

CommonRoad: Documentation of the XML Format

(Version 2020a)

Markus Koschi, Sebastian Maierhofer, Anna-Katharina Rettinger, 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 *CommonRoad* XML files are composed of (1) meta information about the scenario, (2) a formal representation of the road network, (3) static and dynamic obstacles, and (4) 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	2
1.1	Changes Compared to Version 2018b	3
2	Specification of the XML Format	3
2.1	Meta Information Describing CommonRoad Scenarios	5
2.1.1	CommonRoad Root Element	5
2.1.2	Benchmark ID	5
2.1.3	Location of Scenarios	6
2.1.4	Tags for Scenarios	7
2.2	Auxiliary Elements	7
2.3	Lanelets	9
2.3.1	Geometrical Requirements of Lanelets	10
2.3.2	Connection to OpenStreetMap	11
2.4	Traffic Signs	12
2.5	Traffic Lights	12
2.6	Intersections	14
2.7	Obstacles	14
2.7.1	Static Obstacles	15
2.7.2	Dynamic Obstacles with Known Behavior	16
2.7.3	Dynamic Obstacles with Unknown Behavior	17
2.7.4	Dynamic Obstacles with Unknown Stochastic Behavior	17
2.8	Planning Problem	17
3	Conclusions	18

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) meta information describing the scenario (see Section 2.1), (2) a formal representation of the road network (see Section 2.3-2.6), (3) static and dynamic obstacles (see Section 2.7), and (4) the planning problem of the ego vehicle(s) (see Section 2.8). 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 visualization of all scenarios is on our website², where you can also search for specific types of scenarios.

As a brief introduction, the following snippet shows that a *CommonRoad* XML file consists of an XML header (XML version and encoding type) and the *CommonRoad* root element with location, scenario tag, lanelet, traffic sign, traffic light, intersection element, static and dynamic obstacle, and planning problem:

```
<?xml version="1.0" encoding="utf-8"?>
<commonRoad commonRoadVersion='2020a' benchmarkID='documentation_example'
date='2020-02-29' author="Markus Koschi" affiliation="Technical University of
Munich, Germany" source="Next Generation Simulation (NGSIM) and OpenStreetMaps
(OSM)" timeStepSize='0.1'>
  <location>
    ...
  </location>
  <scenarioTags>
    ...
  </scenarioTags>
  <lanelet id='10'>
    ...
  </lanelet>
  ...
  <trafficSign id='20'>
    ...
  </trafficSign>
  ...
  <trafficLight id='30'>
    ...
  </trafficLight>
  ...
  <intersection id='30'>
    ...
  </intersection>
  ...
  <staticObstacle id='50'>
    ...
  </staticObstacle>
```

¹commonroad.in.tum.de

²commonroad.in.tum.de/scenarios

```
...
<dynamicObstacle id='60'>
  ...
</dynamicObstacle>
...
<planningProblem id='100'>
  ...
</planningProblem>
...
</commonRoad>
```

1.1 Changes Compared to Version 2018b

For a quick reference, we summarize the major changes of Version 2020a compared to Version 2018b:

- Adding of location information.
- Moving of tags from the root element's attribute to a child tag of the root element.
- Lanelets have a lanelet type, e.g. urban or intersection.
- New element traffic signs.
- New element traffic lights.
- New element intersection.
- Separating of *obstacle* in *dynamicObstacle* and *staticObstacle* and removing of the element *obstacleRole*.
- Coordinates in 3-dimensional space.

2 Specification of the XML Format

We designed our format such that it can represent all features, is unambiguous and is easy to use. We put additional emphasis to augment the road description by its implications, i.e., our format does not just describe a traffic situation, but already provides the meaning for motion planning. As a result, our scenarios can directly be used by a motion planner and do not require much additional computations in terms of pre-processing.

The *CommonRoad* XML format is specified by the XML Schema Definition (XSD) available on our website. This XSD file can also be used to check the compliance of an XML file. In the following, we describe our specification.

All variables are given by decimal numbers based on SI units. We use a common Cartesian coordinate frame with x-, y-, and z-axis to be able to represent all road networks including bridges and tunnels. If a scenario is only defined in a two-dimensional plane (which is often the case), we use the convention that all z-coordinates are zero. Angles are measured counter-clockwise around the positive z-axis with the zero angle along the x-axis.

