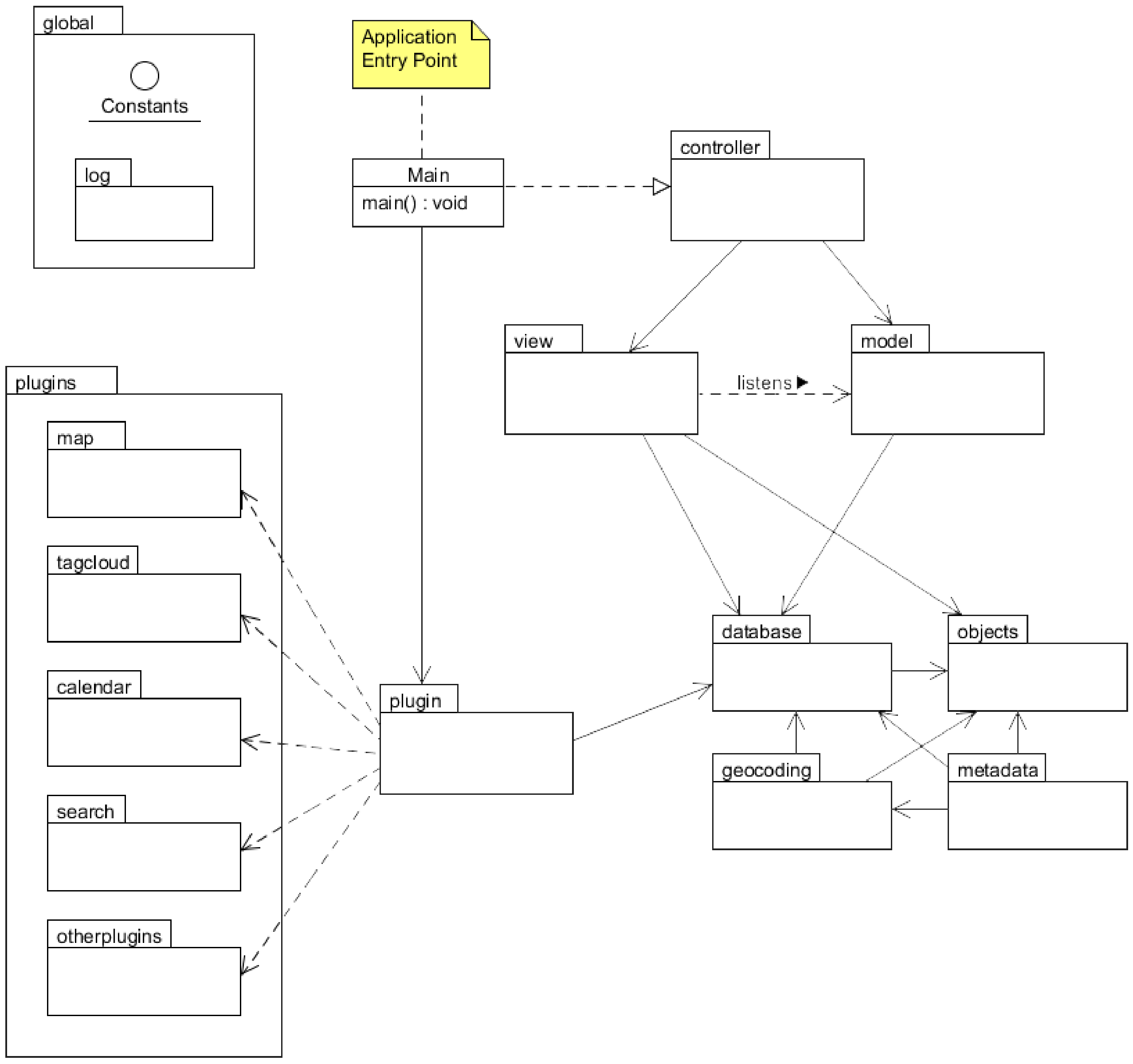
### 2.1.6 Error handling

Each time the program will run a log file which will be written to the root directory of the application. The log file contains debugging information that gets rewritten on each run. There are several logging level ranging from logs where the aim is to inform, to logs that helps aiding on debugging severe issues. Log level severe may contain exceptions and stack-dumps.

## 2.2 Software decomposition

### 2.2.1 General

The application is composed by the following modules, see figure below. For more detailed figures of each subpackage, see APPENDIX.



- View. This is the top level package for all GUI related classes concerning the main window, excluding plugins.
- Model. Starts the database.
- Controller. The top level package for GUI related events/listeners in the view.
- Global. Global accessible classes for everyone.
- Plugin. Root directory of classes needed to compile and read plugins.
- Database. Contains all database related classes to store and get information.
- Objects. Contains classes that specify a certain object which is used widely in the program.
- Geocoding. Subpackage containing all neccesary classes to parse longitude and latitude to a location-name and vice versa.
- Metadata. Top level package for classes that import pictures and read the metadata of files.

### 2.2.2 Layering
The application should use a model-view-controller-pattern where appropriate.

### 2.2.3 Dependency analysis
The application will use interfaces or abstract superclasses where appropriate, to minimize dependencies.

## 2.3 Concurrency issues
The application won't be thread-safe by default, though certain tasks might use threads.

## 2.4 Persistent data management
To store the data permanently, the application uses a database. By storing the file paths of the images, one makes sure that images that are once loaded in, remain in the application, and that data associated with them, don't dissappear.

## 2.5 Access control and security
N/A

## 2.6 Boundary conditions
N/A

## 3 References

MapPanel source code - http://sourceforge.net/projects/mappanel/
Sanselan Framework- http://commons.apache.org/imaging/