Technical Document

Brief Specification

The Phoenix Simulator is a modified version of the AGA simulator - which in turn is a software that allows a PC to be used as a vehicle data simulator, enabling Android developers to test their automotive applications without the need of being connected to an actual vehicle.

The original AGA simulator has support to read and map telemetry data from TORCS - an open source racing simulation game. The Phoenix Simulator expands these capabilities by adding support to another game, Euro Truck Simulator 2 (ETS2).

The main advantages of having ETS2 as a test environment is that it simulates driving a truck in a setting that is closer to reality (i.e. cities and highways at moderate speeds, not racing tracks and cars that can easily reach 300km/h.

The Phoenix Simulator supports all signals found in the FMS specification v03.

In short, the simulator can connect to any Android device running the AGA ROM. This device can be connected to the same LAN as the computer running the simulator, to a virtual Android device on the same computer or to an app using the AGA jar files on the same LAN.

Architecture

The Phoenix Simulator acts as a link between ETS2 and the AGA application, by mapping game telemetry data into AGA FMS signals and sending those signals via a socket to the AGA application. The simulator gets the telemetry data from the game by reading from a web socket. The data is fed into this socket via a third-party free application - the ETS2 Telemetry Server. Figure 1 presents an overview of this system:

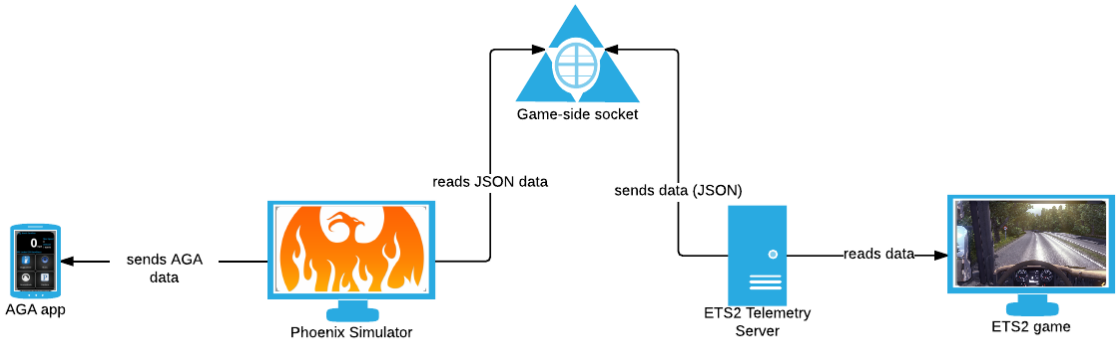

Figure 1: System overview

The ETS2 Telemetry Server

The ETS2 Telemetry Server is an application written in C/C++ that is part of an open source SDK for the ETS2 game. It parses in-game telemetry data and opens a server on either localhost or you current IP-address on port 25555. The values read from the game are sent to this server in JSON format.

The Phoenix Simulator opens a socket that reads from this port, converts the values into FMS/AGA signals and sends them via a socket.

Phoenix Simulator

The Phoenix Simulator is a Java application based on the AGA Simulator. The Phoenix Simulator was built upon the command-line version of the AGA Simulator (A GUI has been implemented in another version, named AGA Simulator FX).

Only two Java classes had to be modified from the original simulator: *Main.java* and *Torcs.java*. The latter has been copied into a new package (*ets2*), and refactored to connect to the ETS2 game. Their location in the project is shown in Figure 2:

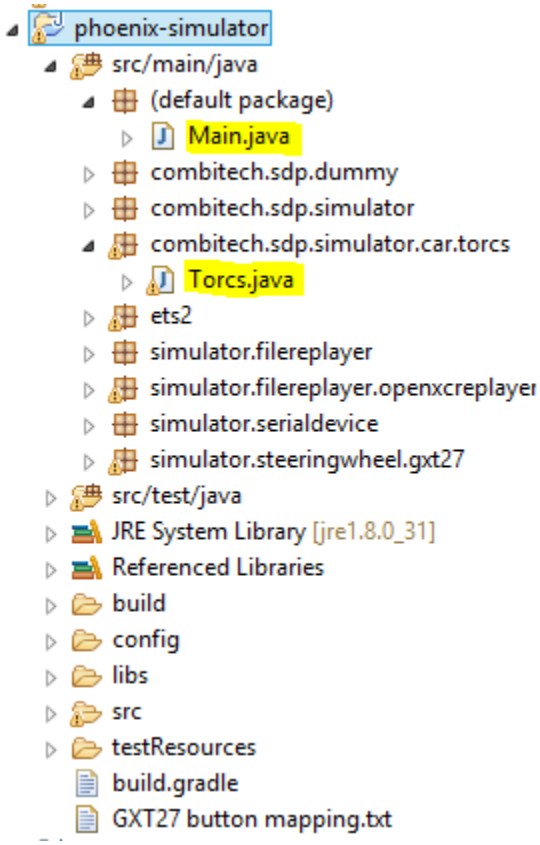

Figure 2: Java classes modified from the original AGA Simulator (highlighted in yellow)

ets2/EuroTruck.java

The class *EuroTruck.java* extends AGA's *BasicModule.java*, implementing our own ETS2 module. As seen in Figure 3, a server socket and a client socket are declared. The first is required by AGA's Simulator Gateway, which is responsible for sending the AGA/FMS signals to the AGA app, while the latter will take care of reading telemetry data from the ETS2 Telemetry Server.

