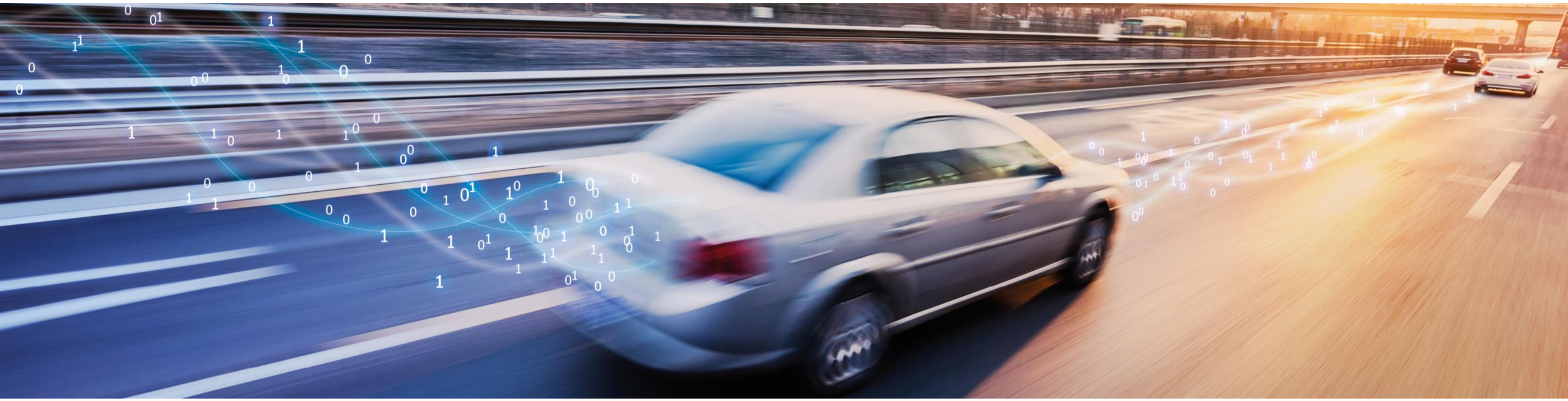
# ASAM OpenSCENARIO 1.0.0

## Release Presentation, Technical Overview & Scenario Creation

**Dr.-Ing. Ludwig Friedmann**

**Solution Architect Simulation Autonomous Driving, BMW AG**

18.03.2020
Munich



ASAM Association for Standardization of Automation and Measuring Systems

## Release Presentation

## Introduction

OpenSCENARIO is used in driving simulation and in virtual development, test and validation of driving assistance functions, automated and autonomous driving.

Within these use cases, OpenSCENARIO describes the dynamic content of the world, i.e. the entities acting on or interacting with the road network. OpenSCENARIO does not describe the road network, road infrastructure or road surface.

OpenSCENARIO was transferred to ASAM by an industry consortium in late 2018. It evolved to the ASAM standard OpenSCENARIO 1.0.0 within the ASAM OpenSCENARIO Transfer project.

## Motivation

Scenarios are essential for testing, validating and certifying the safety of driver assistance systems and autonomous driving cars. The industry, certification agencies and government authorities jointly work on the definition of scenario databases, which can be used to test and validate the safe operation of such systems.

OpenSCENARIO supports this endeavor by enabling the exchange and usability of scenarios in various simulation applications. With the help of this standardization effort, large numbers of critical situations can be run across various simulators. Thus, compared to road testing in real traffic, the amount of driven test kilometers in field tests can be significantly reduced.

The overall goal of ASAM OpenSCENARIO 1.0.0 was to create a standardized scenario description format which provides a quality- and completeness-level that is expected from a public standard and from ASAM members.

# New Features

### Creation of a Data Model and Derived Schema Files

- UML Data Model
- XML Schema Files

### Creation of Specification Documents

- Specification Programmers Reference Guide
- Specification User Guide

### Creation of Comprehensive Examples, Evaluation of Deficits and Potential Improvements

- 9 Examples
- 44 Bugzilla Items

## Other Changes

### Clarification and Technical Improvement

- Coordinate Systems
- Storyboarding and StoryboardElements (state machine, transitions and runtime behavior)
- Parameters and Catalogs
- Triggers, Conditions and ConditionEdges
- TrafficActions
    - TrafficSinkAction
    - TrafficSourceAction
    - TrafficSwarmAction
- RoutingActions
    - AssignRouteAction
    - FollowTrajectoryAction
    - FollowRouteAction
- SynchronizeAction
- Routes and Trajectories
- …

## Backward Compatibility

ASAM OpenSCENARIO 1.0.0 and the predecessor version 0.9.1 differ in terms of semantics, naming and structure. As consequence, version 1.0.0 cannot provide backward compatibility to version 0.9.1.

Instead, OpenSCENARIO 1.0.0 provides an XSLT migration script to transform valid files of the earlier version 0.9.1 into valid OpenSCENARIO 1.0.0 files. Within this script, each element of the 0.9.1 version has a template that transforms and reshapes the element to OpenSCENARIO 1.0.0.

## Relation to Other Standards

### ASAM OpenDRIVE

In order to use semantic road network information within ASAM OpenSCENARIO, the road network description ASAM OpenDRIVE can be referenced.

### ASAM OpenCRG

Road surface profiles defined by OpenCRG can be referenced from the before mentioned OpenDRIVE road network description and thus complement the two other standards.

