OpenDS

An Open-Source Driving Simulation Software for Research

Technical Documentation

Software version: 2.5

Creation date of document: 2014-10-23

1 Introduction

As full-fledged driving simulation software for the evaluation of automotive applications is high in price and low cost simulators often lack of extensibility, the implementation of a basic driving simulation toolkit has been considered to be a part of the EU project “GetHomeSafe”. The resulting software is supposed to be distributed online under open source license, which will allow researchers around the world to use and extend the software according to their needs and free of charge. In return, it is expected to gather valuable contributions from the community.

Former investigations indicated that there exist only a small number of established and freely available driving tasks (as the ISO standardized “Lane Change Test”) to measure driving performance and driver distraction and, moreover, these are not suitable for all kinds of use cases or research questions. If, for example, continuous access to steering performance is needed, unforeseeable events should be presented, or moderating effects of driving task difficulty on performance are in focus of investigation, researchers need to buy expensive tools and frequently extensive modeling effort is required. Another critical issue is that tracks and scenarios are implemented over and over again by researchers or institutes starting to work in this area (e.g. vehicle platooning task, way finding, complex crossroad scenarios) leading to unnecessary modeling effort and even worse this leads to differences in implementation and vocabulary.

Due to these reasons, a driving simulation software for research had to be implemented which allows measuring driving performance and driver distraction. As the resulting software is shared with developers from the researcher community, we do not intend to provide an overall toolkit for all kind of driving experiments, but rather a basic construction kit which can be extended by the user.

In the following, a basic technical documentation will be provided.

Table of Contents

2 Game Engine used by OpenDS

OpenDS is based on jMonkeyEngine[1] (jME), a high performance scene graph based graphics API. This open-source framework has been implemented in Java and has built up a reputation in game development. Its default renderer, the Lightweight Java Game Library (LWJGL), enables full OpenGL 2 through OpenGL 4 support. In version 3.0, the jME framework uses jBullet, a Java port of the Bullet Physics library, in use by top industry developers. Wrapping Bullet Physics library into jME3 objects assures easy interaction and future updates can include support for native bullet, including GPU acceleration. jBullet is a multi-threaded physics engine which allows mesh-accurate collision shapes and enables the experience of forces such as acceleration, friction, torque, gravity and centrifugal forces during simulation. The support of common model formats allows the simulator to load any 3D environment. Further features of jME's renderer are support of different lighting options (per-pixel lighting, multi-pass lighting, Phong lighting, tangent shading, and reflection), texturing (multi-texturing through shaders), and the capability to model special effects such as smoke, fire, rain, snow etc. Supported post processing and 2D filter effects are reflective water, shadow mapping, high dynamic range rendering, screen space ambient occlusion, light scattering, fog, and depth of field blur. Nifty GUI integration enables an easy-to-use toolkit for designing platform independent graphical user interfaces within the rendering frame, which is used for menus and message boxes during simulation. With regard to the development of OpenDS, jME's GUI node (used for speedometer and revmeter panels), multiple view ports (used for rear-view mirror), and basic audio support (used for playing positional and directional sounds) are useful features.

[1] http://jmonkeyengine.org

3 About OpenDS

OpenDS consists of two major components: the simulator and the drive analyzer. The simulation component is capable to load a driving task which is usually available in XML format. This task describes how a given map model will be equipped with additional road objects (such as traffic, signs, traffic lights, obstacles, etc.), events and several other parameters. All visible elements of a processed driving task will be added to the scene graph for rendering and all physical objects will be attached to the main physics node of the jBullet physics simulation. Major features of the extensible simulator implementation are different capabilities to control traffic lights (pre-defined cycles, red/green on approach, interactive external control), simulation of traffic and weather conditions, and a realistic engine and transmission model which can be used to compute the fuel consumption from the current pedal state considering the power needed to overcome rolling resistance, air resistance, inertia, and potential energy, as well as the engine's inner friction. Furthermore, events which have been defined in the driving task can be triggered under given conditions; e.g., set the driving car's position, let objects appear/disappear, move vehicles, perform reaction measurements, play a sound file, etc. The reaction data recorded in this way can be visualized for example as a bar chart with the integrated Jasper Report module and be exported to text or PDF format.

The second major component of OpenDS is the drive analyzer, which is able to visualize the car data recorded during a drive several times per second; e.g., position, direction, speed, and pedal states. It enables the experiment leader to reconstruct the exact simulation environment from an earlier drive in order to analyze the car's state in any position. Furthermore, the car's driven route can be compared to a pre-defined normative track in order to compute the deviation, which can be considered as a measure of driving performance.

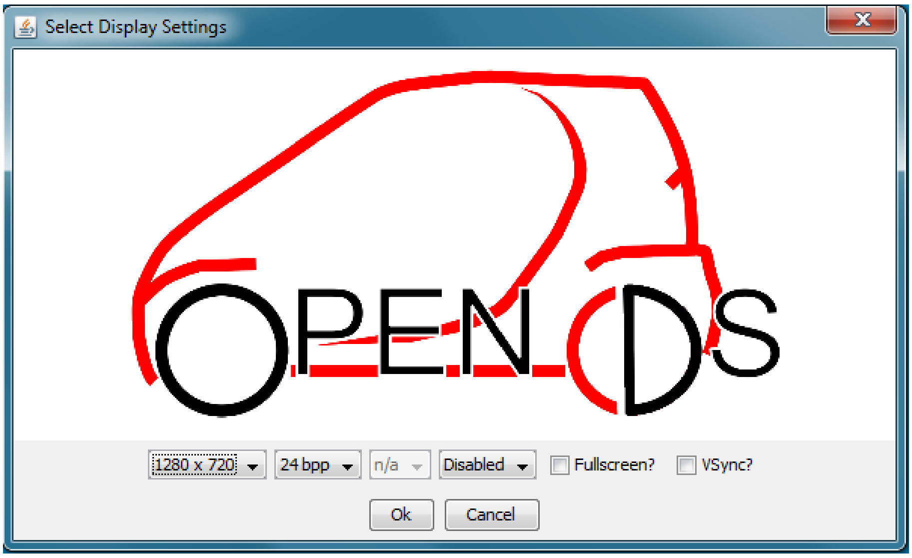
In order to facilitate a realistic simulation, OpenDS not only provides an interface for game controllers, but also a CAN-interface for connecting to real cars, which enables the simulator to request car properties – like steering angle and pedal states – and to provide the in-car devices with simulation values. To increase the driving experience, OpenDS supports the concept of column cameras which allows cylindrical projection and projection to multiple screens.

4 Running OpenDS

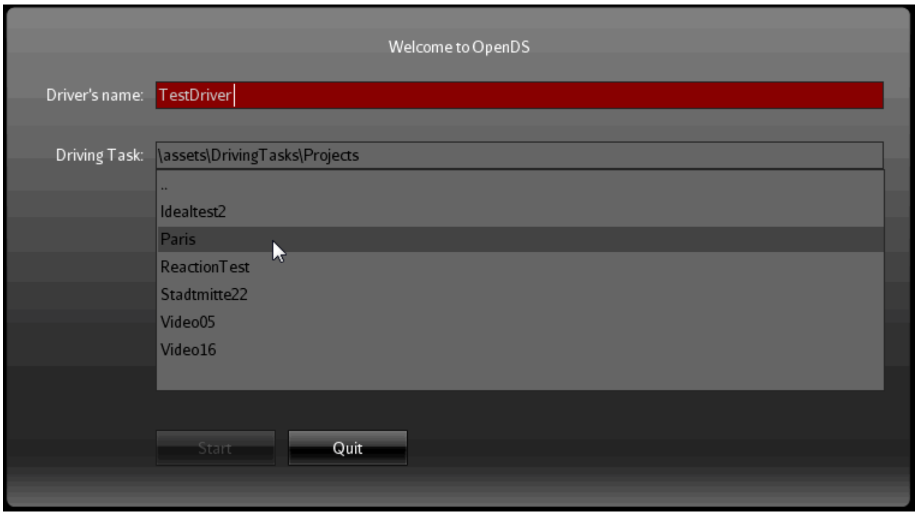
4.1 Running the built version

1. Double click 'OpenDS.jar'.

2. When the following splash screens shows up, select resolution and proceed with clicking 'OK'.



3. Specify driver's name (optional).



4. Select map model and click 'Start'.

4.2 Running from source code

The following steps show how to run OpenDS from source code in Eclipse. However, any other Integrated Development Environment can be used instead.