Variables representing desired signals and their type are also declared, and are not limited to the ones used in our project. A new Simulator Gateway constructor is also required.

The *BasicModule.java* method *getProvidingSignals* is overriden to include only the FMS signals to be handled. Signals can be added or removed, but it is important to updated the declared signal variables accordingly.

```
public class EuroTruck extends BasicModule {

    //Declare server variables
    ServerSocket welcomeSocket;
    private Socket clientSocket;

    //Declare signals variables
    short currentGear;
    float fuelLevel;
    float fuelConsumption;
    float speed;
    float brake;
    float engineSpeed;

    //Simulator Gateway Constructor
    public EuroTruck(SimulatorGateway gateway) {
        super(gateway);
    }

    //Define provided signals
    @Override
    public int[] getProvidingSignals() {
        return new int[] { AutomotiveSignalId.FMS_WHEEL_BASED_SPEED,
                AutomotiveSignalId.FMS_CURRENT_GEAR,
                AutomotiveSignalId.FMS_ENGINE_SPEED,
                AutomotiveSignalId.FMS_TACHOGRAPH_VEHICLE_SPEED,
                AutomotiveSignalId.FMS_FUEL_RATE,
//              AutomotiveSignalId.FMS_FUEL_LEVEL_1,
//              AutomotiveSignalId.FMS_HIGH_RESOLUTION_TOTAL_VEHICLE_DISTANCE,
//              AutomotiveSignalId.FMS_ACCELERATOR_PEDAL_POSITION_1
                };
    }
```

Figure 3: EuroTruck.java, variable declarations, gateway constructor and getProvidingSignals() method

Figure 4 displays the servers initialisation in the *run()* method. The *signalUpdate* string is declared (it will later store the raw telemetry data read from the client socket.

The server socket *welcomeSocket* is initialised on port 6000 (AGA standard), as well as the *clientSocket*. The *clientSocket* points to the port 25555 in the localhost, which is the port where the Euro Truck Telemetry Server streams the game data in real time.

A PrintWriter *out* is declared and initialised, with the sole purpose of sending the commands which will tell the telemetry server to get the latest available data from the game.

BufferedReader *inFromETS2* is also declared and initialised, and starts reading streamed data from the *clientSocket*.

```
//run implementation
@Override
public void run() {
    //Set Module Thread name
    getModuleThread().setName("ETS2");

    //Declare variable that will store the latest acquired values from ETS2
    String signalUpdate;

    //While simulation module is running, initialise servers
    while (state == SimulationModuleState.RUNNING) {
        try {
            if (welcomeSocket == null) {
                //Initialise server socket
                welcomeSocket = new ServerSocket(6000);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        try {
            //Initialise client socket (address where ETS2 data is being sent by ETS2 Telemetry Server
            clientSocket = new Socket("127.0.0.1", 25555);

            //Initialise PrintWriter, used to send refresh command to the ETS2 Telemetry server
            PrintWriter out = new PrintWriter(
                    clientSocket.getOutputStream(), true);
            //REST API command to refresh telemetry data
            out.println("GET /api/ets2/telemetry");
            out.flush();

            //Initialise Buffered Reader that gets input stream data from ETS2 Telemetry server
            BufferedReader inFromETS2 = new BufferedReader(
                    new InputStreamReader(clientSocket.getInputStream()));
```

