CommonRoad: Documentation of the XML Format

(Version 2018a)

Markus Koschi, Stefanie Manzinger, and Matthias Althoff

Technische Universität München, 85748 Garching, Germany

Abstract

This document presents the XML format in *CommonRoad* for specifying road traffic scenarios. The provided scenarios in *CommonRoad* are described in the scenario documentation. The *CommonRoad* XML files are composed of (1) a formal representation of the road network, (2) static and dynamic obstacles, and (3) the planning problem for the ego vehicle(s). So far, we have not discovered any limitations when building scenarios using the proposed format.

Contents

1 Introduction

Within *CommonRoad* [1][1], XML files are used to store the data for specific driving scenarios. In this documentation, we present the definition of the XML files, which are composed of (1) a formal representation of the road network (see Section 2.3), (2) static and dynamic obstacles

[1] commonroad.in.tum.de

(see Section 2.4), and (3) the planning problem of the ego vehicle(s) (see Section 2.5). In non-collaborative scenarios, only one planning problem exists, while in collaborating scenarios several planning problems have to be solved.

Throughout this documentation, we provide some snippets to illustrate the implementation in XML. For more examples, see the example file `XML_commonRoad_minimalExample` or any benchmark scenario on our website.

A *CommonRoad* XML file consists of an XML header (XML version and encoding type) and the *CommonRoad* root element.

```
<?xml version="1.0" encoding="utf-8"?>
<commonRoad commonRoadVersion='2018a' benchmarkID='documentation_example'
date='2018-06-20' author="Markus Koschi" affiliation="Technical University of
Munich, Germany" source="Next Generation Simulation (NGSIM) and OpenStreetMaps
(OSM)" tags="..." timeStepSize='0.1'>
    <lanelet id='10'>
        ...
    </lanelet>
     ...
    <obstacle id='50'>
        ...
    </obstacle>
    ...
    <planningProblem id='100'>
        ...
    </planningProblem>
    ...
</commonRoad>
```

1.1 Changes Compared to Version 2017a

For a quick reference, we summarize the major changes of Version 2018a compared to Version 2017a:

- The scenario IDs have a new, consistent format (see Sec. 2.1.1).
- New attributes of the CommonRoad root element: author, affiliation, source, tags (see Sec. 2.1).
- The future behavior of obstacles must start at $t = 0$ and the time of the initial state must be 0.
- Instead of the intermediate *goalRegion* element, the *goalStates* are directly listed below the *planningProblem* (see Fig. 1).
- In a goal state, only the time element is mandatory so that a planning problem can simply be to remain collision-free for the planning horizon (see Fig. 9).

2 Specification of the XML Format

Fig. 1 specifies the overall structure of the XML files. The *CommonRoad* root element contains one or more elements of type *lanelet*, *obstacle*, and *planningProblem*. Each has a unique[2] ID making it possible to reference it. The numbers in square brackets denote the number of allowed elements (while `N` can be different for each element), the data in round brackets the attributes of an element, `ref to` a reference to one element, the data behind a double column the value of the element.

```
[1] commonRoad (commonRoadVersion, ...)
├─ [1..N] lanelet (id)
│  ├─ [1] leftBound
│  │  ├─ [2..N] point
│  │  └─ [0..1] lineMarking
│  ├─ [1] rightBound
│  │  ├─ [2..N] point
│  │  └─ [0..1] lineMarking
│  ├─ [0..N] predecessor (ref to lanelet)
│  ├─ [0..N] successor (ref to lanelet)
│  ├─ [0..1] adjacentLeft (ref to lanelet, drivingDir=same/opposite)
│  ├─ [0..1] adjacentRight (ref to lanelet, drivingDir=same/opposite)
│  └─ [0..1] speedLimit
├─ [0..N] obstacle (id)
│  ├─ [1] role:  static
│  ├─ [1] type:  parkedVehicle/.../unknown
│  └─ [1] shape
├─ [0..N] obstacle (id)
│  ├─ [1] role:  dynamic
│  ├─ [1] type:  car/truck/.../unknown
│  ├─ [1] shape
│  └─ [1] trajectory
│     └─ [1..N] state
│  OR
├─ [0..N] obstacle (id)
│  ├─ [1] role:  dynamic
│  ├─ [1] type:  car/truck/.../unknown
│  └─ [1] occupancySet
│     └─ [1..N] occupancy
│  OR
├─ [0..N] obstacle (id)
│  ├─ [1] role:  dynamic
│  ├─ [1] type:  car/truck/.../unknown
│  ├─ [1] shape
│  └─ [1] probabilityDistribution
└─ [1..N] planningProblem (id)
   ├─ [1] initialState
   └─ [1..N] goalState
```