1. Start Eclipse and create a new Java Project (File → New → Java Project). Specify an appropriate name (e.g. OpenDS) and click 'Finish'.

2. Download the source code from the OpenDS homepage[2] and move the contents of the archive to folder 'src' of the new project.

3. Add a new folder 'lib' to your project and copy the contents of the library archive to that folder.

4. Add a new folder 'assets' to your project and copy the contents of the assets.zip archive to that folder.

5. Add all jar files (counting more than 100) that can be found in 'lib' or any of its sub-folders to the Build Path.

6. Right-click the project and select 'Build Path' → 'Configure Build Path...' to open the 'Properties'-dialog. Change to tab 'Libraries' and click 'Add Class Folder'. Select the check box of folder 'Logo' which can be found at 'assets/Textures' as well as the check box of folder 'log4j' which can be found at 'assets/JasperReports'. Click 'OK' to close both dialog windows.

7. Run OpenDS by right-clicking class ‘eu.opends.main.Simulator’ in Eclipse's Package Explorer and selecting 'Run As' → 'Java Application'.

8. When the OpenDS splash screen shows up, select resolution and proceed with clicking 'OK'. (c.f. screenshot ‘3.1 Running the built version’)

9. Specify driver’s name (optional).

10. Select map model and click 'Start'

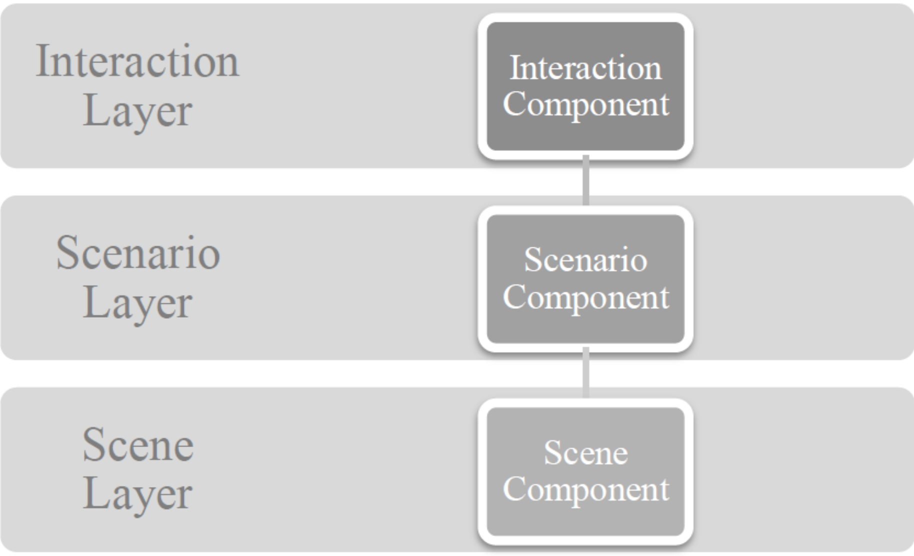
[2] http://www.opends.eu

5 Simulator

This chapter contains the user manual of the OpenDS simulation component. In the following references to the JavaDoc source code documentation[3], driving task documentation (including commented XML Schema files and examples) and the default key assignment table can be found.

5.1 Driving Tasks

An OpenDS driving task description always consists of the following three concepts: scene, scenario, and interaction. To run the simulation, a driving task description which is usually available as XML-files (scene.xml, scenario.xml, and interaction.xml, respectively) and a file containing simulator settings (settings.xml) are required.



All these files can be found in each project folder under:
`assets/DrivingTasks/Projects/<projectfolder>/`

5.1.1 Scene

In OpenDS, the concept of scene refers to the whole set of objects related to the driving environment, such as the driver's car, road signs, traffic lights, buildings, streets, traffic vehicles, etc. Each of these objects has a number of properties which can be specified in the corresponding

[3] http://opends.eu/JavaDoc/

scene.xml file. These properties describe for instance an element's location, shape, size and mass. Technical details can be found in the following.

Each scene.xml file contains the declaration of all objects that are relevant for a driving task. The declaration part is subdivided into the following sections: sounds, images, 3D models, geometries, reset points, gravity, and light.

```
<scene>
    <sounds>…</sounds>
    <pictures>…</pictures>
    <models>…</models>
    <geometries>…</geometries>
    <resetPoints>…</resetPoints>
    <gravity>…</gravity>
    <lights>…</lights>
</scene>
```

5.1.1.1 Sounds

Audio files can be declared within the <sounds> element and must contain at least a unique identifier and the file path. The identifier is required as reference e.g. to initiate the playback of a sound file from the interaction layer. Other properties such as volume and pitch, whether it is a positional or directional sound, and whether it is played in a loop can be optionally specified. All sound files which are supposed to be available for playing at runtime have to be declared in that element.

The following example shows a positional sound "beep" with given position, volume 50%, and no changes to the pitch. When starting the playback, this file will only be played once (c.f. loop = false)

```
<sound id="beep" key="Sounds/Effects/Beep.ogg">
    <positional value="true" >
        <translation>
            <vector>
                <entry>0</entry>
                <entry>-5</entry>
                <entry>3.5</entry>
            </vector>
        </translation>
    </positional>
    <directional value="false" />
    <loop>false</loop>
    <volume>0.5</volume>
```

```
    <pitch>1</pitch>
</sound>
```

5.1.1.2 Pictures

In the <pictures> element several images can be declared that are available for display in the rendering frame during runtime of the simulator. In addition to the essential properties: identifier, file path, and number of level (foreground/background), the user can optionally specify the position (absolute/relative), dimensions, and transparency of the image. Furthermore, an image's visibility at startup can be set. Using the unique identifier, the image can be addressed at runtime in order to change for example its visibility.

The following example shows a transparent 100x100px picture “image01” which is visible and located 10px from top and 20% from the left border of the simulation screen

```
<picture id="image01" key="Textures/image01.png" level="1">
    <vPosition>
        <fromTop unit="px" value="10" />
    </vPosition>
    <hPosition>
        <fromLeft unit="%" value="20" />
    </hPosition>
    <width>100</width>
    <height>100</height>
    <useAlpha>true</useAlpha>
    <visible>true</visible>
</picture>
```

5.1.1.3 Models

The <models> element is used to declare all models which are supposed to be available in the driving environment. At least a unique identifier and a file path (or reference to the underlying geometry) have to be specified. Other optional parameters are mass, initial visibility, collision shape, scale, rotation and initial position (called “translation”). If these parameters are not set default values will be applied instead. Parameters mass, visibility and collision shape are of simple type Float, Boolean and String, whereas, for scale, rotation and translation a vector of three Float values is expected. In case of using a quaternion to describe rotation, one can alternatively specify a vector of four Float values. All the models listed in this element will be processed at startup: depending on the visibility and collision shape values, a model will be added to the scene graph or to the physics node, respectively.

