NegevSat Ground Station Software Platform

User Guide

Table of Contents

1. Maintainer's Guide

In this section we will go over the central classes of the system, their purpose, and will see how we make changes in the system (add/change telemetry data types, change the communication protocol, replace simple serial communication with a reliable one like TCP/IP, etc.)

The project contains 2 source directories:

- *src* – contains all the packages that handle data (storing, acquiring, exchanging with the satellite)
- *GUI* – contains all the Graphical User Interface code and resources

The ***src*** directory has the following packages:

- data – the central package, containing all the data type classes and the Data Manager – the central singleton, that orchestrates all the data activity.
- persistency – this package handles all the database operations
- communication – this package contains modules that handle communication with the satellite
- logger – handles Action/Error logging
- orbit – this package contains code, handling orbit propagation (i.e. computes when exactly the satellite will be in a pass and can communicate with the ground station). Currently contains only dummy code, to be extended in the future project iterations.

1) The "data" package

The main data package classes that you should note:

- DataManager
- Satellite
- Component, Energy, Temperature
- Mission

The DataManager class is the center of all activity; the communication manager and database manager are initiated in the constructor of the DataManager. The DataManager was built as a singleton, so the function **getInstance** returns the dataManager instance.

For every action or query the GUI wished to perform, it activates the following DataManager functions:

- **Public void setLatestSatData(Satellite sat)**- the dataManager keeps track of the latest updates the satellite sent about its components.
- **Public Satellite getLatestSatData()**- the getter for the latest data.

- **Public List<Temprature>getTemprature(Timestamp startDate, Timestamp endDate)**
- **Public List<Energy>getEnergy(Timestamp startDate, Timestamp endDate)**
- **Public List<Satellite>getStatus(Timestamp startDate, Timestamp endDate)**
- **Public List<Mission>getMissions(Timestamp startDate, Timestamp endDate)**
  These four getters create lists of objects in the time range provided. They are directed to the database manager.
- **Public Mission getMission(Timestamp creationTimestamp)** - this getter finds one mission with the input creation timestamp. It is one because this is the primary key of the table.
- **Public ArrayList<Pair<String,Pair<Status,Timestamp>>>getListOfStatusPairs(Satellite satellite)** -returns a list of status and timestamps for every component.
- **Public Map<String,Float>getReadingsPerSensor(Component component)** - returns a map of the values of a component.
- **Public SatelliteState getLastSatelliteState()**-gets the latest satellite state which is by default UNKNOWN.

As well, we have insert, set and delete functions.

<u>Note:</u> you should address all the data queries from outside the package through the DataManager!

One object of type <u>Satellite</u> is always stored by the data manager, and holds the latest data collected from the satellite (statuses, temperature, energy, etc..).

<u>Component</u> type, extended by <u>Temperature</u> and <u>Energy</u> in this version, holds data on specific component of the satellite. In future versions, when more satellite components data will be stored in the ground station – there will be more classes, extending the Component class.

Mission – stores mission data.

<u>Note:</u> Components and Missions are stored in SQLite database using ORM technique, specifically – using ORMLite library. Please review the implementation of Temperature, Energy and Mission.

2) The "persistency" package

This package contains one class - dbConnection. This class handles all the connectivity with SQLite database handling ORM. Its methods are called by the DataManager class exclusively. If you want to work with another database, just change the implementation of this class.