Figure 1: Structure of the XML files encoding each scenario

[2]Unique within the whole XML document.

To check the compliance of an XML file, we provide the XML Schema Definition (XSD) of our format on our website.

All variables are given by decimal numbers based on SI units. We use a common Cartesian coordinate frame, where angles are measured counter-clockwise around the positive z-axis with the zero angle along the x-axis.

2.1 CommonRoad Root Element

The *CommonRoad* root element has the following attributes (its elements are shown in Fig. 1):

- `commonRoadVersion`: version of the XML specification,
- `benchmarkID`: benchmark ID of the scenario (see Sec. 2.1.1),
- `date`: date when scenario was generated,
- `author`: author(s) of the scenario in alphabetic order,
- `affiliation`: affiliation of the author(s), e.g., name and country of the university or company,
- `source`: if applicable, description of the data source of the scenario, e.g., name of dataset or map service,
- `tags`: keywords describing the type of scenario (see Sec. 2.1.2),
- `timeStepSize`: global step size of the time-discrete scenario.

2.1.1 Benchmark ID

The benchmark ID of each scenarios consists of four elements: COUNTRY_SCENE_CONFIG_PRED. The scenario ID has the prefix C- if the scenario has multiple planning problems, i.e. it is a cooperative planning problem (otherwise, it has no prefix).

COUNTRY is the capitalized three-letter country code defined by the ISO 3166-1 standard[3], e.g. Germany has DEU and United States has USA. If a scenario is based on an artificial road network, we use ZAM for Zamunda[4].

SCENE = MAP-{1-9}* specifies the road network. MAP is for rural scenarios a two/three letter city code (e.g. Muc) and for highways/major roads the road code (e.g. A9 or Lanker). It is appended by an integer counting up. Note that if COUNTRY_SCENE is the same for two scenarios, all their lanelets are identical.

CONFIG = {1-9}* specifies the initial configuration of obstacles and the planning problem(s). Note that CONFIG is counting independently for non-cooperative scenarios (i.e. only one planning problem) and cooperative scenarios (i.e. multiple planning problems), since the prefix allows to distinguish between them. Thus, if PREFIX-COUNTRY_SCENE_CONFIG is the same for two scenarios, the road network, initial configuration of obstacles, and the planning problem(s) are equal, and only the prediction of the obstacles differs.

[3] `https://www.iso.org/obp/ui/#search/code/`
[4] en.wikipedia.org/?title=Zamunda

PRED = {S,T,P}-{1-9}* specifies the future behavior of the obstacles, i.e. their prediction, where S = set-based occupancies, T = single trajectories, P = probability distributions, appended by an integer to distinguish predictions on the same initial configuration but with different prediction parameters. If no prediction is used (i.e. the scenario has no dynamic obstacles), we omit the element PRED in the benchmark ID.

Examples C-USA_US101-1_123_T-1, DEU_FFB-2_42_S-4, DEU_Hhr-1_1.

2.1.2 Tags for Scenarios

To allow users to select scenarios meeting their needs, the list of scenarios on our website can be filtered by the tags given in the attribute `tags` in each XML file. These are *type of road* (a one-lane road has only one lane, a two-lane road has one lane per driving direction, and a multi-lane road has multiple lanes per driving direction) and *type of required planning maneuver* among others (see website for all options). The tags are strings in a space-separated list.

