EGNSS4CAP ANDROID APPLICATION SOURCE CODE OVERVIEW

CONTENT

GENERAL OVERVIEW

The application is developed as a regular application for the Android OS in the Java language compiled in the Android Studio system with Gradle build automation tool. The source code is structured according to the android project standard descripted at https://developer.android.com/studio/projects.

The application consists of three parts, from the kernel of the application itself and the two libraries gnss_scan and convex_hull, which are distributed in AAR format, where the gnss_scan library is dependent on the convex_hull library. These libraries are separated from the functionality of the application kernel to allow their independent use in other projects.

All three parts of the code will be described by summarizing the description of their purpose, their classes, and their main configuration files.

APPLICATION KERNEL

The kernel of the application handles all essential tasks related to communication with the server, UI management, access to the phone hardware and other necessary processes required for running in the Android OS.

The primary purposes of this section are:

- Login / logout of user
- application permission checks
- hardware management (camera, gyroscope, compass, location sensors, …)
- taking and modifying photos
- geotagging photos
- sending and receiving photos with defined tasks from CAP (synchronization)
- viewing captured photos and tasks in the UI
- showing captured photos on a map
- record paths
- showing recorded paths on a map
- synchronized recorded paths with server
- display of land polygons on the map, defined by CAP
- displaying of currently received GNSS data
- application settings
- automatic language and locale settings by device

The base code consists of java classes in the main package eu.foxcom.gtphotos. All services and activities are defined in this root package. All activities inherit the base class BaseActivity. All fragments inherit the base class BaseFragment. This basic classes provides common functionalities, which are used in derived activity classes. The BaseActivity also serves as an access point for communicating and controlling the MainService shared between all activities. There are other packages and classes in the model package that implement functions, processes, and data, and are used in individual activities.

The following is a list and a description of each class from the root package:

- **AboutActivity**
  - The activity for displaying the summary of the application with BasicInfoFragment.
  - The layout is defined in the activity_about.xml.
- **BaseActivity**
  - The abstract class providing common processes to other activities
  - Check all necessary permissions to run the application.
  - It controls the user access to the application during synchronization.
  - The management and communication with the MainService.
  - The common broadcast messages management.
  - The main menu operation.
- **BaseFragment**
  - The abstract class providing common processes to other fragments.
  - The life status cycle monitoring.
- **BasicInfoFragment**
  - The fragment showing the hardware attributes of the phone, the ability to receive certain GNSS messages, and the current availability of the location service. Some of this data is displayed in the form of a traffic light.

- It needs to have mediated access to the gnss_scan library functions from its host activity.
- The layout is defined in fragment_basic_info.xml.

- **CameraActivity**
  - The activity that controls all processes while working with the phone's camera and taking photos.
  - The layout is defined in activity_camera.xml
- **CameraExposureCorrector**
  - The helper class used by CameraActivity to manually change brightness.
  - It helps eliminate some brightness issues on certain types of phones.
- **CameraViewModel**
  - The ViewModel that allows CameraActivity to keep the current location data and model/PhotoDataController instance out of the normal activity lifecycle.
- **GnssRawActivity**
  - The activity for displaying captured GNSS data.
  - It displays BasicInfoFragment.
  - It makes high use of the gnss_scan library.
  - The layout is defined in the activity_gnss_raw.xml
- **Launcher**
  - Makes certain settings that are useful when the application is first loaded into memory.
- **LoginActivity**
  - The activity to handle user logins.
  - The layout is defined in activity_login.xml
- **MainActivity**
  - The root activity displaying information about the logged user.
  - It displays BasicInfoFragment.
  - It serves as the main user signpost in the application.
- **MainService**
  - The service shared across all activities during application lifecycle.
  - It provides the main OS notification channel of the application.
  - It implements communication with activities via a broadcast channel.
  - It provides activities with unified access to the local database.
  - It implements the synchronization process between server and application data.
- **MapActivity**
  - The activity for displaying data on the map.
  - It displays photos at their captured location.
  - It displays land polygons provided by CAP.
  - It displays recorded paths and controls path recording process using helper classes in model/pathTrack
  - The layout is defined in the activity_map.xml.
- **PathTrackingOverviewActivity**
  - The activity for displaying the recorded path in a list structure and managing them.
  - The layout is defined in the activity_path_tracking_overview.xml.
- **PermissionActivity**
  - The activity used to prevent the use of the application without sufficient permissions from the OS.
  - The layout is defined in activity_permission.xml.
- **PhotoDetailActivity**
  - The activity to view detail of an unowned photo and to manage this photo.