The overall structure of the XML definition is shown by Fig. 1. The *CommonRoad* root element contains one or more elements of type *lanelet*, *staticObstacle*, *dynamicObstacle*, *trafficSign*, *traf-*

ficLight, *intersection*, and *planningProblem*. Each has a unique³ ID (of type positive integer) making it possible to reference it. Additionally, the root element contains a *location* and *tag* to describe a scenario. 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, and the symbol # a comment. If an element is omitted (number of allowed elements is ≥ 0), the default value is none or its default value is specified in the description of this element below.

```
[1] commonRoad (commonRoadVersion, ...)
├── [1] tags
│   └── [1..N] tag: urban/interstate/.../intersection/
├── [1] location
│   ├── [1] geoNameId
│   ├── [1] gpsLatitude
│   ├── [1] gpsLongitude
│   ├── [0..1] geoTransformation
│   │   ├── [0..1] geoReference
│   │   └── [0..1] additionalTransformation
├── [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] stopLine
│   ├── [1..N] laneletType: highway/urban/busStop/.../crosswalk
│   ├── [0..N] userOneWay: vehicle/car/truck/bicycle/.../pedestrian
│   ├── [0..N] userBidirectional: vehicle/car/truck/bicycle/.../pedestrian
│   ├── [0..N] trafficSignRef (ref to trafficSign)
│   └── [0..N] trafficLightRef (ref to trafficLight)
├── [0..N] trafficSign (id)
├── [0..1] trafficLight (id)
├── [0..N] intersection (id)
└── [1..N] incoming
```

³Unique within the whole XML document.

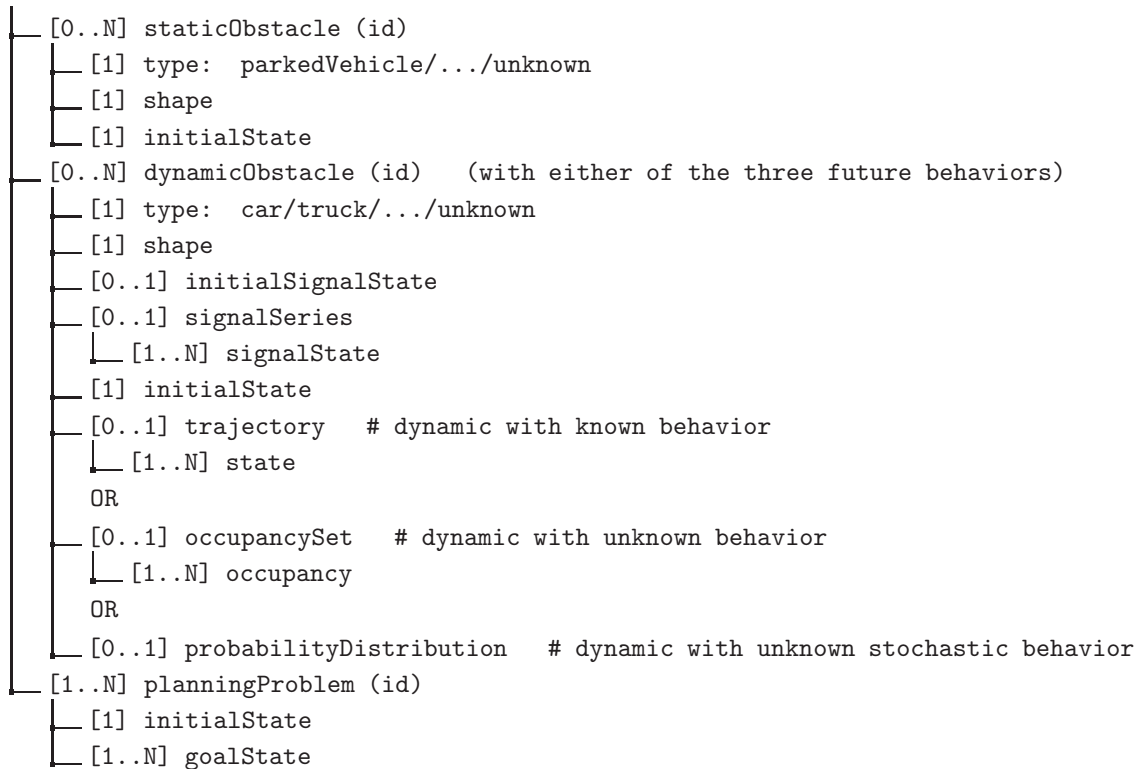


Figure 1: Structure of the XML files encoding each scenario

2.1 Meta Information Describing CommonRoad Scenarios

2.1.1 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. 2.1.2),
- **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,
- **timeStepSize**: global step size of the time-discrete scenario.