The following example shows a visible and “collidable” model “driverCar” which has a mass of 1200 kg and a given rotation (95° around the up axis) and position (unit: meter)

```xml
<model id="driverCar"
       key="Models/Cars/drivingCars/Mercedes/Car.scene">
    <mass>1200</mass>
    <visible>true</visible>
    <collisionShape>meshShape</collisionShape>
    <scale>
        <vector jtype="java_lang_Float" size="3">
            <entry>1</entry>
            <entry>1</entry>
            <entry>1</entry>
        </vector>
    </scale>
    <rotation quaternion="false">
        <vector jtype="java_lang_Float" size="3">
            <entry>0</entry>
            <entry>95</entry>
            <entry>0</entry>
        </vector>
    </rotation>
    <translation>
        <vector jtype="java_lang_Float" size="3">
            <entry>-792.395</entry>
            <entry>0.969</entry>
            <entry>-33.835</entry>
        </vector>
    </translation>
</model>
```

5.1.1.4 Geometries

The `<geometries>` element is used to declare one of the following predefined shapes: box, sphere, cylinder and point.

All of these geometries require the specification of a unique identifier. Declaring a box additionally requires the specification of dimensions (height, width, depth); a sphere declaration requires the specification of radius and number of samples to be used for approximating the curved surface; a cylinder declaration requires the specification of radius, height, number of samples to be used for approximating the curved surface, and whether it is closed; and a point declaration requires the specification of translation.

These geometries (except point) can be referenced in the model declaration, instead of specifying a file path. Points can only be referenced from the scenario layer in order to use them as ideal points or waypoints of computer-controlled vehicles.

```xml
<box id="box01">
    <width>2</width>
```

```xml
        <depth>0.1</depth>
        <height>11.5</height>
</box>
```

This example shows a box geometry “box01” with width 2m, depth 0.1m, and height 11.5m

```xml
<sphere id="sphere02">
    <samples axis="10" radial="10"/>
    <radius>5</radius>
</sphere>
```

This example shows a sphere geometry “sphere02” with radius 5m.

```xml
<cylinder id="cylinder03">
    <samples axis="20" radial="20" />
    <radius>0.5</radius>
    <height>5</height>
    <closed>true</closed>
</cylinder>
```

This example shows a closed cylinder geometry “cylinder03” with radius 0.5m and height 5m.

```xml
<point id="point04">
    <translation>
        <vector jtype="java_lang_Float" size="3">
            <entry>2</entry>
            <entry>0.4</entry>
            <entry>-86</entry>
        </vector>
    </translation>
</point>
```

This example shows a point geometry “point04” with a given position.

5.1.1.5 Reset Points

The <resetPoints> element is used to assign reset points consisting of a translation and rotation to a unique identifier. The driving car can be reset to such a position with the given orientation while runtime as the respective event has been triggered.

```xml
<resetPoint id="reset01">
    <translation>
        <vector jtype="java_lang_Float" size="3">
            <entry>-61</entry>
            <entry>0</entry>
            <entry>38</entry>
        </vector>
    </translation>
    <rotation quaternion="true">
        <vector jtype="java_lang_Float" size="4">
            <entry>0</entry>
            <entry>0.707107</entry>
            <entry>0</entry>
            <entry>-0.707107</entry>
```

```
        </vector>
    </rotation>
</resetPoint>
```

This example shows a reset point “reset01” with a given position (“translation”) and orientation (“rotation”) as quaternion.

5.1.1.6 Gravity

The <gravity> element allows specification of gravitational acceleration (unit: $\frac{m}{s^2}$).

5.1.1.7 Lights

The <light> element can be used to declare different light sources: point light (e.g. from a street lamp), directional light (e.g. sun light) and ambient light with specific parameters.

While point light requires the specification of position vector, radius, and color vector; directional light requires the specification of direction vector and color vector; and ambient light only requires the specification of a color vector.

```
<pointLight>
    <position>
        <vector jtype="java_lang_Float" size="3">
            <entry>5</entry>
            <entry>-20.5</entry>
            <entry>101.75</entry>
        </vector>
    </position>
    <radius>2</radius>
    <color>
        <vector jtype="java_lang_Float" size="4">
            <entry>1.0</entry>
            <entry>0.0</entry>
            <entry>0.0</entry>
            <entry>1.0</entry>
        </vector>
    </color>
</pointLight>
```

This example shows a point light source with a given position and a radius of 2 m emitting red (= RGBA [1,0,0,1]) light.

```
<directionalLight>
    <direction>
        <vector jtype="java_lang_Float" size="3">
            <entry>1</entry>
            <entry>0.5</entry>
            <entry>-5.5</entry>
        </vector>
    </direction>
```

```
    <color>
        <vector jtype="java_lang_Float" size="4">
            <entry>0.7</entry>
            <entry>0.7</entry>
            <entry>0.7</entry>
            <entry>1.0</entry>
        </vector>
    </color>
</directionalLight>
```

This example shows a directional light source with a given direction emitting white (= RGBA [1,1,1,1]) light dimmed by factor 0.7.

```
<ambientLight>
    <color>
        <vector jtype="java_lang_Float" size="4">
            <entry>0</entry>
            <entry>0</entry>
            <entry>1</entry>
            <entry>1</entry>
        </vector>
    </color>
</ambientLight>
```

This example shows global blue (= RGBA [0,0,1,1]) ambient light.

More details can be found in the schema description[4] of the scene.xml file.

5.1.2 Scenario

In OpenDS, the definition of scenario is similar to the one it has in a theater as it describes the role each actor has to play. It may provide semantic information for any object which has been declared in the scene layer before. Technical details can be found in the following.

Each scenario.xml file contains the semantic information that is relevant for a driving task. So far, the supported semantic information covers weather condition, the driving car and traffic vehicles. At a later stage, this file will also be used to annotate semantic information about the infrastructure (road signs, traffic lights ...) as well.

```
<scenario>
    <environment>…</environment>
    <driver>…</driver>
    <traffic>…</traffic>
    <road>…</road>
</scenario>
```

---

[4] http://opends.eu/drivingtask/sceneXSD.html

5.1.2.1 Environment

In the <environment> element, different weather settings can be adjusted; e.g. snow, rain and fog intensity can be given as percentage.

```
<weather>
    <snowingPercentage>100</snowingPercentage>
    <rainingPercentage>0</rainingPercentage>
    <fogPercentage>52</fogPercentage>
</weather>
```

5.1.2.2 Driver

In the <driver> element either the driving car may be described in more detail (<car> element) or alternatively a camera flight (<cameraFlight> element) can be described instead.

```
<cameraFlight>
    <speed>50</speed>
    <automaticStart>true</automaticStart>
    <automaticStop>true</automaticStop>
    <track>
        <point ref="startPoint" />
        <point ref="waypoint01" />
        <point ref="waypoint02" />
        <point ref="waypoint03" />
        <point ref="endPoint" />
    </track>
</cameraFlight>
```

The <car> element requires a reference to the underlying 3D object by passing the corresponding ID to the “ref”-attribute. Furthermore, reset points as well as tire, engine, transmission, suspension, wheel and brake settings can be specified.

```xml
<car ref="driverCar">
    <resetPoints>
        <resetPoint ref="reset01" />
        <resetPoint ref="reset02" />
    </resetPoints>
    <engine>
        <engineOn>true</engineOn>
        <minSpeed>0</minSpeed>
        <maxSpeed>180</maxSpeed>
        <acceleration>3.3</acceleration>
        <minRPM>750</minRPM>
        <maxRPM>7500</maxRPM>
    </engine>
    <transmission>
        <automatic>true</automatic>
        <reverse>3.182</reverse>
        <forward>
            <vector jtype="java_lang_Float" size="6">
                <entry>3.615</entry>
                <entry>1.955</entry>
                <entry>1.281</entry>
                <entry>0.973</entry>
                <entry>0.778</entry>
                <entry>0.646</entry>
            </vector>
        </forward>
    </transmission>
    <suspension>
        <stiffness>120</stiffness>
        <compression>0.2</compression>
        <damping>0.3</damping>
    </suspension>
    <wheel>
        <frictionSlip>50</frictionSlip>
    </wheel>
    <brake>
        <decelerationFreeWheel>2.0</decelerationFreeWheel>
        <decelerationBrake>8.7</decelerationBrake>
    </brake>
</car>
```