3) The "communication" package

This package handles communication with the satellite. Note the following classes:

- CommunicationManager – singleton class, providing interface to the communication layer for the rest of the packages. It holds some necessary objects to handle the communication like input and output Blocking Queues, Locks (some concurrency in this package, will be explained) and another data.
    - Note the connect() method in this class, it initializes a connection to a Satellite (or Simulator) and creates three new threads: SerialReader, SerialWriter, MessageParser.
- SerialReader – this thread is almost constantly asleep, and waken up only when data available in the Serial Buffer (implemented using SerialComm libraries SerialListener). Then the thread reads some quantity of data from the buffer, and goes back to sleep. If there's still data in the buffer – it's wakening up again and so on. When the buffer is empty – the thread continues to sleep until new data arrives. The Reader looks for \2 and \4 bytes at the start and at the end of the message respectively. When whole message arrives, the Reader passes it to the Message parser through messageAcceptorQueue.
- SerialWriter – this thread sleeps on the outputQueue blocking queue. When you want to send a message to the satellite, you put the message into the queue, the SerialWriter wakes up, sends it and goes back to sleep.
- MessageParser –the MessageParser thread then wakes up, parses the message and calls the needed DataManager methods to store the data accepted and update the GUI – and then goes back to sleep.
- SatelliteSimulator – simple main function that you can run on another computer to simulate messages that the Satellite sends to the Ground Station. Created for testing purposes.

4) The "orbit" and the "logger"

The "logger" package handles logging. Currently it creates html-based logs. You can modify the Loggers class to change the way log messages are written, to add more logs, etc. In LoggersFormat class you can change the format of the log messages.

The "orbit" package is currently a dummy, which in the future should be replaced by a much more sophisticated system, maybe using 3rd party software like AGI STK (which we didn't get to use this year). As in all the packages, you have an OrbitManager, providing interface for another packages.

Overall, the purpose of the "orbit" package is to tell the Ground Station when the Satellite will be in pass over the Ground Station.

"OrbitPropogator" interface should be implemented by a class, which can tell us when the next pass is going to occur and give us a list of passes that will occur in a certain period of time.

The Pass class, given the start and the end time, can tell us if we are in pass phase right now, or how much time is left to the pass start or end time.

Although right now this package does not do a lot, it's a nice preparation for future development, where all you need is just to add a good implementation for the OrbitPropogator interface.

5) Modification guide

DB:

There are currently four tables in the database: Energy, Temprature, Satellite and Mission.

On the first time you will run the program, the tables will not exist. That's not a problem because the DataManager constructor calls the function **createTables** from class dbConnection and creates then if they don't exist.

In case you want to clean your tables from content call the function **clearTables**.

We only allowed changing objects of type Mission (after they were created and inserted to the database) – this is because all other data is received from the satellite.

Data:

If you want to add components, take a look at the existing components classes and build them similarly. You should read about ORMLite in order to understand what needs to be added to every class. In the same way, look for an existing component in dbConnection class in persistency package; you will need to add functions and fields with the same logic.

Communication:

When you change data types that the Ground Station works with (change or add Components), you naturally will find yourself changing the communication protocol, and, thus, the Message Parser. You will find this extremely easy here.

The MessageParser class handles all types of messages that it accepts from the messageAcceptorQueue Blocking Queue. This is the only stage where you operate with the application protocol (see the Protocol Guide). MessageParser also holds all the String constants in local variables, thus you will find it extremely easy to change anything you want in the protocol. So, when adding or changing the data types – in the communication package you will need to change only the MessageParser.

To change the carrier protocol (i.e. to transfer from simple Serial Communication to TCP/IP) just replace the SerialReader and SerialWriter with another threads that will receive and send the messages. This is a lot of work, but you won't find yourself changing anything except these classes and the Communication Manager.

GUI:

1. Menus:
    a. Go to `DefaultMenuFactoryImpl` and add a menu to the corresponding parent menus such as `createViewMenu` for the View or add a new function in the class.
    b. If creating a new menu create the function in DefaultMenuFactoryImpl and update its interface, after that add the call for the function in class `MainMenu` `init function`
2. Mission components for building a mission go to MissionSplitFrameImpl and add an item to one of the functions : `getDateAndLocationItems` , `getGeneralMissionDescription`, `getSatteliteComponentCommands`.

```
Example:
list.add(new TreeItem (new MissionTreeItem(new MissionTextWrapper("Type Text
Here"), "Custom mission", new Label("Mission description:"), MISSION_DESC)));
```