2.1.2 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 standard⁴, e.g. Germany has DEU and United States has USA. If a scenario is based on an artificial road network, we use ZAM for Zamunda⁵.

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.

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: Possible examples of a benchmark ID are: C-USA_US101-1_123_T-1, DEU_FFB-2_42_S-4, DEU_Hhr-1_1.

2.1.3 Location of Scenarios

The location element consists of (1) a GeoName-ID⁶, (2) GPS latitude coordinate, (3) the GPS longitude coordinate, and (4) a geometrical transformation introduced in the following subsection. If the GeoName-ID and the GPS coordinates are unknown, e.g., in artificial road networks, the GeoName-ID is set to -999 and the values of the coordinates are set to 999.

GeoTransformation Element To specify the geometrical transformation which were performed while creating the lanelets, one can add a **geoTransformation** element. This may contain two children:

1. The optional **geoReference** element contains a *proj-strings*⁷ describing a coordinate transformation from geodetic coordinates to the projected (Cartesian) coordinates used in the XML file. This projection can then be used to transform the Cartesian coordinates back to geodetic coordinates used by OSM (cf. Sec. 2.3.2).
2. The **additionalTransformation** element (see Fig. 2) describes geometrical operations which were performed (after the geoReference) to transform the Cartesian coordinates. The execution order of the transformations is according to the order of the following list:

⁴<https://www.iso.org/obp/ui/#search/code/>

⁵en.wikipedia.org/?title=Zamunda

⁶[geonames.org](https://www.geonames.org/)

⁷proj.org

- `xTranslation` Translating x-coordinates of all points by this value.
- `yTranslation` Translating y-coordinates of all points by this value.
- `zRotation` Rotating all points by this value around the origin, with respect to a right-handed coordinate system.
- `scaling` Multiplying all x- and y-coordinates by this value.

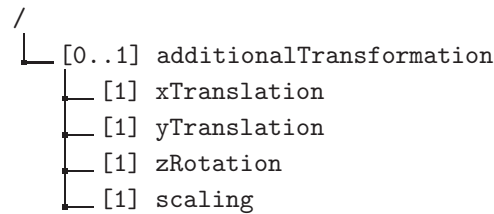


Figure 2: Element *additionalTransformation*

2.1.4 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 element `tags` in each XML file. Additionally, the filtering based on the number of static obstacles, dynamic obstacles, obstacle types, type of future behavior of obstacles, number of ego vehicles, number of goal states, and time horizon of the scenario is possible.

2.2 Auxiliary Elements

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

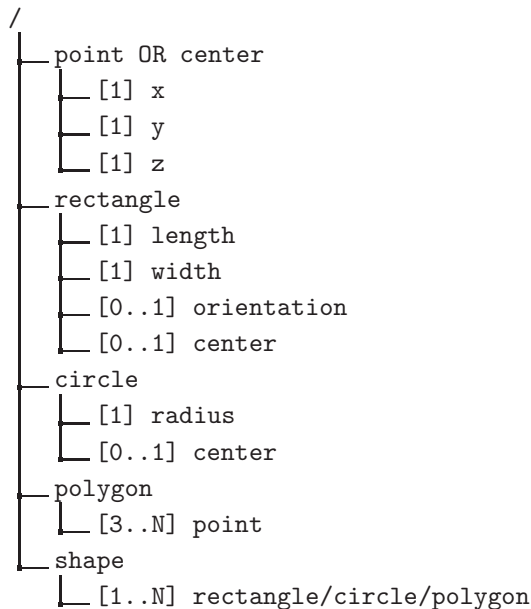


Figure 3: Auxiliary elements of the XML file

Point A point is the simplest primitives and described by an x-, y-, and z-coordinate. If the z-coordinate is zero (for all two-dimensional scenarios), we omit the z-element.

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 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 all its coordinates 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 (unless for a planning problem as specified later), as shown in Fig. 4.