Furthermore, way points of a pre-defined normative track (<idealTrack> element) can be specified in the <driver> element as ordered list. This ideal line can be used by the Analyzer tool to compare it with the driven track in order to compute a measure for the driver's performance.

```xml
<idealTrack>
    <point ref="point_01" />
    <point ref="point_02" />
    <point ref="point_03" />
    <point ref="point_04" />
    <point ref="point_05" />
    <point ref="point_06" />
</idealTrack>
```

5.1.2.3 Traffic

The <traffic> element contains a list of all vehicles (in the future also pedestrians, bicycles, etc.) that are moving around autonomously following the waypoints in the attached <wayPoints> element. The vehicle properties that can be specified in the <vehicle> element are: the file path of the moving 3D object, its mass, acceleration and deceleration values, and whether the engine will be running initially. Properties related to the path that a vehicle is following are: the maximum foresight distance, the curve tension, whether the path is cyclic, whether the path is visible (debug mode), the starting waypoint, and the waypoint list containing an ordered set of 3D-coordinates and speed limits not to be exceeded at the corresponding way point.

```
<vehicle id="car01">
    <modelPath>Models/Cars/drivingCars/CarGreen/Car.scene</modelPath>
    <mass>800</mass>
    <acceleration>3.3</acceleration>
    <decelerationBrake>8.7</decelerationBrake>
    <decelerationFreeWheel>2.0</decelerationFreeWheel>
    <engineOn>true</engineOn>
    <maxDistanceFromPath>3.0</maxDistanceFromPath>
    <curveTension>0.05</curveTension>
    <pathIsCycle>false</pathIsCycle>
    <pathIsVisible>true</pathIsVisible>
    <startWayPoint>WayPoint_01</startWayPoint>
    <wayPoints>
        <wayPoint id="WayPoint_1">
            <translation>
                <vector jtype="java_lang_Float" size="3">
                    <entry>72.61561</entry>
                    <entry>0.108792834</entry>
                    <entry>-277.24188</entry>
                </vector>
            </translation>
            <speed>50</speed>
        </wayPoint>
        <wayPoint id="WayPoint_2">
            <translation>
                <vector jtype="java_lang_Float" size="3">
```

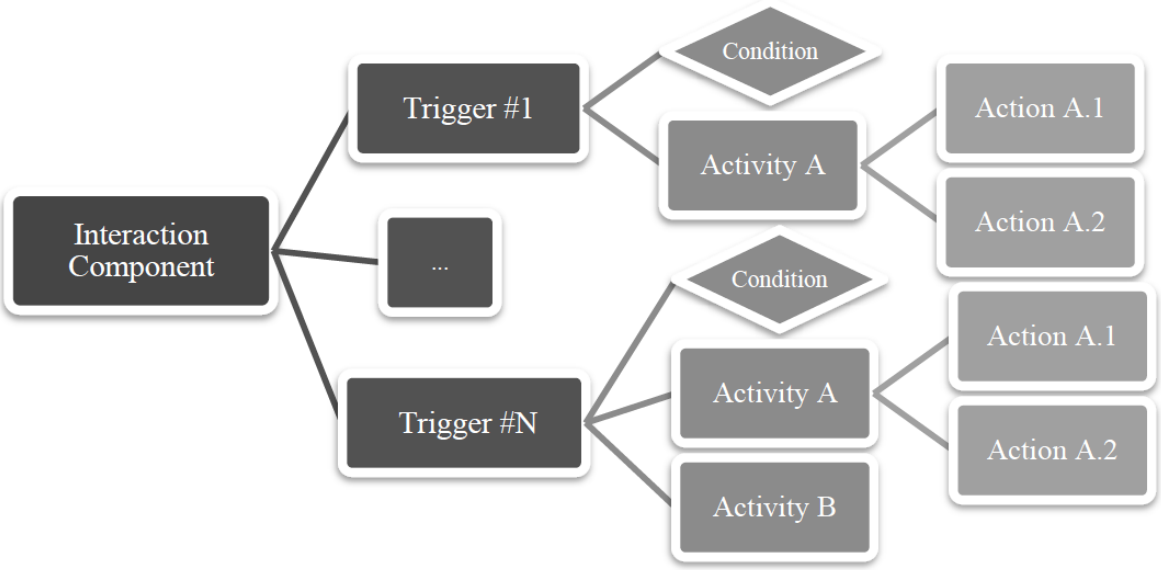
```
                    <entry>38.26012</entry>
                    <entry>0.108847</entry>
                    <entry>-229.40056</entry>
                </vector>
            </translation>
            <speed>50</speed>
        </wayPoint>
    </wayPoints>
</vehicle>
```

More details can be found in the schema description[5] of the scenario.xml file.

5.1.3 Interaction

On top of scene and scenario layer, the interaction layer may, at any time during the simulation, change the behavior parameters of any object triggered by a met condition. The interaction layer is subdivided into two parts: the activity declaration and the trigger declaration. In the trigger declaration part, a condition may be assigned to a list of activities (from the activity declaration part) which will be executed in a sequence when triggered. Each activity consists of a sequence of actions, which constitute the most atomic level of instructions that can be processed by OpenDS. Each action itself consists of a set of parameter-value-pairs describing the parameters to be updated when the corresponding action is performed. While the simulation is running, every trigger's condition is continuously checked. Once a condition is evaluated to true, the associated activities will be executed according to their sequential order. The following figure depicts how the interaction component's structure looks like.

[5] http://opends.eu/drivingtask/scenarioXSD.html



Technical details can be found in the following.

The interaction.xml file allows the definition of various events that could be triggered at runtime. For this purpose, a condition under which an event is triggered and an action to be executed can be specified. Examples for such a condition are the collision of the vehicle with a given road object and the stroke of a given key; examples of actions that could be triggered are: manipulate objects, pause simulation, reset the driving car, move traffic, play sound files, etc. A complete list can be found in the appendix.

The interaction layer is subdivided into the activity declaration and the trigger declaration.

```
<interaction>
    <activities>…</activities>
    <triggers>…</triggers>
</interaction>
```

5.1.3.1 Activities

In the `<activities>` element custom activities can be defined which consist of a sequence of actions – the most atomic level of instructions that can be processed by OpenDS. Each action itself consists of a set of parameter-value-pairs describing the parameters to be updated when the action is performed. In the following, the described structure is highlighted:

```
<activities>
```

```
    <activity id="activity01">
        <action id="" delay="" repeat="">
            <parameter name="" value="" />
            <parameter name="" value="" />
            <parameter name="" value="" />
            ...
        </action>
        <action id="" delay="" repeat="">
            <parameter name="" value="" />
            <parameter name="" value="" />
            <parameter name="" value="" />
            ...
        </action>
        ...
    </activity>
    <activity id="activity02">
        <action id="" delay="" repeat="">
            <parameter name="" value="" />
            <parameter name="" value="" />
            <parameter name="" value="" />
            ...
        </action>
        <action id="" delay="" repeat="">
            <parameter name="" value="" />
            <parameter name="" value="" />
            <parameter name="" value="" />
            ...
        </action>
        ...
    </activity>
    ...
</activities>
```

The <activities> element may contain any number of <activity> elements, which again may contain any number of <action> elements. In order to assign conditions to an activity, a unique identifier is required.