## Deliverables

### Documents

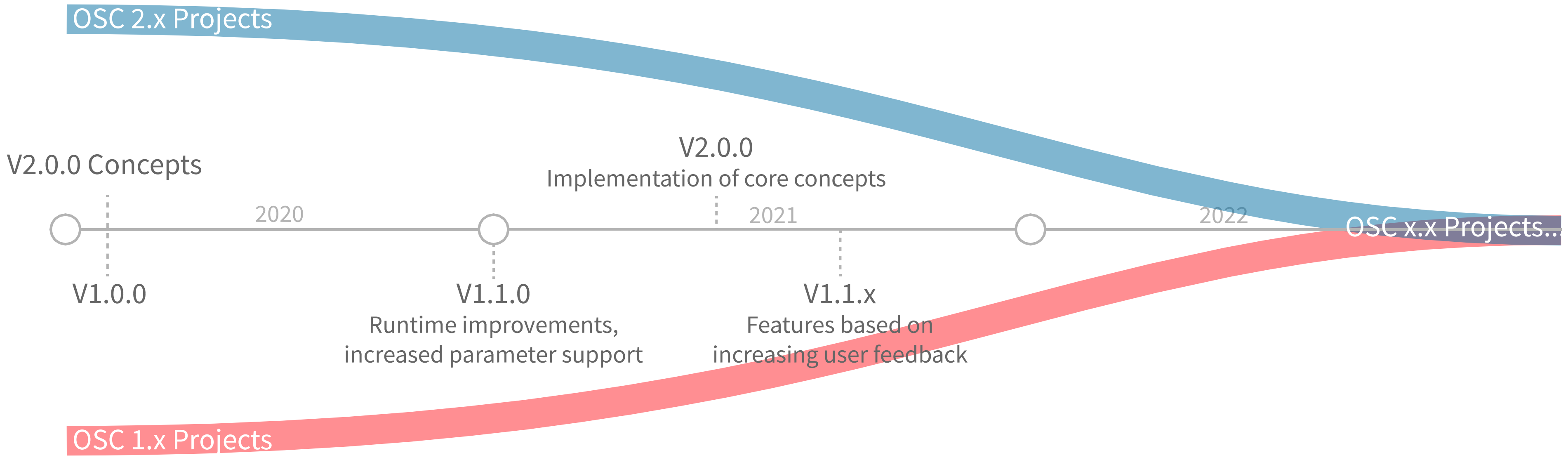
- Specification Programmers Reference Guide
- Specification User Guide

### Data Model and Supplementary Files

- UML Data Model
- UML Modeling Rules
- HTML Documentation
- XML Schema File
- Examples
- Migration Script (0.9.1 –> 1.0.0)
- List of Analyzed Deficits and Proposed Improvements

## Outlook

**The Proposal Workshop for a follow-up “ASAM OpenSCENARIO 1.x” project will take place on March 25th ([https://www.asam.net/project-detail/asam-openscenario-v1x/](https://www.asam.net/project-detail/asam-openscenario-v1x/)).**


OSC 2.x Projects
V2.0.0 Concepts
V2.0.0
Implementation of core concepts
2020
2021
2022
OSC x.x Projects...
V1.0.0
V1.1.0
Runtime improvements, increased parameter support
V1.1.x
Features based on increasing user feedback
OSC 1.x Projects

Goals of the ASAM OpenSCENARIO 1.x project:

- Completion of tasks that could not be completed within ASAM OpenSCENARIO 1.0.0
- Support for users and implementers of OpenSCENARIO 1.0.0 and 1.x
- Close collaboration with proposed ASAM OpenSCENARIO 2.0 project

## ASAM OpenSCENARIO 1.0.0

Technical Overview

## Technical Overview

## Fundamental Concepts

1. **Road Network** is populated by **Entities** (Vehicles, Pedestrians and MiscObjects, interaction defined by the Storyboard).

2. **Storyboard** contains at least one **Story** (structuring Acts, ManeuverGroups, Maneuvers, Events and Actions)

3. **Actions** are triggered by **Conditions** in **Triggers** (Triggers also start Acts and Events or stop Acts and the Storyboard)
4. **Catalogs** and **Parameters** allow re-use of Scenario elements