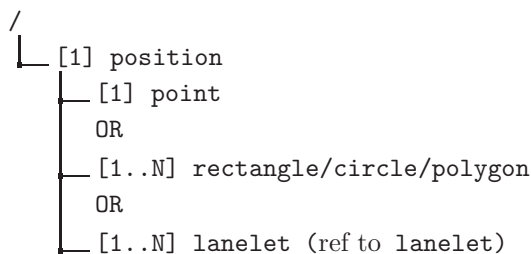


Figure 4: Element *position*.

Note that if the position of an object is given as an area (i.e. not a single point), the area does not enclose the geometric shape of the object, but only models the interval of possible positions, e.g. the uncertainty of the position measurement.

Numeric Values Elements describing the state of an object, e.g. orientation or velocity, can have either an exact value or an interval of values, e.g. to specify the goal state or to include uncertainties. For example, an *orientation* element can be defined using `exact` or `intervalStart` and `intervalEnd`:

```

<orientation>
  <exact>0.0</exact>
  <!-- or -->
  <intervalStart>-1.5708</intervalStart>
  <intervalEnd>1.5708</intervalEnd>
</orientation>

```


Time All time elements are not given as numeric values, but as integers (i.e. non-negative whole numbers). Thus, the time element can specify the time stamp of an time-discrete object. Since the initial time is always 0 and the constant time step size is given in the CommonRoad root element, the time in seconds can be directly calculated.

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. 5. Optionally, line markings (solid, dashed, ...) can be included to model the boundary more precisely. We have chosen lanelets since they are as expressible as other formats, such as e.g. OpenDRIVE⁸, yet have a lightweight and extensible representation. Our converter from OpenDRIVE to Lanelets is available on our website.

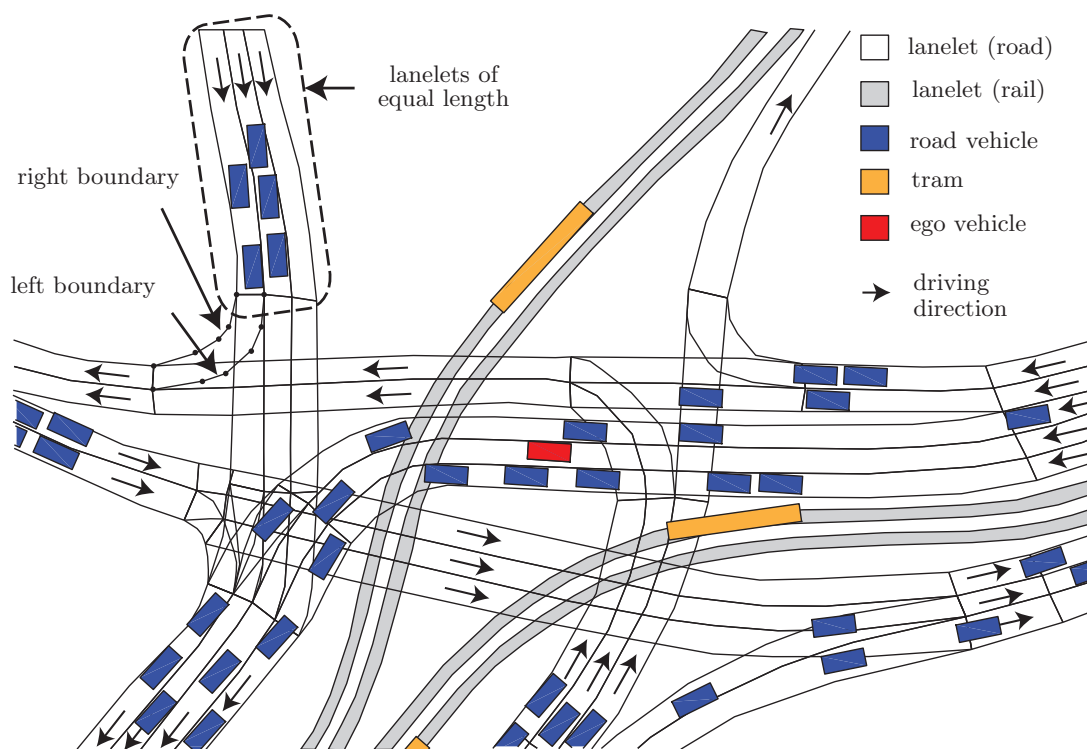
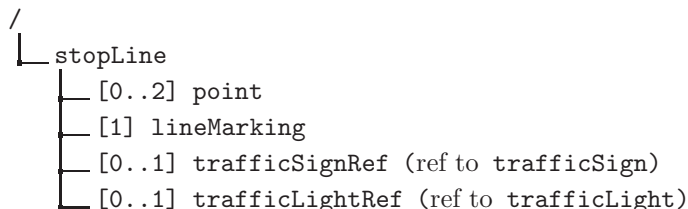


Figure 5: Lanelets of a complex intersection in the city center of Munich. Besides roads, also tram rails are modeled by lanelets.