In the appendix, a complete collection of pre-defined actions which OpenDS is able to process is shown. They may consist of required, optional, and mutually exclusive parameters. In addition to the specific parameters of an action, the general parameters “delay” and “repeat” can be specified for all actions. These parameters allow to delay the execution of the corresponding action or to limit the number of maximum executions, respectively.

General Parameter	Type	Required	Default	Description
delay	float	no	0	Delay of execution (in seconds)
repeat	integer	no	0	Number of times a trigger can be used before it will be removed (0 = infinite)

The following example shows the “manipulateObject” action which will move, rotate, scale, and change the visibility of object “RoadworksSign1” which has been declared in the scene.xml file. This action will be executed immediately after triggering and can be repeated up to four times before the action will become invalid.

```
<action id="manipulateObject" delay="0" repeat="4">
    <parameter name="id" value="RoadworksSign1" />
    <parameter name="translationX" value="-81" />
    <parameter name="translationY" value="-1.693" />
    <parameter name="translationZ" value="-48" />
    <parameter name="rotationX" value="0" />
    <parameter name="rotationY" value="135" />
    <parameter name="rotationZ" value="0" />
    <parameter name="scaleX" value="0.02" />
    <parameter name="scaleY" value="0.02" />
    <parameter name="scaleZ" value="0.02" />
    <parameter name="visible" value="true" />
</action>
```

5.1.3.2 Triggers

In the <triggers> element, a single activity or a sequence of activities which have been declared in the <activities> element can be assigned to a condition by referencing the corresponding identifier. Alternatively, activities can be declared locally without referencing them. While the simulation is running, all conditions are continuously checked. Once a condition is evaluated to true, the associated activities will be executed according to their sequential order. In the following the structure of the triggers declaration part is highlighted:

```xml
<triggers>
    <trigger id="collide" priority="1">
        <activities>
            <!—referenced global activities -->
            <activity ref="activity01" />
            <activity ref="activity02" />
            <activity ref="activity03" />
            ...
            <!-- locally declared activities -->
            <activity id="localActivity01">
                <action id="manipulateObject" delay="0" repeat="1">
                    <parameter name="id" value="RoadWorksSign01" />
                    <parameter name="visible" value="true" />
                </action>
            </activity>
            ...
        </activities>
        <condition>collideWith:redBox</condition>
    </trigger>
    <trigger id="pressKey" priority="2">
        <activities>
            <activity ref="activity01" />
            <activity ref="activity04" />
            <activity ref="activity05" />
        </activities>
        <condition>pressKey:KEY_X</condition>
    </trigger>
    ...
</triggers>
```

The <triggers> element may contain any number of <trigger> elements, whereof each consists of an <activities> element and a <condition> element in order to assign a condition to a list of activities. Activities can either be declared locally in the <activities> element or referenced (using the "ref" attribute) from the activity declaration part above. Each trigger may be equipped with a unique identifier and a priority which determines the order of execution when two triggers will be fired at the same time (not implemented yet).

The following table shows the pre-defined conditions which OpenDS is currently able to process.

Condition Prefix	Description	Example
`collideWith:`	Trigger will be fired when the driving car	`collideWith:redBox`

	has collided with the given object	
`pressKey:`	Trigger will be fired when the given key has been pressed	`pressKey:KEY_X`

A list of available keys can be found at:
http://jmonkeyengine.org/javadoc/com/jme3/input/KeyInput.html

More details on the interaction.xml file can be found in the schema description[6].

5.1.4 Settings

Each driving task project contains a separate settings file which is required to customize the simulator to a specific task. This file differs from scene, scenario and interaction mentioned before, as it does not influence the simulated environment but rather the simulator's appearance. Technical details can be found in the following.

The settings.xml file is used to configure settings for the respective driving task. For example, one can specify whether the rear view mirror will be displayed. Furthermore, all external connections can be controlled from that file: already available is an interface for external visualization, a CAN server interface for controlling the driving car from external devices and an interface for controlling simulator settings from external applications. In addition, the settings.xml file can be used to configure input devices, like assignment of steering wheel axes as well as their sensitivity and the key assignment of the keyboard. Furthermore, the mouse wheel can be configured to setup the available zoom distance in the external camera mode.

Structure of settings.xml:

```
<settings>
    <general>…</general>
    <analyzer>…</analyzer>
    <externalVisualization>… </externalVisualization>
    <CANInterface>…</CANInterface>
    <settingsControllerServer>…</settingsControllerServer>
    <reactionMeasurement>…</reactionMeasurement>
    <controllers>
        <joystick>…</joystick>
        <keyboard>…</keyboard>
```

[6] http://opends.eu/drivingtask/interactionXSD.html

```
        <mouse>…</mouse>
    </controllers>
</settings>
```

In the following, the specification of all available parameters is listed in more detail with some examples.

General	General settings.		
Parameter	**Type**	**Default**	**Description**
driverName	string	“”	Name of the driver
showRearviewMirror	boolean	false	Rear view mirror will be visible at startup
showStats	boolean	false	Statistic renderer information (number of triangles, framerate, etc.) will be visible at startup
showAnalogIndicators	boolean	true	Analog speed and rpm indicators will be visible
showDigitalIndicators	boolean	false	Digital speed and rpm indicators will be visible
showFuelConsumption	boolean	false	Digital fuel consumption indicator will be visible
numberOfScreens	integer	1	Split screen vertically into x parts
angleBetweenAdjacentCameras	float	40	Angle (in degree) of horizontal rotation between two adjacent cameras (if number of screens > 1)
frameOfView	float	30.5	Opening angle of each camera (only if numberOfScreens > 1)
rearviewMirror/viewPortBottom	float	0.78	Position of bottom frame of rear view mirror (0.0 = bottom of screen, 1.0 = top of screen)
rearviewMirror/viewPortTop	float	0.98	Position of top frame of rear view mirror (0.0 = bottom of screen, 1.0 = top of screen)
rearviewMirror/viewPortLeft	float	0.3	Position of left frame of rear view mirror (0.0 = left of screen, 1.0 = right of screen)
rearviewMirror/viewPortRight	float	0.7	Position of right frame of rear view mirror (0.0 = left of screen, 1.0 = right of screen)

Example:

```
<general>
    <driverName>Peter</driverName>
    <showRearviewMirror>false</showRearviewMirror>
    <showStats>false</showStats>
    <showAnalogIndicators>true</showAnalogIndicators>
```

```
    <showDigitalIndicators>false</showDigitalIndicators>
    <showFuelConsumption>false</showFuelConsumption>
    <numberOfScreens>3</numberOfScreens>
    <angleBetweenAdjacentCameras>40</angleBetweenAdjacentCameras>
    <frameOfView>30</frameOfView>
    <rearviewMirror>
        <viewPortBottom>0.75</viewPortBottom>
        <viewPortTop>0.95</viewPortTop>
        <viewPortLeft>0.4</viewPortLeft>
        <viewPortRight>0.6</viewPortRight>
    </rearviewMirror>
</general>
```

analyzer		Analyzer settings.	
Parameter	**Type**	**Default**	**Description**
suppressPDFPopup	boolean	false	Disable automatic popup of analyzer PDF file after reaction measurements have been taken (PDF will be stored in analyzerData)

Example:

```
<analyzer>
    <suppressPDFPopup>false</suppressPDFPopup>
</analyzer>
```

externalVisualization		Settings for establishing a connection to an external visualization. Camera data (position and orientation) will be sent.	
Parameter	**Type**	**Default**	**Description**
enableConnection	boolean	false	Connection will be established at startup
Ip	string	“192.168.0.1”	IP-address of visualization server
Port	integer	1234	Port to be addressed on visualization server
updateRate	integer	25	Number of updates per second

scalingFactor	float	1.0	Scaling factor that will be applied to the camera position data.
sendPosOriAsOneString	boolean	false	Merge position and orientation data to one string. Otherwise, two strings will be sent.