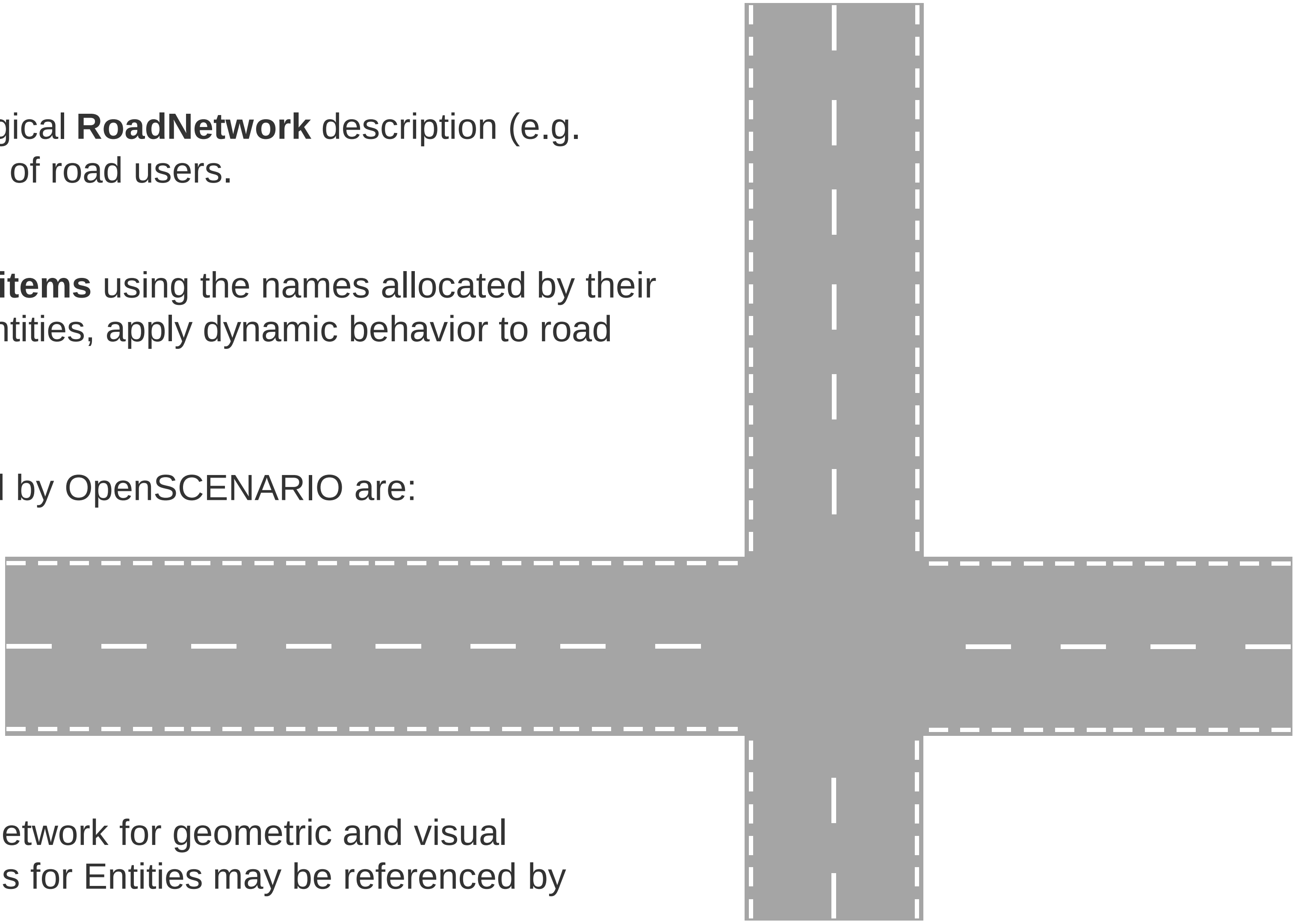
## Road Networks and 3D Models

OpenSCENARIO uses a **reference** to the logical **RoadNetwork** description (e.g. an OpenDRIVE file) to describe the behavior of road users.

OpenSCENARIO references **RoadNetwork items** using the names allocated by their own file format (e.g. to locate and position Entities, apply dynamic behavior to road infrastructure).

Examples for RoadNetwork items referenced by OpenSCENARIO are:

- Individual road
- Lane within a road
- Traffic signal
- Traffic signal controller

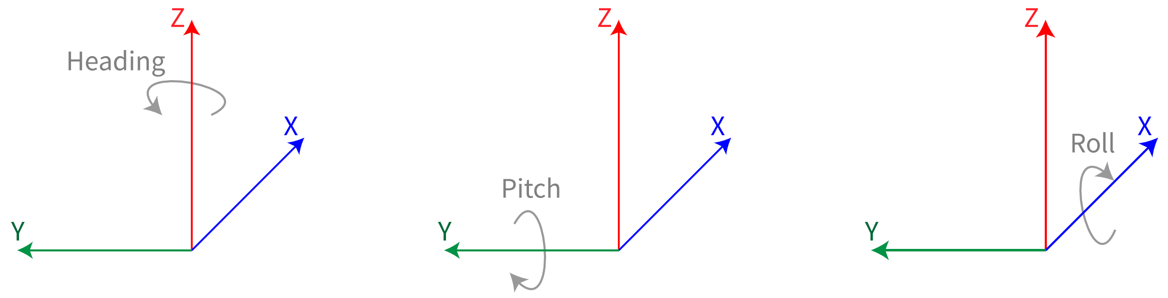


**3D models** can be referenced by the RoadNetwork for geometric and visual representation of the environment. 3D models for Entities may be referenced by the simulator based on Entity naming.

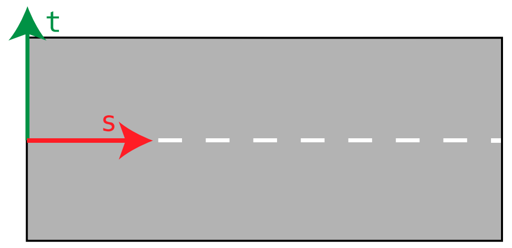
# Coordinate System Types

In OpenSCENARIO, two coordinate system types are used:

- Right handed **Cartesian coordinate systems**, compliant with ISO 8855:2011 definition. Orientation is expressed by heading(yaw)-pitch-roll sequence.



- Right handed, **road based coordinate systems** defined by coordinate axes associated with the reference line of the road (s-axis) and the direction orthogonal to it (t-axis).