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 to their unique ID. The elements *predecessor* and *successor* can be used 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.

⁸opendrive.org

Additionally, the element *stopLine* can be used to model a stop line or give way line (see Fig. 6). This element is defined by two points, representing the line. If no points are given for a stop line, we assume that its points correspond to the end points of the lanelet it belongs to. Optionally, traffic lights or traffic signs associated with this stop line can be specified to model the traffic conditions more precisely. Similarly to the neighboring lanelets, traffic signs and traffic lights are already present in the XML file. Thus, we use an attribute referring to their unique ID. By defining the line marking (solid or dashed), a stop line or give way line can be represented.

Figure 6: Element *stopLine*

For precise modelling of traffic conditions, additional properties of a lanelet are required. The element *laneletType* can be used multiple times to clearly define the type of a lanelet. In order to specify which types of traffic participants are allowed to use a lanelet and in which direction, the elements *userOneWay* and *userBidirectional* can be included. The supported types and users of lanelets are listed in Tab. 1.

Table 1: Types and users of lanelets.

LaneletType	urban, country, highway, driveWay, mainCarriageWay, accessRamp, exitRamp, shoulder, busLane, busStop, bikeLane, sidewalk, crosswalk, interstate, unknown
User	vehicle (car, truck, bus, motorcycle, priorityVehicle), car, truck, bus, priorityVehicle, motorcycle, bicycle, pedestrian, train

Optionally, traffic signs or a traffic light valid for a lanelet can be included by referring to the unique ID of the respective element.

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 ensure 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. 7.

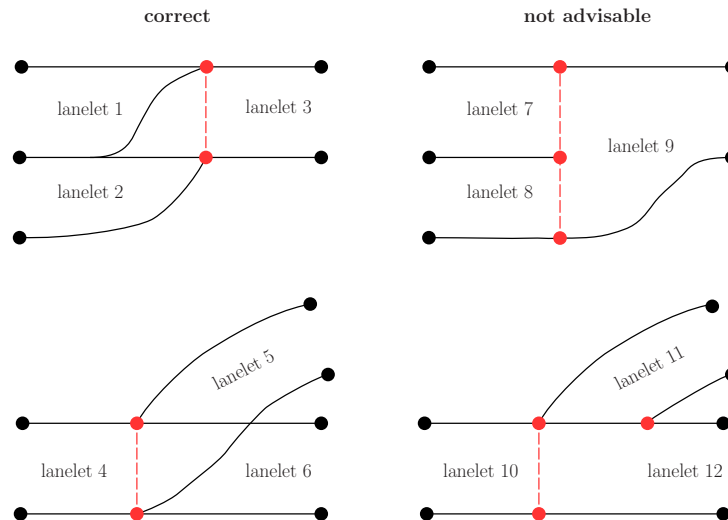


Figure 7: Spatial division of merging and forking lanelets.

- Roads are divided in so called *Lane Sections*⁹. As shown in Fig. 8, 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. 8, lanelet 1 and 2 are lateral adjacent to each other; as well as lanelet 4, 5, and 6; and lanelet 7 and 8).

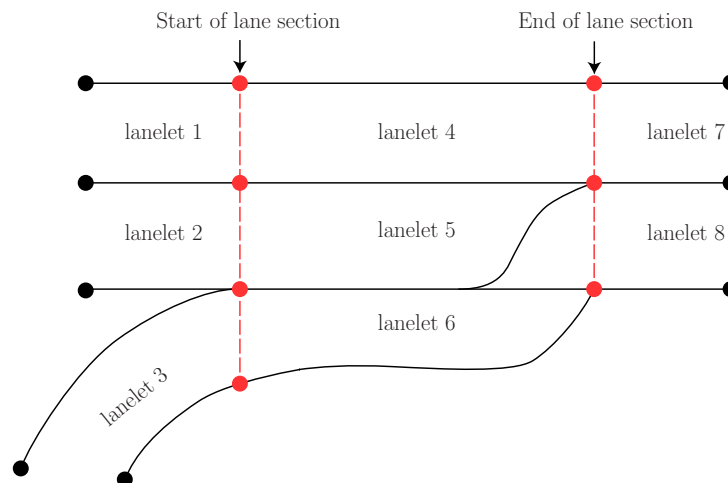


Figure 8: Definition of lane sections.