Example:

```
<externalVisualization>
    <enableConnection>false</enableConnection>
    <ip>192.168.0.1</ip>
    <port>1234</port>
    <updateRate>20</updateRate>
    <scalingFactor>1</scalingFactor>

<sendPosOriAsOneString>false</sendPosOriAsOneString>
</externalVisualization>
```

CANInterface	Settings for establishing a connection to an external server providing data from a car's CAN-bus. Steering angle and pedal states can be received, speed and position data can be returned via feedback channel.		
Parameter	**Type**	**Default**	**Description**
enableConnection	boolean	false	Connection will be established at startup
Ip	string	"192.168.0.2"	IP-address of CAN server
Port	integer	5678	Port to be addressed on CAN server
updateRate	integer	20	Number of updates per second (only for feedback channel)
maxSteeringAngle	float	270.0	Value that will be provided by CAN server at steering stop (used to compute steering intensity (in %)

Example:

```
<CANInterface>
    <enableConnection>false</enableConnection>
```

```
    <ip>192.168.0.2</ip>
    <port>5678</port>
    <updateRate>20</updateRate>
    <maxSteeringAngle>180</maxSteeringAngle>
</CANInterface>
```

settingsControllerServer			Settings for establishing a connection to an external server controlling experiment settings while the simulation is running.
Parameter	**Type**	**Default**	**Description**
startServer	boolean	false	Server will be started at simulator startup
Port	integer	1000	Server will be listening on that port

Example:

```
<settingsControllerServer>
    <startServer>false</startServer>
    <port>1000</port>
</settingsControllerServer>
```

reactionMeasurement			Reaction groups (defined in interaction.xml to aggregate similar reaction measurements) can be assigned to pre-defined colors for bar chart visualization.
Parameter	**Type**	**Default**	**Description**
groupRed	string	“”	This reaction group will be colored red
groupGreen	string	“”	This reaction group will be colored green
groupYellow	string	“”	This reaction group will be colored yellow
groupCyan	string	“”	This reaction group will be colored cyan
groupBlue	string	“”	This reaction group will be colored blue
groupMagenta	string	“”	This reaction group will be colored magenta

Example:

```
<reactionMeasurement>
    <groupRed>reactionGroup01</groupRed>
    <groupGreen>reactionGroup02</groupGreen>
```

```
    <groupYellow>brakeReaction</groupYellow>
    <groupCyan>steeringReaction</groupCyan>
    <groupBlue>noReaction</groupBlue>
    <groupMagenta></groupMagenta>
</reactionMeasurement>
```

Keyboard	Settings related to the keyboard		
Parameter	**Type**	**Default**	**Description**
keyAssignments	List of function/key pairs		ID of game controller to be used (only if more than one device plugged in)

Notice:

Pre-defined simulator functions can be assigned to a list of keys. If the function has already been assigned to a key, the assignment will be overwritten. This can be used to replace default assignments.

Be aware: if you provide the empty string instead of a key, the function cannot be triggered anymore.

For a complete list of available functions see “Default Key Assignment” in the appendix

List of available keys: http://jmonkeyengine.org/javadoc/com/jme3/input/KeyInput.html

Example:

```
<keyboard>
    <keyAssignments>
        <!-- start engine with E-key -->
        <keyAssignment function="start_engine" key="KEY_E" />

        <!-- change camera view with V-key and C-key -->
        <keyAssignment function="toggle_cam" key="KEY_V,KEY_C" />

        <!-- disable horn -->
        <keyAssignment function="horn" key="" />
    </keyAssignments>
</keyboard>
```

joystick	Settings related to the game controller (e.g. steering wheel)

Parameter	Type	Default	Description
controllerID	Integer	0	ID of game controller to be used (only if more than one device plugged in)
steeringAxis	Integer	1	Specify which axis is the steering axis
invertSteeringAxis	boolean	false	Invert values returned from steering axis
steeringSensitivityFactor	Float	1.0	Increase/decrease sensitivity of steering
pedalAxis	Integer	2	Specify which axis is the pedal axis
invertPedalAxis	boolean	false	Invert values returned from pedal axis
pedalSensitivityFactor	float	1.0	Increase/decrease sensitivity of pedals
Notice: Make sure, that accelerator and brake pedal have been setup to use the same axis in your game controller's driver. In the future, there will be a way to assign game controller buttons to simulator functions (as already available for the keyboard)			

Example:

```
<joystick>
    <controllerID>0</controllerID>
    <steeringAxis>1</steeringAxis>
    <invertSteeringAxis>false</invertSteeringAxis>
    <steeringSensitivityFactor>1.0</steeringSensitivityFactor>
    <pedalAxis>2</pedalAxis>
    <invertPedalAxis>false</invertPedalAxis>
    <pedalSensitivityFactor>1.0</pedalSensitivityFactor>

    <!-- button assignment not available yet -->
    <keyAssignments>
        <keyAssignment function="start_engine" key="JOY_1" />
        <keyAssignment function="stop_engine" key="JOY_2" />
    </keyAssignments>
</joystick>
```

Mouse			Settings related to the mouse (only for camera movement in outside view)
Parameter	**Type**	**Default**	**Description**

minScrollZoom	float	1.0	Minimum camera distance from car (when using scroll wheel) in meters
maxScrollZoom	float	40.0	Maximum camera distance from car (when using scroll wheel) in meters
scrollSensitivityFactor	float	5.0	Sensitivity of scroll wheel

Example:

```
<mouse>
    <minScrollZoom>1.0</minScrollZoom>
    <maxScrollZoom>40.0</maxScrollZoom>
    <scrollSensitivityFactor>5.0</scrollSensitivityFactor>
</mouse>
```

More details on the settings.xml file can be found in the schema description[7].

[7] http://opends.eu/drivingtask/settingsXSD.html

6 Miscellaneous

6.1 Converting models

In order to convert a 3D model to a simulator compatible file format (OgreXML) one can use the OgreXML-Exporter for Blender. To get it running proceed as follows:

1. Install Blender[8]. Make sure to install version 2.49b as there are known problems with newer versions.

2. Install Python[9]. This is not needed for Blender itself, but for the exporter script. Choose the same Python version that is used as base for your Blender version (you'll see it when you start Blender).

3. Copy the meshes exporter[10] to the Blender scripts folder.
   Note: For Blender 2.49b and Windows 7 this folder is located at:
   C:\Users\<name>\AppData\Roaming\Blender Foundation\Blender\.blender\scripts

4. Copy the scene exporter[11] to the same folder as the meshes exporter.

More details on how to use the meshes and scene exporter can be found here:

http://www.ogre3d.org/tikiwiki/Blender+Exporter

http://www.ogre3d.org/tikiwiki/Blender+dotScene+Exporter

Wen converting models from *.obj format to OgreXML it turned out to be helpful to use the following settings when importing the *.obj model to Blender.