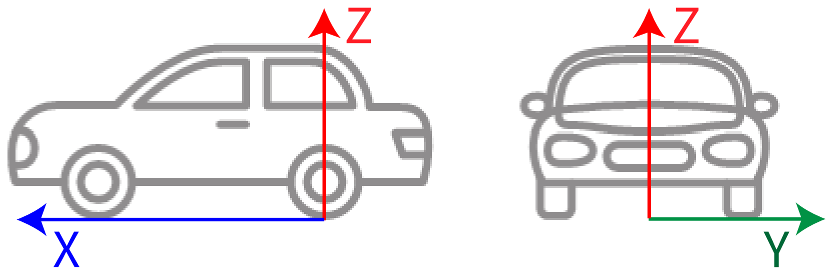


# Coordinate Systems and Positioning/Localization

## Coordinate Systems

The afore mentioned coordinate system types are referenced to create the following coordinate systems

- World coordinate system ($X_w$, $Y_w$, $Z_w$)
- Road coordinate system (s, t)
- Vehicle coordinate system (Xv, Yv, Zv)
- Pedestrian / MiscObject coordinate system ($X_{p/m}$ , $Y_{p/m}$ , $Z_{p/m}$)



## Positioning/Localization

OpenSCENARIO provides various ways to position or localize Entities:

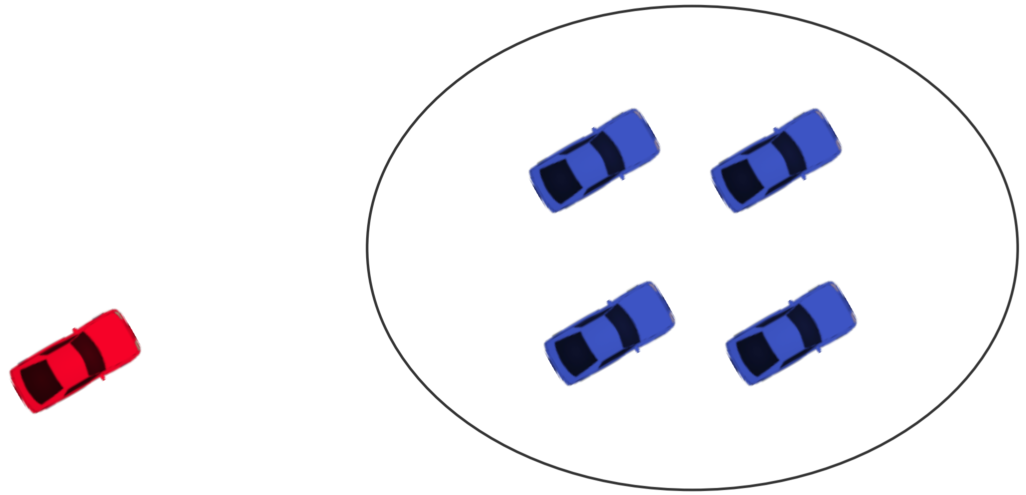
- Absolute/relative in the World coordinate system
- Relative to another Entity
- Absolute/relative in the Road coordinate system
- Absolute/relative in the Lane coordinate system
- Relative to a Route

## Entities

Entities are **objects** within a scenario that can (but do not have to) **change their location dynamically** over time.

Entities can be **Vehicles**, **Pedestrians** or **MiscObjects**:

- Obstacle
- Pole
- Tree
- Vegetation
- Barrier
- …



**Actions** can change the state of an Entity, e.g. its position, speed, or Controller (user defined or default).

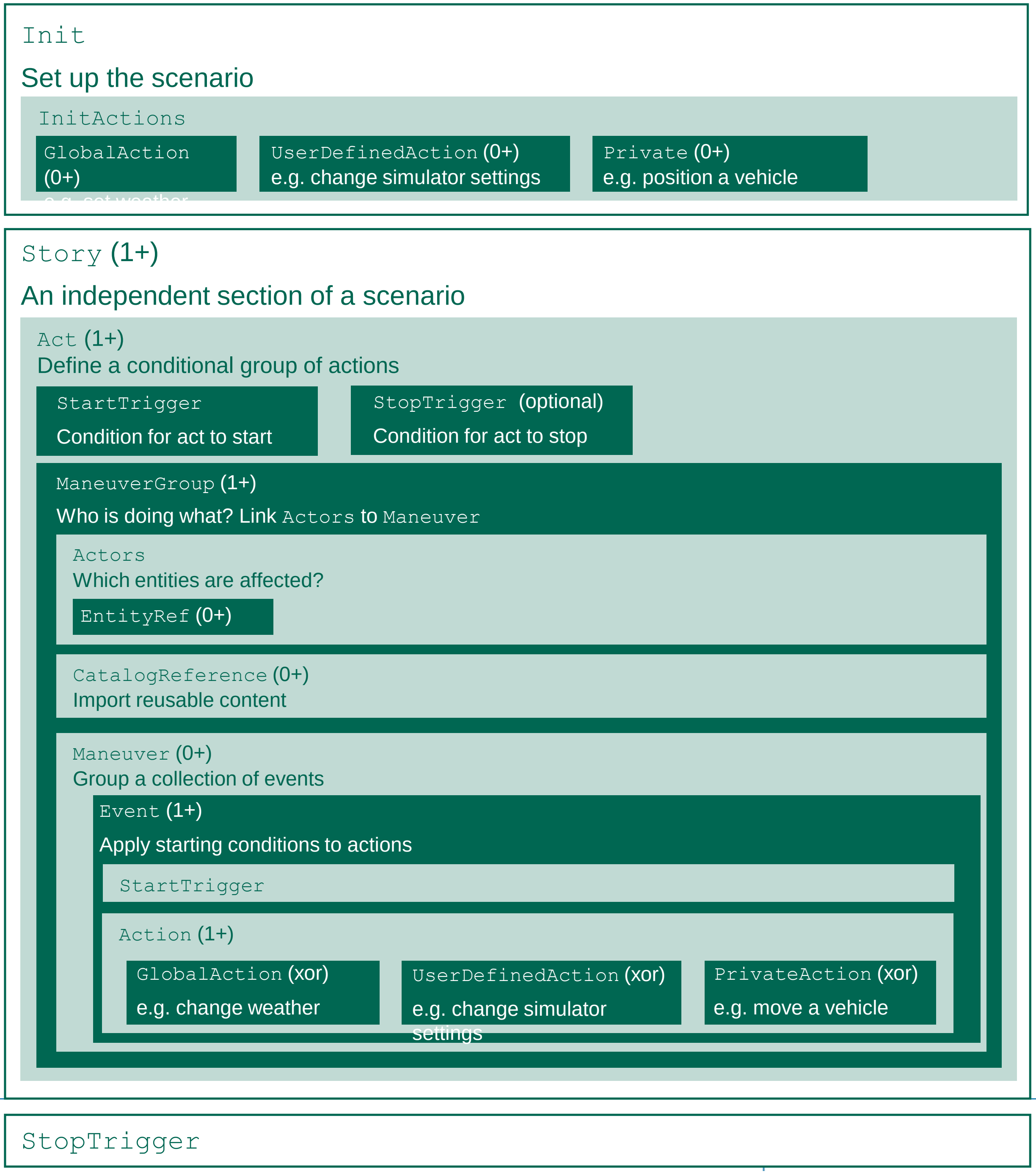
The state of an Entity can be queried to **trigger an Action**.