2.3.2 Connection to OpenStreetMap

As proposed in [2]¹⁰, our XML structure is inspired by the data structure of OpenStreetMap¹¹ (OSM), which uses the elements *nodes*, *ways* and *relations*.

With the XML format of OpenStreetMap, it is easy to create new road networks using *JavaOpenStreetMap* (JOSM)¹² and also to edit existing road networks. However, the OSM format has several shortcomings when representing traffic scenarios, e.g. non-Cartesian coordinate frame

⁹opendrive.org/

¹⁰github.com/phbender/liblanelet

¹¹openstreetmap.org

¹²josm.openstreetmap.de

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 Traffic Signs

The element *trafficSign* is used to represent different traffic signs within the scenario (see Fig. 9). Traffic signs are always valid either from the beginning of a lanelet or the end of a lanelet. For example, a priority sign at an intersection is valid from the end of the incoming lanelet. Therefore, it is not necessary to place the sign on the lanelets which go straight, left, or right. On the other hand, a speed limit sign is valid from the beginning of a lanelet so that the speed limit has to be considered for the entire lanelet.

A *trafficSign* element consists of one or multiple *trafficSignElements*. This way, a combination of traffic signs can be represented. Additionally, the position of the traffic sign can be included and it can be specified whether the traffic sign is existent in the real world by the element *virtual*, e.g., in the case one wants to add a speed limit which is in the real world outside of the captured road network. If *virtual* is not specified or it is set to false, we assume that the traffic sign exists also in the real world at this position. A *trafficSignElement* is defined by the *trafficSignId* specified in national traffic laws and can be further described by one or multiple *additionalValues*. For example, the *trafficSignElement* for a speed limit of 16.66 m/s in Germany would be defined as:

```
<trafficSignElement>
<trafficSignId>274</trafficSignId>
<additionalValue>16.66</additionalValue>
</trafficSignElement>
```

The traffic signs from Germany currently supported by *CommonRoad* are listed in Tab. 2. In Tab. 3, the German traffic signs valid from the lanelet beginning and lanelet end are summarized. Supported traffic signs from other countries are specified in the *CommonRoad* XSD.

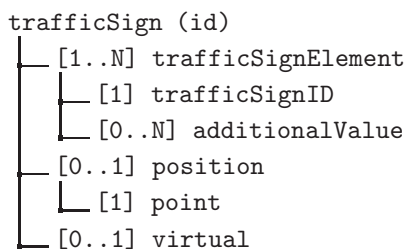


Figure 9: Element *trafficSign*.

2.5 Traffic Lights

The element *trafficLight* is used to represent traffic lights within the scenario (see Fig. 10). Each phase/color of a *trafficLight* and its duration is defined by the element *cycleElement*. Similarly to the *time* elements, the duration is not given as numeric value, but as integer. The order of the different phases is determined by the order of the *cycleElements* in the element *cycle*. By specifying the element *timeOffset*, the cycle is shifted by this value.

In order to define for which driving direction the traffic light is valid, i.e., the direction arrow(s) in a traffic light, the direction can be included. If the direction is not specified, the traffic light is valid for all directions. Additionally, the element *active* can be used to determine whether a

Table 2: German traffic sign IDs supported by CommonRoad with the corresponding symbol.














Symbol ¹³	Description	Traffic Sign ID
	Right before left rule	102
	Yield	205
	Stop	206
	Right of way	301
	Priority road	306
	Turning priority road	1002-10, 1002-12, 1002-13,
		1002-20, 1002-22, 1002-23
		1002-11, 1002-14, 1002-21, 1002-24
	Speed limit	274
	Required speed	275
	No overtaking (except of non-motorized traffic participants, trains, and motorcycles without sidecar)	276
	Green arrow sign	720
	Town sign	310

Table 3: Validity of traffic signs.