2.2 Auxiliary Elements

Within the XML file, we use the following auxiliary geometry elements:

```
/
├── point OR center
│   ├── [1] x
│   └── [1] y
├── rectangle
│   ├── [1] length
│   ├── [1] width
│   ├── [0..1] orientation
│   └── [0..1] center
├── circle
│   ├── [1] radius
│   └── [0..1] center
├── polygon
│   └── [3..N] point
└── shape
    └── [1..N] rectangle/circle/polygon
```

Figure 2: Auxiliary elements of the XML file

Point A point is the simplest primitives and described by an x-coordinate and a y-coordinate.

Rectangle The element *rectangle* can be used to model rectangular obstacles, e.g. a vehicle. It is specified by the length (longitudinal direction) and the width (lateral direction), the orientation, and a center point (reference point of a rectangle is its geometric center). If the orientation and the x and y coordinates of the center are zero, both elements can be omitted.

Circle The element *circle* can be used to model circular obstacles, for example a pedestrian or a vehicle by using three circles. A circle is defined by its radius and its center (reference point of a circle is its geometric center). Analogously to the rectangle, the center can be omitted if its x and y values are zero.

Polygon The element *polygon* can be used to model any other two-dimensional obstacle. A polygon is defined by an ordered list of points, in which the first one is its reference point. We adhere to the convention that the polygon points are ordered clockwise.

Shape Elements of type *shape* specify the dimension of an object and can contain one or more elements of the geometric primitives (i.e. rectangle, circle, or polygon). Please note that we separate the representation of the dimension and position/orientation of an object into the elements shape and position/orientation (described subsequently), respectively. Thus, the shape elements should usually use the origin as center point and an orientation of zero, unless a certain offset is desired.

Positions The position of an object is specified by the element *position* which contains either a point, rectangle, circle, polygon, or lanelet.

Note that if an area is specified as position, it does not enclose the geometric shape of an object, but only models the interval of possible positions, e.g. the uncertainty of the position measurement.

Numeric Values Elements like orientation, time, velocity, acceleration, yaw rate, or slip angle can have either an exact value or an interval of values, e.g. to specify the goal state or to include uncertainties. For example, a *time* element can be defined using `exact` or `intervalStart` and `intervalEnd`:

```
<time>
        <exact>0.0</exact>
        <!-- or -->
        <intervalStart>0.0</intervalStart>
        <intervalEnd>1.0</intervalEnd>
</time>
```

2.3 Lanelets

For our benchmarks we use *lanelets* [2] as drivable road segments to represent the road network. Fig. 1 shows the specification of a *lanelet* element. It is defined by its *left* and *right boundary*, where each boundary is represented by an array of points (a polyline), as shown in Fig. 3. We have chosen lanelets since they are as expressible as other formats, such as e.g. OpenDRIVE[5], yet have a lightweight and extensible representation. Our converter from OpenDRIVE to Lanelets is available on our website.

In order to represent the graph of the road network, the elements *predecessor*, *successor*, *adjacentLeft*, and *adjacentRight* are used, which are omitted if they are empty (see Fig. 1). Since these elements only contain objects which are already present in the XML file, we refrain from copying their data but introduce references to the neighboring lanelets by an attribute referring