**EntitySelections** are used to reference groups of Entities.

## Storyboard

The Storyboard is structuring the scenario as a storybook for the dynamic content of the virtual world:

- **Init** is used as initialization phase to set initial conditions of the scenario.
- **Stories** allow grouping aspects into a higher-level hierarchy to provide structure in large scenarios.
- Stories contain **Acts** that define conditional groups of Actions.
- Triggers (i.e. **startTriggers** and **stopTriggers**) control the execution of Acts.
- **ManeuverGroups** are assigning Entities as Actors to Maneuvers.
- **Maneuvers** are containers for Events that share a common scope.
- **Events** control the simulated world or corresponding Entities. This is achieved through triggering **Actions**, given user-defined **Conditions**.


Init
Set up the scenario
InitActions
GlobalAction (0+)
UserDefinedAction (0+) e.g. change simulator settings
Private (0+) e.g. position a vehicle
Story (1+)
An independent section of a scenario
Act (1+)
Define a conditional group of actions
StartTrigger
Condition for act to start
StopTrigger (optional)
Condition for act to stop
ManeuverGroup (1+)
Who is doing what? Link Actors to Maneuver
Actors
Which entities are affected?
EntityRef (0+)
CatalogReference (0+)
Import reusable content
Maneuver (0+)
Group a collection of events
Event (1+)
Apply starting conditions to actions
StartTrigger
Action (1+)
GlobalAction (xor) e.g. change weather
UserDefinedAction (xor) e.g. change simulator settings
PrivateAction (xor) e.g. move a vehicle
StopTrigger

## Actions

Actions serve to **create or modify dynamic elements** of a scenario, e.g. change in lateral dynamics of a vehicle or change of the time of day. Actions are divided in three categories:

- PrivateActions
- GlobalActions
- UserDefinedActions

In the **Init** phase of a scenario, Actions are responsible for **setting up initial states** of dynamic objects, environment, infrastructure, etc.

In any **later phase** of the scenario Actions are executed when Events are **triggered**.

## Global Actions

GlobalActions are used to set or modify non-entity related quantities:

| | |
|---|---|
| • **EnvironmentAction** | Setting Weather, RoadConditions and TimeofDay. |
| • **EntityAction** | Removing or adding instances of Entity. |
| • **ParameterAction** | Setting/modifying values of parameters. |
| • **InfrastructureAction** | Setting/modifying the state of a traffic signal or a traffic signal controller phase. |
| • **TrafficAction** | Populating ambient traffic by defining sources, sinks and swarms. |

## Private Actions

PrivateActions are assigned to Entities to describe motion, position, and visibility:

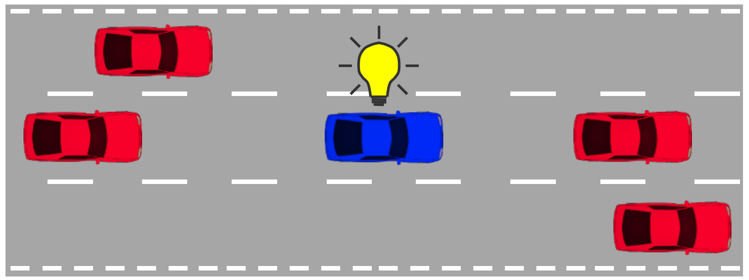
| | |
|---|---|
| • **LongitudinalActions** | Controlling speed or distance to a target (SpeedAction, LongitudinalDistanceAction). |
| • **LateralActions** | Targeting a lateral position within a lane (LaneChangeAction, LaneOffsetAction). |
| • **VisibilityAction** | Enabling/disabling detectability/visibility of Entities. |
| • **SynchronizeAction** | Controlling an Entity to arrive at a reference position and speed. |
| • **ActivateControllerAction** | (De-)activate a Controller model. |
| • **ControllerAction** | Override Controller signals, e.g. apply the brakes. |
| • **TeleportAction** | Defining a teleport destination for an Entity. |
| • **RoutingAction** | Specifying the Route or Trajectory that an Entity should follow (AssignRouteAction, AcquirePositionAction, FollowTrajectoryAction). |

## Controllers

Controllers can be assigned to Entities of type Vehicle or Pedestrian. They are activated for a given domain (i.e. longitudinal or lateral) using the ActivateControllerAction and may be internal (part of the simulator) or external (defined in another file).