Valid from	Traffic Sign ID
Start	274, 275, 276, 310
End	102, 205, 206, 301, 306, 1002, 720

traffic light is active or not. Optionally, the position of the traffic sign can be included. Traffic lights are always valid starting from the end of a lanelet.

Figure 10: Element *trafficLight*.

2.6 Intersections

The element *intersection* is used to represent an intersection within the road network (see Fig. 12). An *intersection* element is defined by multiple *incomings*, which consist of *incomingLanelets*, *successorsRight*, *successorsStraight*, *successorsLeft* and the element *isLeftOf*.

The elements *successorsRight*/*Straight*/*Left* are used to infer for which lanelets the traffic light is valid for and to facilitate the calculation of priorities at intersections. The element *isLeftOf* is used to infer the right-before-left-rule. Since all elements of a *incoming* are already present in the XML file, we use an attribute referring to their unique ID. An example for an *intersection* with all its elements is illustrated in Fig. 11.

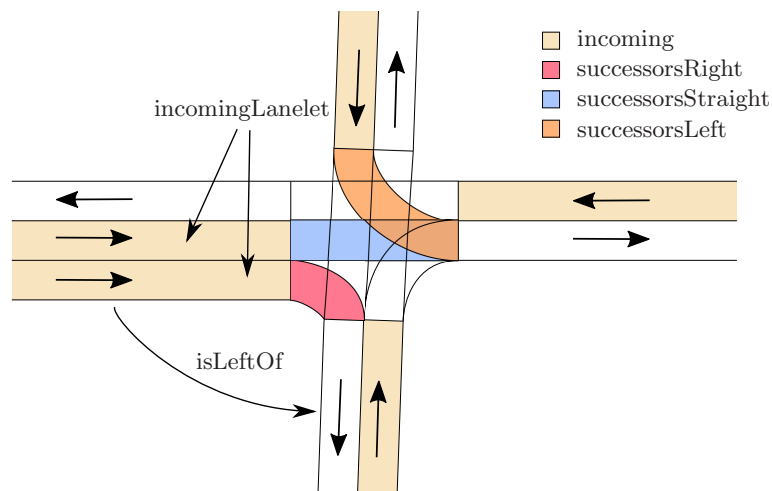


Figure 11: Example intersection. For better visibility, only one example lanelet is highlighted for each *successorsRight*/*Straight*/*Left* element.

2.7 Obstacles

The elements *staticObstacle* and *dynamicObstacle* are used to represent different kinds of traffic participants within the scenario. Each obstacle element can have a *type* as listed in Table 4.

The dimensions of an obstacle is specified by the element *shape* (cf. Sec. 2.2), and its initial configuration by the element *initialState*. Additionally, the element *initialSignalState* can be

¹³commons.wikimedia.org

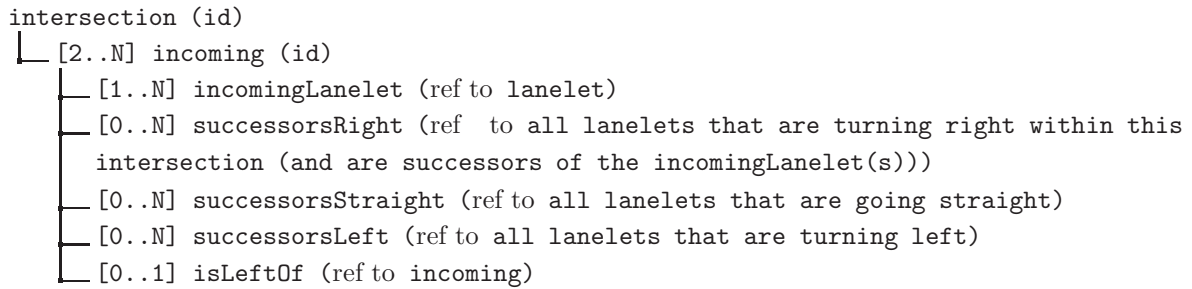


Figure 12: Element *intersection*.

Table 4: Types of obstacles.

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

included, to account for properties which are not related to the dynamic of an obstacle, e.g., whether the indicators are turned on or off.

Initial state of obstacles The configuration of an obstacle at the initial time ($t = 0$) is specified by the element *initialState* with the following state variables: *position*, *orientation*, *time*, *velocity* (scalar), *acceleration* (scalar), *yawRate*, and *slipAngle*, as shown in Fig. 13.