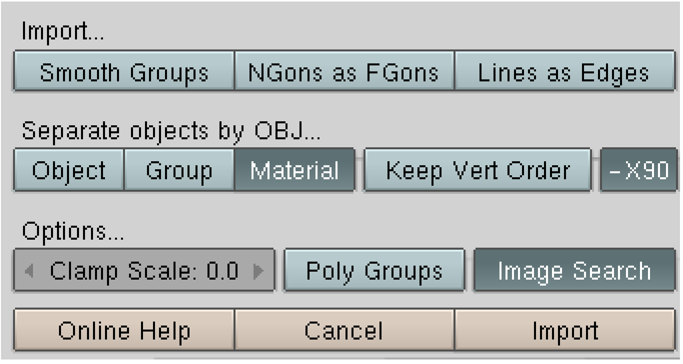
---

[8] http://blender.org

[9] http://www.python.org/download/releases

[10] http://www.xullum.net/lefthand/downloads/temp/BlenderExport.zip

[11] http://ogreaddons.svn.sourceforge.net/viewvc/ogreaddons/trunk/blendersceneexporter/ogredotscene.py



After importing, select all parts you want to be included in your OgreXML export and proceed with the scene exporter (File→ Export → OGRE Scene) and meshes exporter (File→ Export → OGRE Meshes) as described in the tutorials mentioned above.

6.2 CAN interface

In order to setup a simulator-to-car communication, you will have to provide a TCP-server which receives data from a car's CAN-bus and forwards this data to the simulator. In the settings.xml you can specify which server IP and port have to be addressed by the simulator (client) in the `<CANInterface>` element. After the connection has been established, the simulator is waiting for the following instructions:

Steering angle (from -x to +x, where x is the max. steering angle specified in settings.xml):

```
<message>
    <action name="steering">-92.5</action>
</message>
```

Acceleration (values in % from 0.0 to 1.0):

```
<message>
    <action name="acceleration">0.5</action>
</message>
```

Brake (values in % from 0.0 to 1.0):

```
<message>
```

```
    <action name="brake">0.3</action>
</message>
```

Change view:

```
<message>
    <action name="button">cs</action>
</message>
```

Reset driving car:

```
<message>
    <action name="button">return</action>
</message>
```

6.3 Graphical User Interfaces

The graphical user interfaces, which are available in OpenDS, have been created with the Nifty GUI toolkit which is fully integrated into jMonkeyEngine. The layouts of the user interfaces are stored in xml format in subfolder “Interfaces” of the “assets” folder:

File	**Description**
AnalyzerFileSelectionGUI.xml	GUI for selection of a drive data file for the analyzer component
DrivingTaskSelectionGUI.xml	GUI for selection of a driving task file for the simulation component
InstructionScreenGUI.xml	GUI for showing instructions
KeyMappingGUI.xml	GUI for showing all key assignments (when F1 key was pressed)
MessageBoxGUI.xml	Message box for displaying messages during simulation
ShutDownGUI.xml	GUI showing shut down dialog (when ESC key was pressed)

In package “eu.opends.niftyGui” of the OpenDS source code, event handlers for the available GUIs can be found.

More Information about the use of Nifty GUI can be found in the tutorials of the jMonkeyEngine site[12] and in the Nifty GUI user manual[13].

[12] http://jmonkeyengine.org/wiki/doku.php/jme3:advanced:nifty_gui

[13] http://sourceforge.net/projects/nifty-gui/files/nifty-gui/1.3.2/nifty-gui-the-manual-1.3.2.pdf/download

7 Appendix

7.1 Default Key Assignment of Simulator

The following table shows the default keys that are available in the simulator. The function ID can be used in the settings.xml file to change the key assignments manually.

Default key	Function ID	Description
UP	Accelerate	Accelerate car
DOWN	accelerate_back	Accelerate car backwards
SPACE	Brake	Brake car
F5	close_instruction_screen	Close instruction screen
F	hazard_lights	Flash hazard lights
H	Horn	Horn
SPACE	report_landmark	Report landmark
G	report_reaction	Report reaction
R	reset_car	Reset car to next reset position
1	reset_car_pos1	Reset car to reset position 1 (start position)
2	reset_car_pos2	Reset car to reset position 2 (if defined)
3	reset_car_pos3	Reset car to reset position 3 (if defined)
4	reset_car_pos4	Reset car to reset position 4 (if defined)
5	reset_car_pos5	Reset car to reset position 5 (if defined)
6	reset_car_pos6	Reset car to reset position 6 (if defined)
7	reset_car_pos7	Reset car to reset position 7 (if defined)
8	reset_car_pos8	Reset car to reset position 8 (if defined)
9	reset_car_pos9	Reset car to reset position 9 (if defined)
0	reset_car_pos10	Reset car to reset position 10 (if defined)
T	reset_fuel_consumption	Reset fuel consumption
F9	rotate_object_left	ObjectLocator: Rotate object to the left
F7	rotate_object_left_fast	ObjectLocator: Fast rotate object to the left
F10	rotate_object_right	ObjectLocator: Rotate object to the right
F8	rotate_object_right_fast	ObjectLocator: Fast rotate object to the right
F11	set_object	ObjectLocator: Place object on the map
PGDN	shift_down	Shift gear down
PGUP	shift_up	Shift gear up
ESCAPE	Shutdown	Exit simulator
O	start_pause	Pause simulation
LEFT	steer_left	Steer car to the left
RIGHT	steer_right	Steer car to the right
I	stop_pause	Resume simulation
END	toggle_automatic	Toggle between automatic/manual transmission

BACK	toggle_backmirror	Show/hide back view mirror
V	toggle_cam	Change camera view
RETURN	toggle_cinematics	Enable/disable camera flight
E	toggle_engine	Engine on/off
L	toggle_headlight	Toggle head light state (off-1-2)
F1	toggle_keymapping	Show/hide key mapping
M	toggle_messagebox	Show/hide message box
D	toggle_min_speed	Cruise control on/off
F12	toggle_object	ObjectLocator: select next object
P	toggle_pause	Pause/resume simulation
F6	toggle_physics_debug	Show/hide physics debug
S	toggle_record_data	Start/stop recording of car data
F4	toggle_stats	Show/hide statistical renderer information
A	toggle_trafficlightmode	Change traffic light mode (trigger-program-external-blinking-off)
W	toggle_wireframe	Show/hide wire frame
J	turn_left	Flash left turn signal
K	turn_right	Flash right turn signal

7.2 Default Key Assignment of Analyzer

The following table shows the default keys that are available in the simulator.

Default key	Description
F1	Show/hide key mapping
ESCAPE	Exit analyzer
V	Change camera view
1	Show/hide points
2	Show/hide line
3	Show/hide cone
UP	Move camera position to next data point
DOWN	Move camera position to previous data point
LEFT	Move camera position backward
RIGHT	Move camera position forward

7.3 Available Events

In the following a complete list of available events is shown. These events can be referenced to in the interaction.xml file (element `<action>`).

7.3.1 sendMessage

sendMessage	Outputs text to the screen for the given amount of seconds			
Parameter	**Type**	**Required**	**Default**	**Description**
Text	string	yes		Text to display on screen
Duration	Integer	no	1	Amount of seconds to show text (0 = infinite)

7.3.2 manipulateObject

manipulateObject	Manipulates translation, rotation, scale and/or visibility of the given model			
Parameter	**Type**	**Required**	**Default**	**Description**
Id	string	yes		ID of the model to manipulate
setTranslationX	float	No	0.0	Translate model to this x-coordinate
setTranslationY	float	No	0.0	Translate model to this y-coordinate
setTranslationz	float	No	0.0	Translate model to this z-coordinate
setRotationX	float	No	0.0	Rotate model around x-axis
setRotationY	float	No	0.0	Rotate model around y-axis
setRotationZ	float	No	0.0	Rotate model around z-axis
setScaleX	float	No	1.0	Scale model to this x-coordinate
setScaleY	float	No	1.0	Scale model to this y-coordinate
setScaleZ	float	No	1.0	Scale model to this z-coordinate
addTranslationX	float	No	0.0	Adds this value to the model's x-coordinate
addTranslationY	float	no	0.0	Adds this value to the model's y-coordinate
addTranslationz	float	no	0.0	Adds this value to the model's z-coordinate
addRotationX	float	no	0.0	Adds this value to the models rotation around the x-axis
addRotationY	float	no	0.0	Adds this value to the models rotation around the y-axis
addRotationZ	float	no	0.0	Adds this value to the models rotation around the z-axis
addScaleX	float	no	0.0	Adds this value to the models x-coordinate scale
addScaleY	float	no	0.0	Adds this value to the models y-coordinate scale
addScaleZ	float	no	0.0	Adds this value to the models z-coordinate scale

visible	boolean	no	true	Makes the model (in)visible

7.3.3 manipulatePicture

manipulatePicture	Manipulates visibility of the given picture			
Parameter	**Type**	**Required**	**Default**	**Description**
id	string	yes		ID of the picture to manipulate
visible	Boolean	yes		Makes the picture (in)visible