Use cases for controllers comprise:

- Specifying that a vehicle should be controlled by the **system under test**.
- Defining ”**smart agent**” behavior, i.e. have the Controller take intelligent decisions in response to the road network and/or other vehicles.
- Assigning a vehicle to direct **human control**.



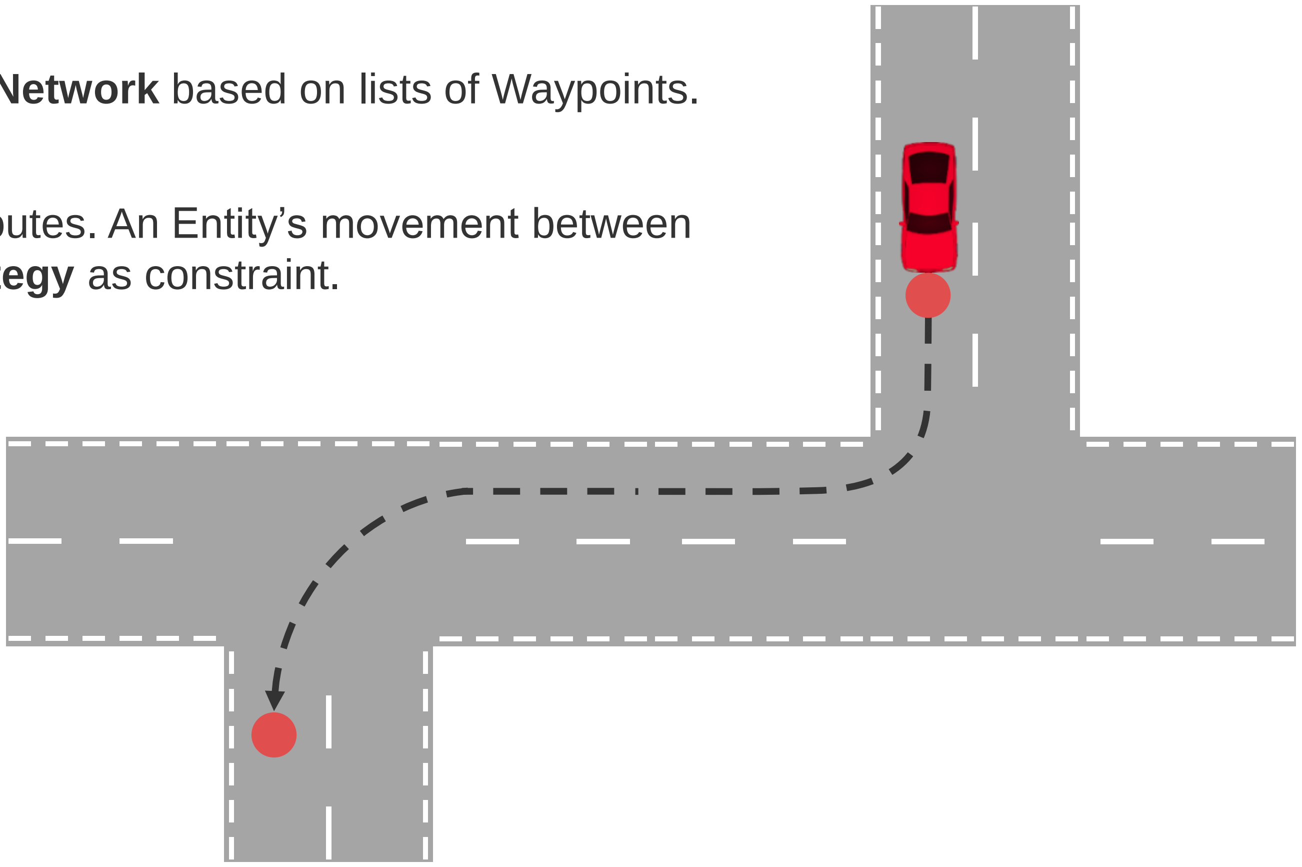
## Routes

**Routes** are used to **navigate** Entities through the **RoadNetwork** based on lists of Waypoints.

**Waypoints** are linked in order, resulting in directional Routes. An Entity's movement between Waypoints is **left to the simulator** using the **RouteStrategy** as constraint.

**Unambiguous Routes** can be specified using a sufficient number of Waypoints, resulting in a one-dimensional coordinate system that enables unambiguous localization and positioning.

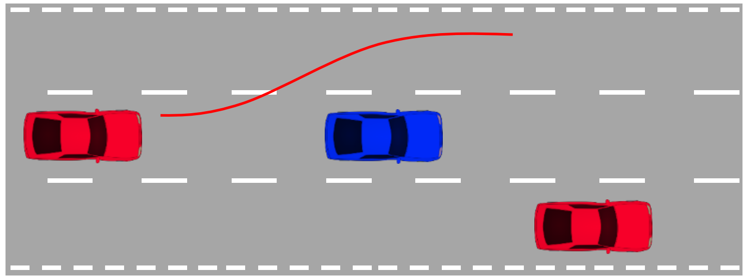
Routes may be assigned to Entities using **AssignRouteAction** or **AcquirePositionAction**. While the former directly assigns a route to the corresponding Entity, the latter is used to create a route "on the fly" by specifying a target destination.