[5] opendrive.org

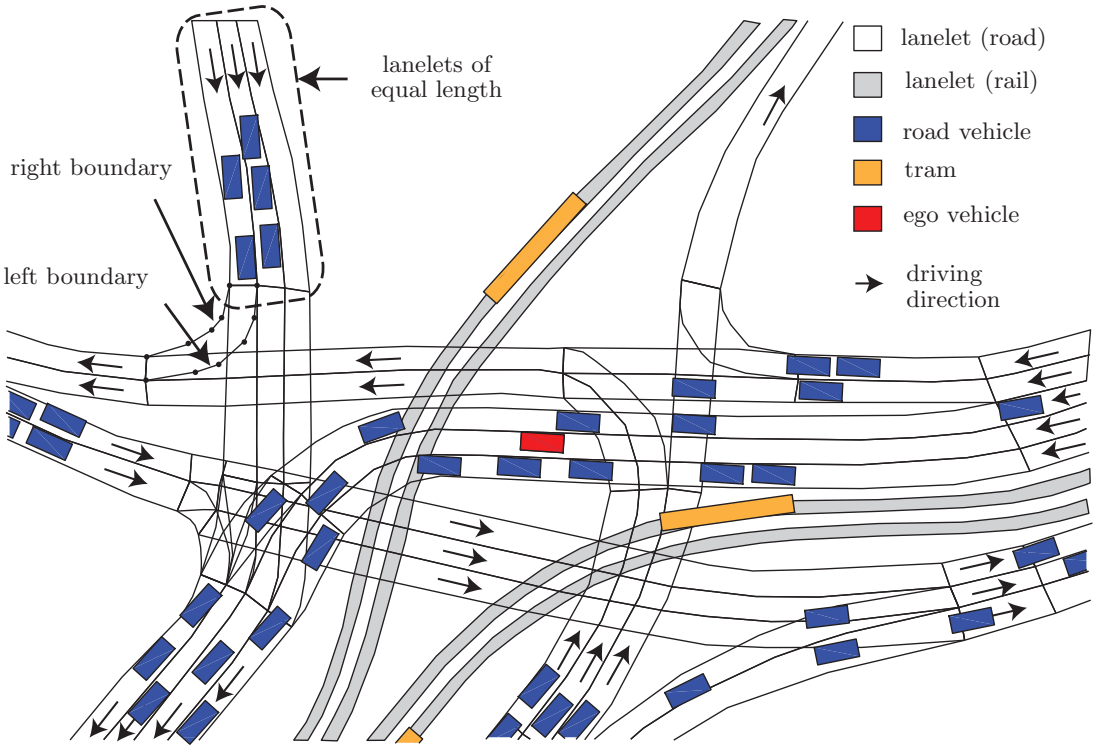

Figure 3: Lanelets of a complex intersection in the city center of Munich. Besides roads, also tram rails are modeled by lanelets.

to their unique ID. The elements *predecessor* and *successor* can be present multiple times to represent multiple longitudinally adjacent lanelets, e.g. for a road fork. In contrast, a lanelet can have at the most one *adjacentLeft* and one *adjacentRight* neighbor and thus at the most one element of this type. The additional attribute *drivingDir* specifies the driving direction of the neighboring lanelet as `same` or `opposite`.

The driving direction of a lanelet is implicitly defined by its left and right bound. Optionally, line markings (solid, dashed, ...) or the speed limit can be included to model the traffic conditions more precisely (see Fig. 1). Further traffic signs will be added to the *CommonRoad* XML format in a later release to represent traffic conditions more precisely.

2.3.1 Geometrical Requirements of Lanelets

All *CommonRoad* scenarios meet the following requirements, which assure that lanelets form a road without holes or incorrect overlaps.

- The two polylines forming the right and left bound of a lanelet must consist of the same amount of nodes. In addition, the imaginary straight line connection between two corresponding nodes, one in the left and one in the right bound, should be perpendicular to the center line of the lanelet.
- In case of a two-lane or multi-lane road, a polyline can be shared by two lanelets, i.e. the same points are used to mark the right respectively left boundary of the corresponding lanelets.
- For longitudinal adjacent lanelets, the connection nodes of two consecutive lanelets have to be identical, i.e. the end nodes of the predecessor are identical to the start nodes of the successor.

- To ensures continuous lanes, the bounds of merging and forking lanelets start/end at the corresponding left or right bound of another lanelet, as shown in Fig. 4.