`Where TreeItem is javaFX class for TreeView Nodes` MissionTreeItem is a wrapper for the actual mission item with all mission data.
MissionTextWrapper is a class wrapper for text representation it implements `MissionItemWrapper which add functionality for text representation.`

For adding new mission components that are not related to the said functions, create new function that returns ObservableList<TreeItem> and add the return value in `populateLeftList function and update the finals in the top of the class:`

```
private final int MISSION_DESC = 0;
private final int MISSION_DATE = 1;
private final int MISSION_DATE_END = 2;

private final int COMPONENT_ON_OFF_LOCATION = 4;
private final int MISSION_LOCATION_TABLE = 5;
private final int MISSION_MAX_NUM_ITEMS = 6;
```

<- inc this one if you add a new function or if you are adding new mission attribute: similar to MISSION_DATE or MISSION_DATE_END

If you are adding a new Mission type add its enum to MissionType Enums and add a case in `createMission() function` and add the mission in `getGeneralMissionDescription`.

3. Adding new statistics Panel:
   Extend the abstract class AbstractComponentStatistics and implement the two abstract functions.
   Then just add it as an action where ever you want for example in `MenuItemTableEnergy the statistics class EnergyComponentStatistics is called.`

Packages:
CSSPackage – Contains the most of the Images and css graphic files
Factories – Contains the menu factory, used only for menus
MenuItems – Contains all the menu items that are shown in the gui
MissionFrames – Contains the Mission Frame that are used to build the mission panel
MissionItems – Contains the Mission Items and also the wrapper classes for the Mission Frames.
Negevsatgui – Containst the main windows and the Controller for the satellite Image
Panels-Contains All the panels(Except the two in MissionFrames), and the interfaces for those panels.

SatteliteData – A package with classes for tabbed satellite components presentation, it wasn't finished and probably could be deleted
StatisticsItems – Helper Items for statistics view
Utils – Containst the Utils class that has some static common used functions and the Constant Class that contains all the strings and path to files.
Furthermore it contains the GuiManager which connects between GUI, Data and communications(From the gui side).
Webmap – A package that contains the API and some implementation for google maps, should be replaces with STK

2. User Guide

Deployment:

To deploy the development environment, you will need the following:

- Java JDK 1.8.0 32-bit and Eclipse (or any other IDE you like)
- Git
- Serial Connection between two machines
    - We advise you to use Virtual Serial Ports Emulator
    - Using this software you can create local serial connection between two processes (e.g. COM1 and COM2 ports connected between them), and also COM ports on two machines, connected with each other on top of TCP protocol – just use your local WiFi to connect!
    - http://www.eterlogic.com/Products.VSPE.html
- Java Serial Communication package RXTXcomm. Download it from http://www.jcontrol.org/download/rxtx_en.html
- Sqlite command-line shell manager executable in C:\sqlite. Download it from: http://www.sqlite.org/2014/sqlite-shell-osx-x86-3080600.zip
- Additional Java packages, you can find them in the *lib* directory in the Git repository.

```
ormlite-jdbc-4.48.jar
ormlite-core-4.48.jar
sqlite-jdbc4-3.8.2-SNAPSHOT
hamcrest-core-1.3.jar
jfxtras.jar
joda-time-2.3.jar
junit-4.11.jar
poi-3.10.1-20140818.jar
```

The deployment procedure:

1. Install JDK1.8 32-bit
2. Install Eclipse
3. Add RXTXcomm to the JDK1.8's JRE directories:
    a. Place the `rxtxSerial.dll` file in your C:\`Program Files(x86)\Java\jdk1.8.0*\jre\bin` directory.
    b. Place the `RXTXcomm.jar` file in your C:\`Program Files(x86)\Java\jdk1.8.0*\jre\lib\ext` directory.
4. In GitBash interface, go to your workspace directory and clone the git repository there.
    git clone https://github.com/markkemel/negevsatgc.git
    (replace the link with the updated one)