    - The layout is defined in activity_photo_detail.xml
- **ServiceController, ServiceGetter, ServiceInit**
    - The general classes and interface that facilitates the management of services in activities.
- **SettingActivity, SettingsFilterActivity**
    - The activities allowing access to the settings of the application.
    - The layout is defined in activity_settings.xml and activity_settings_filter.xml.
- **StartActivity**
    - The activity that serves as an initial router among other activities.
    - The layout is defined in activity_start.xml
- **TaskFulfillActivity**
    - The activity to view detail of a task and to manage this task.
    - The layout is defined in activity_task_fulfill.xml
- **TaskOverviewActivity**
    - The activity for displaying the task list.
    - The layout is defined in activity_task_overview.xml
- **UnownedPhotoOverviewActivity**
    - The activity for displaying a list of unowned photos and managing them.
    - The layout is defined in activity_unowned_photo_overview.xml.
- **model/component/…**
    - The classes representing some custom graphic elements used in the application.
- **model/functionInterface/…**
    - The interfaces simulating some interfaces used in standard Java 8 libraries.
- **model/gnss/NMEAParserApp**
    - The class extending NMEAParser from the gnss_scan library for the specific needs of the application kernel.
- **model/groundGeometry/…**
    - The classes for managing the display of land polygons on the map in the MapActivity class.
- **model/pathTrack/…**
    - The classes to record and to draw paths on the map used in the MapActivity class.
- **model/AppDatabase**
    - The class for defining the local database configuration and its incremental changes.
- **model/CameraController**
    - The helper class for obtaining static hardware information about phone's camera.
- **model/Converters**
    - The class that defines rules for converting data types between a local database and application runtime memory.
- **model/Digest**
    - The class used to create the file signature of the captured photo.
- **model/ExifUtil**
    - The class that provides functions for reading or writing data in EXIF format.
- **model/LoggedUser, model/LoggedUserDao**
    - The database classes for manipulation and operations with a user data entity.
- **model/PersistData**
    - The class that serves as a unified gateway for writing and reading data stored in sharedPreferences memory.
- **model/Photo, model/PhotoDao**
    - The database classes for manipulation and operations with a photo data entity.
- **model/PhotoDataController**

    - The class to control the acquisition of current geolocation data along with the orientation of the telephone in space.
- **model/PhotoList**
    - The class representing a data collection of classes Photo and PhotoDao and implementing the necessary operations on that collection.
- **model/PositionSensorController**
    - The class that implements operations on gyroscope and compass sensors and calculate phone's orientation in space over raw data from these sensors.
- **model/Requestor**
    - The class for sending and receiving http data between a server and an application in a unified format and authentication.
- **model/SyncQueue**
    - The general class to support synchronization of asynchronous processes
    - It is mainly used in the process of synchronization between server and application data.
- **model/Task, model/TaskDao**
    - The database classes for manipulation and operations with a task data entity.
- **model/TaskList**
    - The class representing a data collection of classes Task and TaskDao and implementing the necessary operations on that collection.
- **model/UpdateReceiver**
    - The abstract class that provides a basic interface to nested classes of other classes to register callbacks over a specific instance of the SyncQueue class.

The AndroidManifest.xml file contains the hierarchy between activities, their default configurations, explicitly required permissions as well as explicitly required device properties and other application configurations.

All UI layouts are defined in …/res/layout folder.

All supported application languages are in string.xml files:

- values/string.xml (english)
- values-cs/string.xml (czech)

Dependencies on other external libraries can be read from gradle configuration files.

All source files are located and structured in accordance with the required public standards for android projects (https://developer.android.com/studio/projects) .

GNSS_SCAN LIBRARY

The library that provides the ability to receive and process GNSS data and mobile network data in real time in a convenient way by registering callbacks to scanners of the mobile network and data from location satellites. This library is dependent on convex_hull library.

The primary purposes of this library are:

- receive GNSS data and enable their further processing
- receive mobile network data and enable their further processing

The following is a list and a description of each class from the root package eu.foxcom.gnss_scan:

- **CentroidFilter**
    - The interface allowing custom definition of filter rules when calculating the convex hull centroid over GGA messages.
- **CentroidFinisher**
    - The interface allowing custom definition of a rule for ending the data stream and starting the calculation of the convex hull centroid over this data.
- **Scanner**
    - The base class for all scanner class.
    - It implements the processes of registration and notifying of Receiver classes and ensures thread security during their asynchronous processing.
    - Not intended for public use.
- **Receiver**
    - The base class for all receiver classes.
    - The implementations of public derived classes are defined as nested classes in specific implementations of individual scanner classes.
    - The custom processes over the currently received data are created by extending public implementations of this class.
    - Not intended for public use.
- **Holder**
    - The base class for all receiver classes.
    - The implementations of public derived classes are defined as nested classes in specific implementations of individual scanner classes.
    - It serves as a data carrier when transmitting data between the scan and the receiver.
    - Not intended for public use.
- **GnssStatusScanner**
    - The scanner class implementing processes and classes for scanning data from location service.
- **NetworkInfoScanner**
    - The scanner class implementing processes and classes for scanning mobile network data.
- **NMEAScanner**
    - The scanner class implementing processes and classes for scanning raw NMEA messages.
- **NMEAParser**
    - The derived class from NMEAScanner.NMEAReceiver class.
    - It is used to process a raw text NMEA message into data objects based on the NMEA sentence types.
    - It provides classes for the ability to define custom processes based on NMEA sentence types.

  - It provides the ability to calculate convex hull centroid location above this data.

The AndroidManifest.xml file contains explicitly required permissions.

Dependencies on other external libraries can be read from gradle configuration files.

All source files are located and structured in accordance with the required public standards for android AAR library (https://developer.android.com/studio/projects/android-library#aar-contents).

CONVEX_HULL LIBRARY

The library providing the ability to calculate the convex hull centroid based on the provided data in the format of a two-dimensional Cartesian coordinate system.

The primary purposes of this library are:

- To calculate a convex hull centroid of the provided data

The following is a list and a description of each class from the root package eu.foxcom.convex_hull:

- **Cluster**
  - The class represents a collection of all points.
  - It calculates the convex hull centroid.
- **Point**
  - The class representing a single point data object.
- **QuickHull**
  - The helper class for computing convex hull centroid according to the quickhull algorithm.
  - Not intended for public use.

Dependencies on other external libraries can be read from gradle configuration files.

All source files are located and structured in accordance with the required public standards for android AAR library (https://developer.android.com/studio/projects/android-library#aar-contents).