# Trajectories

Trajectories define **mathematical precise paths** for Entity motion. They can be specified using one of the following mathematical shapes:

- **Polyline** (a concatenation of simple line segments across a set of vertices)
- **Clothoid** (Euler spiral, i.e. a curve with linearly increasing curvature)
- **Non-Uniform Rational B-Splines** (NURBS) of arbitrary order



**NURBS** can **express most relevant paths** either directly, or with arbitrary approximation. They directly support the expression of conic sections (such as circles and ellipses).

Trajectories can be specified using the three **positional dimensions**. Three **rotational dimensions** can be added to specify Entity orientation. An optional **time dimension** allows for the specification of an Entity's velocity.

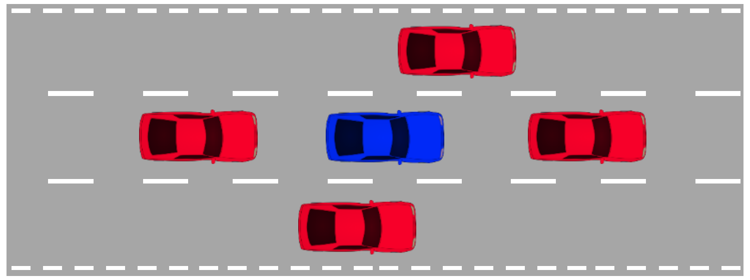
Whilst Trajectories provide mathematically precise paths, Entities can follow these paths either directly or use it as guidance for their controller (as defined in the **FollowTrajectoryAction**).

## Traffic Simulation

Besides the definition of deterministic Entity behavior, OpenSCENARIO provides ways to define **stochastic or not precisely defined behavior** for traffic simulation. This can be used to create **macroscopic traffic** within a scenario or **around Entities**.

For this purpose, traffic agents can be defined using **TrafficActions**. With the help of these actions, the following aspects of traffic simulation can be specified:

- Parameterization of traffic sources (**TrafficSinkAction**)
- Parameterization of traffic sinks (**TrafficSinkAction**)
- Parametrization of traffic swarms (**TrafficSwarmAction**)



TrafficActions do not specify which maneuvers will be executed by the traffic agents.
This task is up to the implementation of the simulator.

# ASAM OpenSCENARIO 1.0.0

Scenario Creation

## Scenario Creation

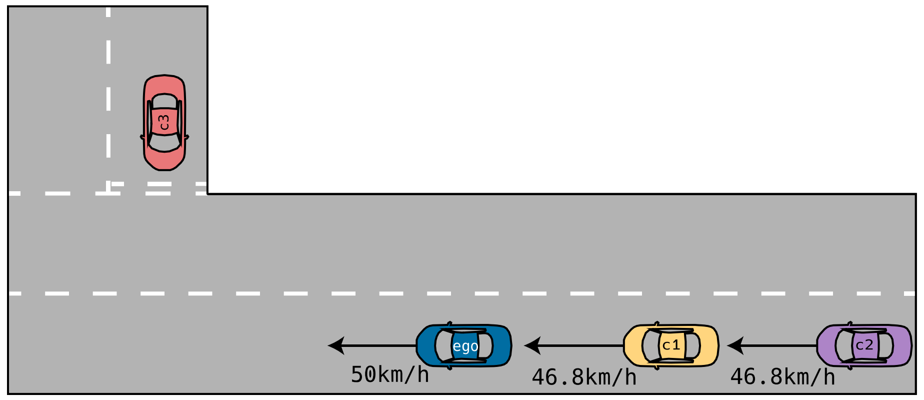
## Example Scenario

The Ego vehicle, an externally controlled vehicle, is driving along an urban road approaching a junction on the offside.

Ego is being followed by two vehicles, C1 and C2. A third vehicle (C3) is waiting to turn right at the junction.

As Ego approaches the junction, C1 and C2 start to overtake. Slightly later, C3 starts to turn right, which prompts C1 and C2 to make an emergency stop.

Initial positions of the vehicles:



## Init

The XML example to the right shows an Action which positions Ego using global coordinates at the Init phase. In this phase of the scenario, Actions do not require Triggers to be executed.

Similar Actions (not shown) are used to specify speeds and positions for the other vehicles.

```
<Storyboard>
    <Init>
        <Actions>
            <Private entityRef = "Ego">
                <PrivateAction>
                    <!-- Set Ego to its initial position -->
                    <TeleportAction>
                        <Position>
                            <WorldPosition x = "-2.51"
                      y = "-115.75"
                      z = "0"
                      h = "1.57"
                      p = "0"
                      r = "0" />
                        </Position>
                    </TeleportAction>
                </PrivateAction>
                ...
                <!-- Similar actions -->
            </Private>
        </Actions>
   </Init>
     ...
</Storyboard>
```

## Story

Stories are used to **group Acts**. While it’s never required to use more than one Story, here, two Stories are used:

- **AbortedOvertake** describes the overtakes and emergency stops
- **RightTurn** describes the right turn

AbortedOvertake contains two Acts

- **AbortedOvertakeAct1** controls the overtaking behavior
- **AbortedOvertakeAct2** control the emergency stops

RightTurn contains only a single Act, **RightTurnAct**.

```
<Story name      = "AbortedOvertake">
    <Act name    = "AbortedOvertakeAct1">
        ...
        <!-- Act content describing overtakes -->
    </Act>
    <Act name    = "AbortedOvertakeAct2">
        ...
        <!-- Act content describing emergency stops -->
    </Act>
</Story>
<Story name = "RightTurn">
    <Act name    = "RightTurnAct">
        ...
        <!-- Act content describing right turn -->
    </Act>
</Story>
```

# Acts

Acts (which contain ManeuverGroups), allow application of **Triggers** to parts of the scenario.