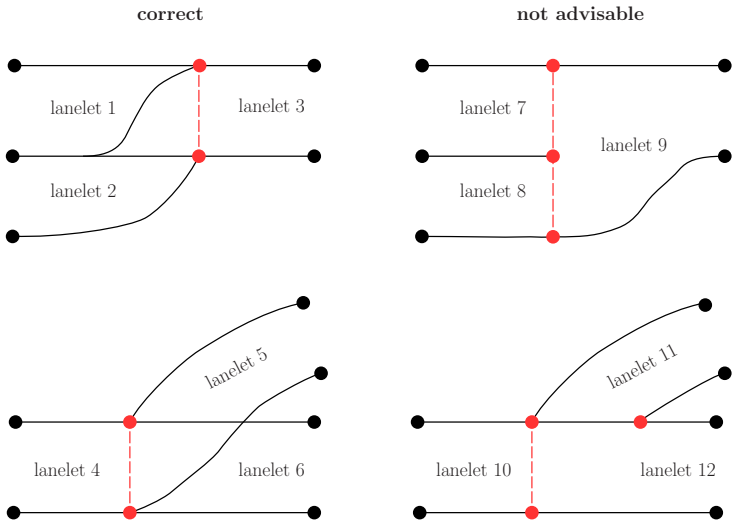

Figure 4: Spatial division of merging and forking lanelets.

- Roads are divided in so called *Lane Sections*[6]. As shown in Fig. 5, each lane section has the same number of lateral adjacent lanelets and all lanelets start and end at the border of a lane section. Thus, all laterally adjacent lanes have the same *length*, which allows us to set the lateral adjacencies correctly (e.g. in Fig. 5, lanelet 1 and 2 are lateral adjacent to each other; as well as lanelet 4, 5, and 6; and lanelet 7 and 8).

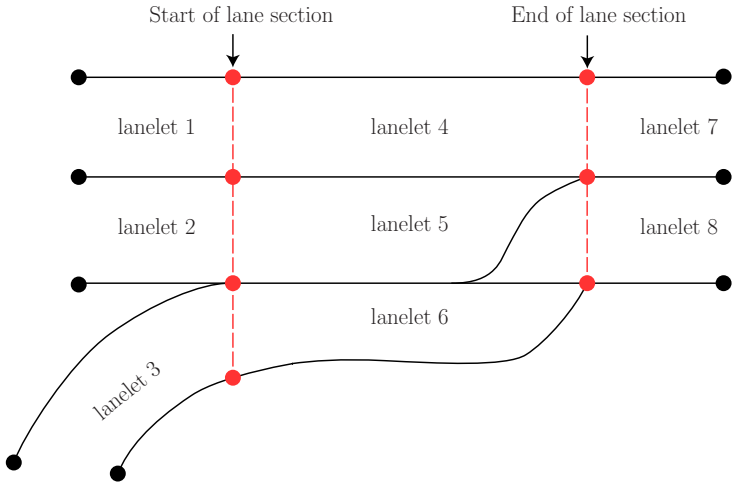

Figure 5: Definition of lane sections.

2.3.2 Connection to OpenStreetMap

As proposed in [2][7], our XML structure is inspired by the data structure of OpenSteetMap[8] (OSM), which uses the elements *nodes*, *ways* and *relations*.

[6] opendrive.org/
[7] github.com/phbender/liblanelet
[8] openstreetmap.org

With the XML format of OpenStreetMap, it easy to create new road networks using *JavaOpenStreetMap* (JOSM)[9] and also to edit existing road networks. However, the OSM format has several shortcomings when representing traffic scenarios, e.g. non-Cartesian coordinate frame and inconsistent use of attributes/references instead of elements. Thus, we have developed the *CommonRoad* XML format, which uses a different XML notation and incorporates only a small subset of the OSM attributes, but offers all elements required to specify a traffic scenario.

2.4 Obstacles

The element *obstacle* is used to represent different kinds of traffic participants within the scenario. An obstacle is either static or dynamic, which is specified by the element *role*. Each role allows different types of an obstacle as listed in Table 1.

Table 1: Types of obstacles.

Role	**Type**
Static	parkedVehicle, constructionZone, unknown
Dynamic	car, truck, bus, bicycle, pedestrian, priorityVehicle, train, unknown

2.4.1 Static Obstacles

A static obstacle is specified by the elements *role*, *type*, and *shape*, as shown in Fig. 1.

In addition to static obstacles, traffic scenarios can contain dynamic obstacles. Please note that only elements of either of the following three behavior models (with known behavior, with unknown behavior, or with unknown stochastic behavior) may be present. We do not use these different behavior models together within one traffic scenario, as indicated in Fig. 1.