Figure 4: run() method, initialising sockets

After sockets are initialised and communication between servers is established, it is possible to map the data from the game into AGA/FMS signals. This is shown in Figure 5, where the *signalUpdate* variable takes the value of the latest available game data from the BufferedReader *inFromETS2*. The method *extractValues()* is used to extract relevant values from the raw data, which in turn are mapped and sent as AGA/FMS signals through the gateway.

```
            //While simulation module is running and client socket is connected
            while (state == SimulationModuleState.RUNNING
                    && clientSocket.isConnected()) {

                //initialise variable signalUpdate with latest values from ETS2
                signalUpdate = inFromETS2.readLine();
                if (signalUpdate == null) {
                    break;
                }
                if (signalUpdate.length() > 30) {
                    //map ETS2 values into AGA/FMS signals
                    extractValues(signalUpdate);
                    gateway.sendValue(
                            AutomotiveSignalId.FMS_WHEEL_BASED_SPEED,
                            new SCSFloat(speed));
                    gateway.sendValue(AutomotiveSignalId.FMS_CURRENT_GEAR,
                            new SCSShort(currentGear));
                    gateway.sendValue(AutomotiveSignalId.FMS_ENGINE_SPEED,
                            new SCSFloat(engineSpeed));
                    gateway.sendValue(AutomotiveSignalId.FMS_TACHOGRAPH_VEHICLE_SPEED,
                            new SCSFloat(brake));
                    gateway.sendValue(AutomotiveSignalId.FMS_FUEL_RATE,
                            new SCSFloat(fuelConsumption));
                }
                Thread.sleep(18);
            }
        } catch (Exception e1) {
            e1.printStackTrace();
        }
    }
}
```

Figure 5: run() method, mapping telemetry data into AGA/FMS signals and sending them to the AGA app via the gateway.

The *extractValues()* method (Figure 6) saves the raw game data from the *signalUpdate* variable into a string array. Then, by analysing the raw data sent from the ETS2 Telemetry Server it is possible to determine which string in the array holds the desired data value, and to store it in its proper variable. An example of the raw data provided by the ETS2 Telemetry Server can be seen in Figure 7.

For example, truck speed is the 6th value present in the raw data. The first 12 characters in the string define the in game name of the variable *“truckSpeed”:* - and the actual float value is stored from the 13th character onwards - therefore the *.substring(13)* method call.

```
    /*Method that stores telemetry data values into a string array,
    and initialise signals variables with corresponding values*/
    private void extractValues(String signalUpdate) {
        String[] values = signalUpdate.split(",");

        try {
            speed = Float.parseFloat(values[6].substring(13));
            currentGear = Short.parseShort(values[16].substring(7));
            engineSpeed = Float.parseFloat(values[20].substring(12));
            brake = Float.parseFloat(values[31].substring(12));
            fuelConsumption = Float.parseFloat(values[24].substring(25));
            fuelConsumption *= 100;
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //Stop simulation module method
    @Override
    public void stopModule() {
        state = SimulationModuleState.STOPPED;
        try {
            if (clientSocket != null) {
                clientSocket.close();
            }
            if (welcomeSocket != null) {
                welcomeSocket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        super.stopModule();
    }
}
```

Figure 6: extractValues() and stopModule() methods.


```
{"connected":false,"gameTime":"0001-01-
01T00:00:00Z","gamePaused":false,"telemetryPluginVersion":"0","gameVersion":"0.0","trailerAttached":false,"truckSpeed":0.0,
.0,"rotationZ":0.0,"gear":0,"gears":0,"gearRanges":0,"gearRangeActive":0,"engineRpm":0.0,"engineRpmMax":0.0,"fuel":0.0,"fue
0.0,"gameBrake":0.0,"gameClutch":0.0,"truckMass":0.0,"truckModelLength":0,"truckModelOffset":0,"trailerMass":0.0,"trailerId
01T00:00:00Z","sourceCity":"","destinationCity":"","sourceCompany":"","destinationCompany":"","retarderBrake":0,"shifterSlo
linkerLeftActive":false,"blinkerRightActive":false,"blinkerLeftOn":false,"blinkerRightOn":false,"lightsParkingOn":false,"li
ghtsReverseOn":false,"batteryVoltageWarning":false,"airPressureWarning":false,"airPressureEmergency":false,"adblueWarning":
eConsumpton":0.0,"oilPressure":0.0,"oilTemperature":0.0,"waterTemperature":0.0,"batteryVoltage":0.0,"lightsDashboard":0.0,"
```


Figure 7: Snippet of the raw data generated by the ETS2 Telemetry Server.

Main.java

The only modifications done in the *Main.java* class was to set the ip adress to match the localhost, and to start the previously created EuroTruck gateway and module, including a call to its *getProvidingSignals()* method. (Figure 8).

```
public class Main {
    public static void main(String[] args) throws Exception {
        String ipaddress = "127.0.0.1";
        SimulatorGateway gateway = new SimulatorGateway();
        gateway.addAndInitiateNode(ipaddress, 8251, null, null);
        gateway.addAndInitiateNode(ipaddress, 9898, null, null);
        gateway.addAndInitiateNode(ipaddress, 9899, null, null);
        Thread.sleep(1000);

        //Initialise EuroTruck gateway
        EuroTruck ets2 = new EuroTruck(gateway);
        for (int i : ets2.getProvidingSignals()) {
            gateway.provideSignal(i);
        }

        Thread.sleep(2000);
        //Start EuroTruck module
        ets2.startModule();
    }
}
```

Figure 8: Main.java class.