Triggers may be present at Act and Event level:

- At Act level, they are used to start the overtake
- At Event level, they control its execution

In this example, both C1 and C2 should start to overtake at the same time. This makes it convenient to put all content associated with both overtakes in the same Act (AbortedOvertakeAct1).

The example to the right shows the structure of the RightTurnAct which triggers when Ego is close to the junction.

```
<Act name = "RightTurnAct">
    <!-- Maneuver Group -->
    ...
    <StartTrigger>
        <ConditionGroup>
            <Condition
                name     = "EgoCloseToJunction"
                delay    = "0"
                conditionEdge    = "rising">
                <!-- ByEntity condition: Ego close to junction -->
                ...
            </Condition>
        </ConditionGroup>
    </StartTrigger>
</Act>
```

## ManeuverGroups

In AbortedOvertakeAct1, **two vehicles** perform the **same Actions**. However, not all of these Actions should happen at the same time.

C1 and C2 return to their original lane when they have passed the Ego vehicle, **independent of what the other one is doing**.

This behavior can be achieved by using **separate ManeuverGroups** for each vehicle (C1ManeuverGroup and C2ManeuverGroup).

Each ManeuverGroup allocates a Maneuver (from a Catalog) to one vehicle. This Maneuver instructs that vehicle to change lane, accelerate, and then return to the previous lane ahead of the Ego vehicle.

```
<ManeuverGroup name                  = "C1ManeuverGroup"
               maximumExecutionCount = "1">
    <Actors    selectTriggeringEntities = "false">
        <EntityRef entityRef    = "C1"/>
    </Actors>
    <CatalogReference catalogName   = "overtake"
                        entryName     = "Overtake Ego vehicle">
        <!—Parameter assignment -->
        ...
    </CatalogReference>
</ManeuverGroup>

<ManeuverGroup name                  = "C2ManeuverGroup"
               numberOfExecutions    = "1">
    ...
            <!-- similar to above -->
</ManeuverGroup>
```

## Maneuvers (1/2)

Analogous to Stories, it’s never essential to use more than one Maneuver.

If an Event is moved from one Maneuver to another (within the same ManeuverGroup) the scenario will work in the same way.

In AbortedOvertakeAct1, vehicles C1 and C2 need to perform an **overtake in the same way**, but it must be specified in **two different ManeuverGroup** elements.

Therefore, a **Catalog Maneuver** is defined:

```
<Catalog name = "Overtake">
    <Maneuver name = "Overtake Ego Vehicle">
        <ParameterDeclarations>
            <ParameterDeclaration name = " $OvertakingVehicle"

                               parameterType = " string"
                                      value = ""/>
            <!-- "" will be overwritten by scenario -->
        </ParameterDeclarations>
        <!-- Events to define overtake behaviour -->
        <Event ...> ... </Event>
        ...
    </Maneuver>
</Catalog>
```

## Maneuvers (2/2)

The before-mentioned maneuver is referenced within both ManeuverGroups. Thus, the Catalog reference itself does not define which Vehicle executes the Actions.

However, it does contain a **Condition** to check **when the overtaking vehicle can return to its lane**.

This requires the names of the two vehicles involved to be specified. To achieve this, a **Parameter** with the name of the vehicle overtaking is included in the Catalog reference.

```
<ManeuverGroup  name     = "C1ManeuverGroup"
                maximumExecutionCount   = "1">
    <Actors  selectTriggeringEntities    = "false">
        <EntityRef  entityRef    =   "C1"/>
    </Actors>
    <CatalogReference   catalogName     = "Overtake"
                        entryName    = "OvertakeEgoVehicle">
        <ParameterAssignments>
            <ParameterAssignment parameterRef   = "OvertakingVehicle"
                                                value = "C1"/>
        </ParameterAssignments>
    </CatalogReference>
</ManeuverGroup>

<ManeuverGroup  name     = "C2ManeuverGroup"
                maximumExecutionCount   = "1">
    <Actors     selectTriggeringEntities    = "false">
        <EntityRef  entityRef    = "C2"/>
    </Actors>
    <CatalogReference   catalogName = "Overtake"
                        entryName    = "OvertakeEgoVehicle">
        <ParameterAssignments>
            <ParameterAssignment parameterRef   = "OvertakingVehicle"
                                    value = "C2"/>
        </ParameterAssignments>
    </CatalogReference>
</ManeuverGroup>
```

## Events

The lane change Action should start straight away when its parent Act is triggered. Events are required to **apply Triggers to Actions**. In this example, a trivial **Condition** is used to trigger immediate execution.

For other Events, Conditions are used to ensure a certain state is reached before the Action is applied (for example, the acceleration Event must not start until the vehicle has changed lane).

```
<Event  name     = "brake event"
    priority     = "overwrite">
    ...
    <!-- Emergency stop action -->
    <StartTrigger>
        <ConditionGroup>
            <Condition  name = "StartConditionOfAborted
OvertakeAct2"
                        delay = "0"
                        conditionEdge = "none">
                <ByValueCondition>
                    <SimulationTimeCondition value = "0
"
                                             rule  = "g
reaterThan"/>
                </ByValueCondition>
            </Condition>
        </ConditionGroup>
    </StartTrigger>
</Event>
```

Dr.-Ing. Ludwig Friedmann

BMW AG

Phone: +49 151 601 22102

Email: ludwig.friedmann@bmw.de