2.4.2 Dynamic Obstacles with Known Behavior

A dynamic obstacle with known behavior contains the same elements as a static obstacle and additionally a trajectory of states. The trajectory allows us to represent the states of a dynamic traffic participant along a path. The trajectory is obtained from a dataset (whose measurements can be exact or with uncertainties), from a prediction (which generates a single trajectory for each obstacle), or created hand-crafted.

States The time-discrete states of a trajectory are specified by the element *state* with the following state variables: *position*, *orientation*, and *time*, *velocity* (scalar), *acceleration* (scalar), *yawRate*, and *slipAngle*, as shown in Fig. 6. The first state of a trajectory must start at $t = 0$. Note that we optionally include acceleration as a state variable for obstacles to provide additional information, e.g. for motion prediction, even though acceleration is often used as input for vehicle models.

[9] josm.openstreetmap.de

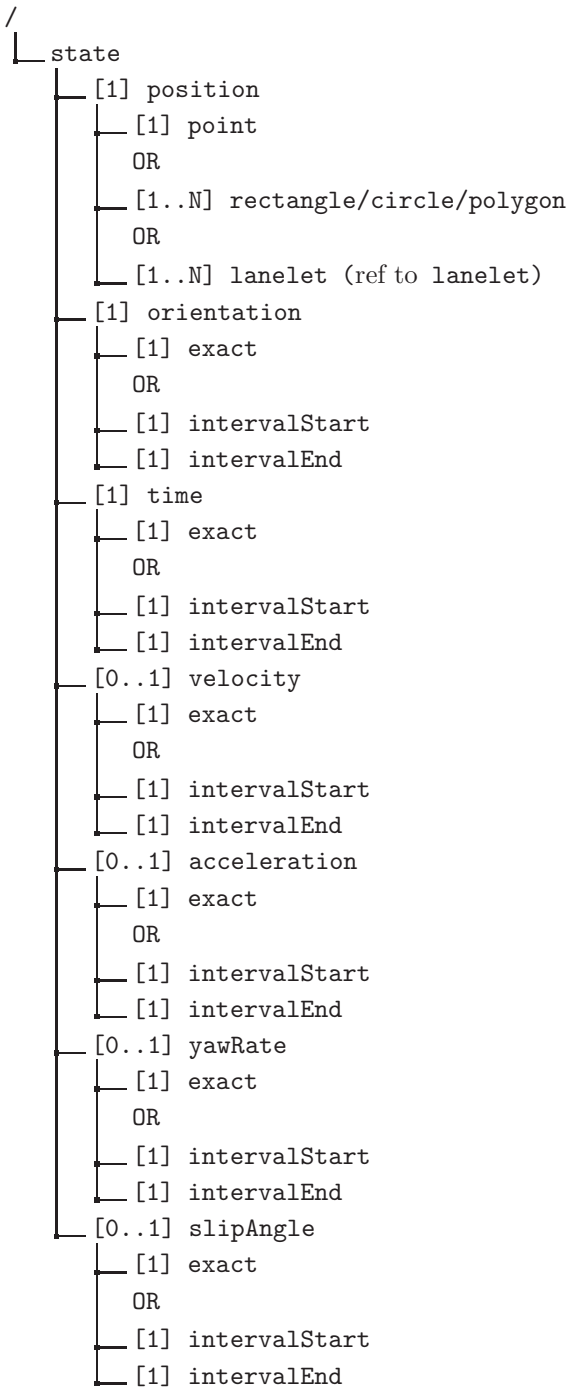

Figure 6: Element *state* of a trajectory.

2.4.3 Dynamic Obstacles with Unknown Behavior

For motion planning, we often do not know the exact future behavior of dynamic obstacles, but we instead represent their future behavior by bounded sets. Thus, dynamic obstacles with a unknown behavior are specified by the elements *role*, *type*, and *occupancy set*, where the latter represents the occupied area over time by bounded sets.

Occupancies An *occupancy set* contains a list of *occupancy* elements, which consists of a shape (occupied area) and a time, as shown in Fig. 7.