5. Now go to Eclipse, make sure that you are in the right workspace directory, and create a project called "*negevsatgc*". Eclipse will recognize that the project directory already exists and will import all the needed settings.
6. Check that the Project uses JDK1.8 32 bit as its default JDK and JRE. To ensure that, go to *Project -> Properties -> Java Build Path -> Libraries -> JRE System Library*
7. Check that all the packages in the *lib* directory are used by the project and that the RXTXcomm library is recognized. You should get no compilation errors.
8. If you have compilation error with the Serial Communication library, try to reload the JDK1.8 to Eclipse.
9. Create an empty SQLite database in C:\sqlite\negevSatDB.db. The tables will be created automatically. To do so, go to cmd, and run:
```
cd C:\sqlite
sqlite3.exe negevSatDB.db
.quit
```

Running the program:

To run the program, run the GUI/negevsatgui/NegevSatGui.java class.

If you have any problems, send us an e-mail to m.kemel@gmail.com and we'll be happy to assist!

Using the GUI

When the program starts, a login screen will appear. Just press sign in.

Choose one of the tabs above:

File:

Home – will bring you to the online screen.

Exit – will end the program.

View:

View>screens will navigate you to the screen you choose.

The online screen is only active when the satellite is in the pass phase.

If you wish to view one of the tables choose **View>View Tables** and then choose a component.

3. Testing document

DataManager:

function	Input	Result
getTempratue	A valid range of start date and end date	A list of Temprature objects with primary key in the date range
getTempratue	Invalid range of start date and end date	An empty list
getEnergy	Valid range of start date and end date	A list of Energy objects with primary key in the date range
getEnergy	Invalid range of start date and end date	An empty list
getSatellite	Valid range of start date and end date	A list of Satellite objects with primary key in the date range
getSatellite	Invalid range of start date and end date	An empty list
getMissions	Valid range of start date and end date	A list of Missions objects with primary key in the date range
getMissions	Invalid range of start date and end date	An empty list
getMission	A creation timestamp of a mission	A mission with the creation timestamp
getMission	A creation timestamp that doesn't belong to any mission	Null
getLatestSatelliteData	None	A satellite object with the latest data
getListOfStatustPairs	Satellite object	Status and timestamp for every component
getReadingsPerSensor	A component object	A map of sensors and their values
insertMission	Timestamp, command, priority	A new mission was added
insertSatellite	Status and timestamp for each field	A new satellite was added
insertSatellite	Satellite state, status and timestamp for each field	A new satellite was added
insertTemprature	Sensors values and a timestamp	A new temperature was added
insertEnergy	Battery data and timestamp	A new Energy was added
setMission	Mission, timestamp,	Mission updated 3 fields

	command, priority	
setMission	null, timestamp, command, priority	Error message
setMission	Mission, null, null, priority	Mission updated priority
setMission	Mission, timestamp, null, 0	Mission updated timestamp
setMission	Mission, null, command, 0	Mission updated command
setMissionSentTS	Mission, timestamp	Mission updated sent time
setMissionSentTS	null, timestamp	Error message
deleteComponent	null, timestamp	Nothing
deleteComponent	“string”, timestamp	Nothing
deleteComponent	“Energy”, timestamp	Energy data was delete
deleteComponent	“Temperature”, timestamp	Temprature data was delete
deleteComponent	“Temperature”, not_exist_ts	Nothing
deleteComponent	“Temperature”, null	Nothing
deleteCompletedMission	timestamp	Mission data was delete
deleteCompletedMission	null	Nothing

4. Future Work

- Work with Orbit Propagator software, like AGI STK
- Add more data types
- Replace the communication protocol to be secure and reliable protocol, like TCP.
- Improve the Orbit visualization in the GUI main screen