7.3.4 pauseSimulation

pauseSimulation	Pauses the simulation for the given amount of time			
Parameter	**Type**	**Required**	**Default**	**Description**
duration	integer	no	1	Amount of seconds to pause (0 = infinite)

7.3.5 startRecording

startRecording	Starts recording driver information			
Parameter	**Type**	**Required**	**Default**	**Description**
Track	integer	No	1	ID of recording

7.3.6 stopRecording

stopRecording	Stops recording driver information			
Parameter	**Type**	**Required**	**Default**	**Description**

7.3.7 resetCar

resetCar	Moves the driving car to the given reset point			
Parameter	**Type**	**Required**	**Default**	**Description**
resetPointID	string	yes		ID of the reset point to move the driving car to

7.3.8 moveTraffic

moveTraffic	Moves a traffic vehicle to the given way point			
Parameter	**Type**	**Required**	**Default**	**Description**
trafficObjectID	string	yes		ID of the traffic vehicle to move
wayPointID	string	yes		ID of the way point to move the traffic vehicle to

7.3.9 setCurrentSpeedLimit

setCurrentSpeedLimit	Sets the speed limit to the given value			
Parameter	**Type**	**Required**	**Default**	**Description**
speedLimit	integer	No	0	Speed limit in kph (0 = unlimited)

7.3.10 setUpcominigSpeedLimit

setUpcomingSpeedLimit	Sets the upcoming speed limit to the given value			
Parameter	**Type**	**Required**	**Default**	**Description**
speedLimit	integer	no	0	Speed limit in kph (0 = unlimited)

7.3.11 measureTimeUntilBrake

measureTimeUntilBrake	Measures time until brake was applied			
Parameter	**Type**	**Required**	**Default**	**Description**
triggerName	string	yes		ID of trigger for identification in output file

7.3.12 measureTimeUntilSpeedChange

measureTimeUntilSpeedChange		Measures time until speed was changed by the given amount		
Parameter	**Type**	**Required**	**Default**	**Description**
triggerName	string	yes		ID of trigger for identification in output file
speedChange	integer	yes		Amount of speed (in kph) that has to be in- or decreased

7.3.13 playSound

playSound		Plays a sound file specified in the scene layer		
Parameter	**Type**	**Required**	**Default**	**Description**
soundID	string	yes		ID of sound file to play

7.3.14 requestGreenTrafficLight

requestGreenTrafficLight			Requests a given traffic light to turn green	
Parameter	**Type**	**Required**	**Default**	**Description**
trafficLightID	string	yes		ID of traffic light to request for green

7.3.15 setupKeyReactionTimer

setupKeyReactionTimer			Sets up a reaction timer for keyboard and game controller button input	
Parameter	**Type**	**Required**	**Default**	**Description**
timerID	string	no	“timer1”	ID of timer to be used for measurement. If timer is in use, previous measurement will result in a missed reaction
reactionGroup	string	yes		Assign reaction measurement to a group (e.g. for color representation defined in “reactionGroup” in settings.xml)

correctReaction	string	yes		List of keys triggering the correct reaction (e.g. “Key_H, Joy_1”)
failureReaction	string	yes		List of keys triggering the failure reaction (e.g. “Key_G, Joy_2”)
comment	string	no	“”	Comment (will be forwarded to output)

7.3.16 setupLaneChangeReactionTimer

setupLaneChangeReactionTimer				Sets up a reaction timer for lane changing
Parameter	**Type**	**Required**	**Default**	**Description**
timerID	string	no	“timer1”	ID of timer to be used for measurement. If timer is in use, previous measurement will result in a missed reaction
congruenceClass	string	yes		Assign reaction measurement to a group (e.g. for color representation defined in “reactionGroup” in settings.xml)
startLane	string	yes		Lane where lane change must start from
targetLane	string	yes		Lane where lane change must end
minSteeringAngle	Float	no	0.0	Minimal steering angle that has to be overcome (in percent)
taskCompletionAfterTime	Float	no	0.0	Task must be completed after x milliseconds (0 = no limit)
taskCompletionAfterDistance	Float	no	0.0	Task must be completed after x meters (0 = no limit)
allowBrake	boolean	no	true	Driver may brake while changing lanes? (If false, failure reaction will be reported)
holdLaneFor	float	no	0.0	Number of milliseconds the target lane must be kept
failSound	string	no		Sound file that will be played after failed/missed lane change
successSound	string	no		Sound file that will be played after failed/missed lane change
Comment	string	no	“”	Comment (will be forwarded to output)

7.3.17 setupBrakeReactionTimer

setupBrakeReactionTimer			Sets up a reaction timer for braking	
Parameter	**Type**	**Required**	**Default**	**Description**
timerID	string	no	“timer1”	ID of timer to be used for measurement. If timer is in use, previous measurement will result in a missed reaction
congruenceClass	string	yes		Assign reaction measurement to a group (e.g. for color representation defined in “reactionGroup” in settings.xml)
startSpeed	float	no	80.0	Minimum speed the car must drive to start reaction measurement
targetSpeed	float	no	60.0	Maximum speed the car must drive to stop reaction measurement
mustPressBrakePedal	boolean	no	True	Driver must press brake pedal for successful reaction
taskCompletionAfterTime	float	no	0.0	Task must be completed after x milliseconds (0 = no limit)
taskCompletionAfterDistance	float	no	0.0	Task must be completed after x meters (0 = no limit)
allowLaneChange	boolean	no	True	Driver may change lanes while braking? (If false, failure reaction will be reported)
holdSpeedFor	float	No	0.0	Number of milliseconds the target speed must be kept
failSound	string	No		Sound file that will be played after failed/missed braking
successSound	string	no		Sound file that will be played after failed/missed braking
comment	string	no	“”	Comment (will be forwarded to output)

7.3.18 openInstructionScreen

openInstructionsScreen			Shows up a screen with instructions	
Parameter	**Type**	**Required**	**Default**	**Description**
instructionID	string	yes		ID of instruction screen to show (can be defined in assets/Interface/InstructionScreenGUI.xml)

7.3.19 setTVPTStimulus

setTVPTStimulus	Sets a stimulus for the Three-Vehicle-Platoon-Test			
Parameter	**Type**	**Required**	**Default**	**Description**
stimulusID	string	yes		ID of stimulus to trigger: - “brakeLight” (lead vehicle brake light), - “turnSignal” (follow vehicle turn signal), - “speedReduction” (lead vehicle deceleration), - “emergencyBrake” (lead vehicle brake light and deceleration)

7.3.20 writeToKnowledgeBase

writeToKnowledgeBase	Inserts/edits a property in the knowledge base			
Parameter	**Type**	**Required**	**Default**	**Description**
path	string	yes		Path of property to insert/edit
propertyName	string	yes		Name of property to insert/edit
propertyValue	string	yes		Value of property to insert/edit
propertyType	string	yes		Type of property to insert/edit