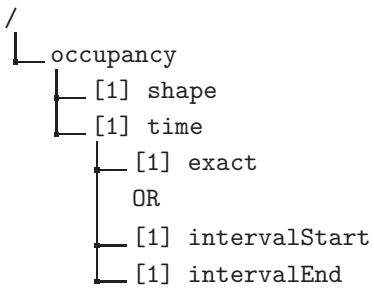

Figure 7: Element *occupancy* of an occupancy set.

2.4.4 Dynamic Obstacles with Unknown Stochastic Behavior

One can describe unknown stochastic behavior by probability distributions of states. Since many different probability distributions are used, we only provide a placeholder for probability distributions.

2.5 Planning Problem

The element *planningProblem* is used to specify the initial state and one or more goal state(s) for the motion planning problem. Note that the shape of the ego vehicle is not included in the XML scenario description, since this property depends on which vehicle parameter set is chosen (see the *vehicle model documentation* on our website).

Initial States We use the element *initial state* to describe the initial state of the planning problem. In contrast to the general element *state*, all state variables are mandatory and must be given exact, as shown in Fig. 8. The element *initial state* of each planning problem allows the initialization of each vehicle model, as described in more detail in our *vehicle model documentation*.

Goal States A planning problem may contain several elements *goal state* (cf. Fig. 1). In contrast to the general element *state*, all state variables except time are optional and all variables can only be given as an interval, as specified in Fig. 9.

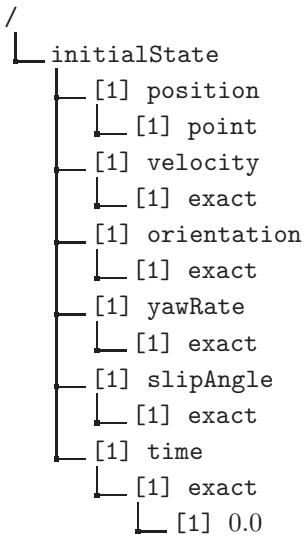

Figure 8: Element *initial state* of a planning problem

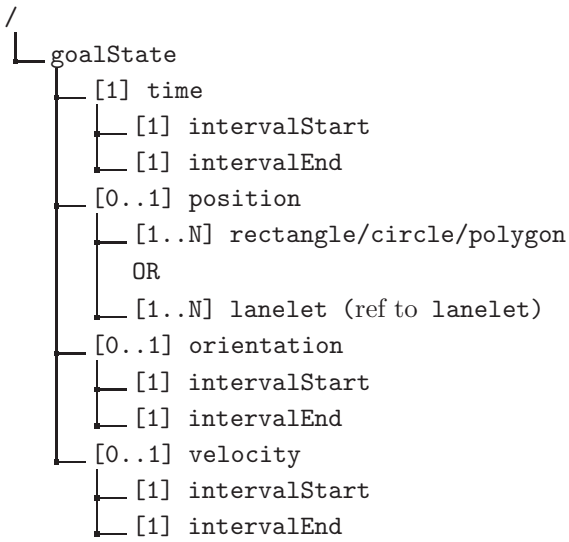

Figure 9: Element *goal state* of a planning problem

3 Conclusions

The *CommonRoad* XML format is a platform-independent format for specifying road traffic scenarios for motion planning. Complex traffic situations can be encoded by specifying the road network, static and dynamic obstacles, and the planning problem. Details on models for the ego vehicle dynamics can be found in the *vehicle model documentation*. Examples of traffic situations that are specified by this format can be found on the *CommonRoad* website[10]. Please contact us if you have any comments.

Acknowledgment

The author gratefully acknowledge financial support by the BMW Group within the Car@TUM project and by the Free State of Bavaria.

References

[1] M. Althoff, M. Koschi, and S. Manzinger. CommonRoad: Composable benchmarks for motion planning on roads. In *Proc. of the IEEE Intelligent Vehicles Symposium*, pages 719–726, 2017.

[2] P. Bender, J. Ziegler, and C. Stiller. Lanelets: Efficient map representation for autonomous driving. In *Proc. of the IEEE Intelligent Vehicles Symposium*, pages 420–425, 2014.

[10] commonroad.in.tum.de