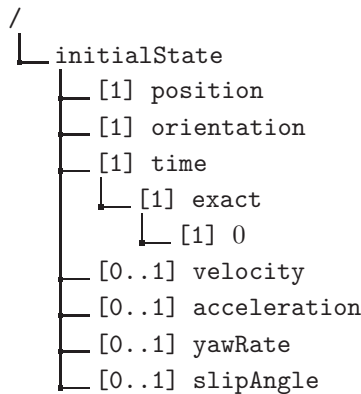
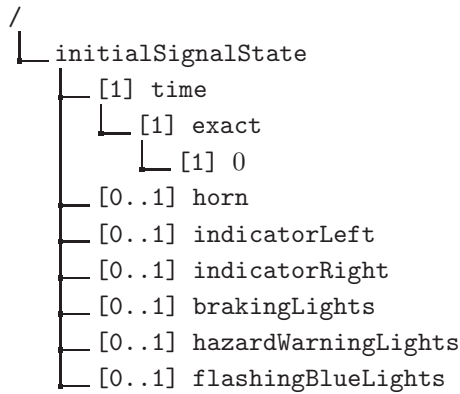


Figure 13: Element *initialState* of an obstacle, where each state variable (except time) can be exact or an interval.

Initial signal state of obstacles The state of various signals of an obstacle at the initial time ($t = 0$) is specified by the element *initialSignalState* with the following state variables: *time*, *horn*, *indicatorLeft*, *indicatorRight*, *brakingLights*, *hazardWarningLights*, and *flashingBlueLights*, as shown in Fig. 14.

2.7.1 Static Obstacles

A static obstacle has no further information, as shown in Fig. 1.

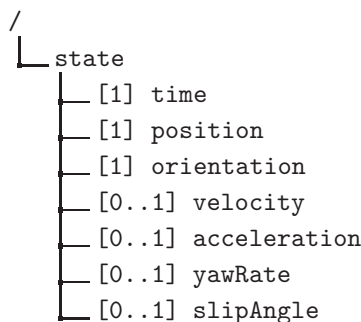
Figure 14: Element *initialSignalState* of an obstacle.

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

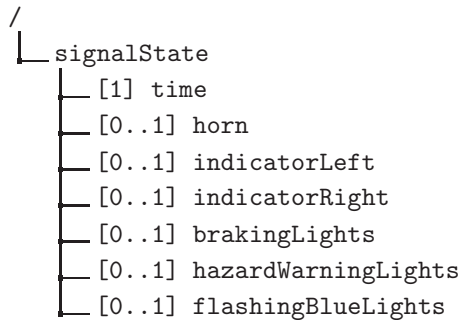
2.7.2 Dynamic Obstacles with Known Behavior

A dynamic obstacle with known behavior contains a trajectory of states and a series of signals (cf. Fig. 1). The trajectory allows us to represent the states of a dynamic traffic participant along a path for $t > 0$. 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. The signal series is obtained from a simulator 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. 15.

Figure 15: Element *state* of a trajectory, where each state variable can be exact or an interval.

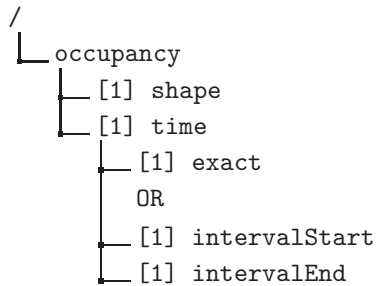
Signal States The time-discrete states of a signal series are specified by the element *signalState* with the following state variables: *time*, *horn*, *indicatorLeft*, *indicatorRight*, *brakingLights*, *hazardWarningLights*, and *flashingBlueLights*, as shown in Fig. 16.

Figure 16: Element *signalState* of a signal series.

2.7.3 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 an *occupancy set*, which represents the occupied area over time by bounded sets. As shown in Fig. 1, an *occupancy set* contains a list of *occupancy* elements.

Occupancies The *occupancy* element consists of a shape (occupied area) and a time, as shown in Fig. 17.

Figure 17: Element *occupancy* of an occupancy set.

2.7.4 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.

2.8 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

3. CONCLUSIONS

be given exact, as shown in Fig. 18. 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*.

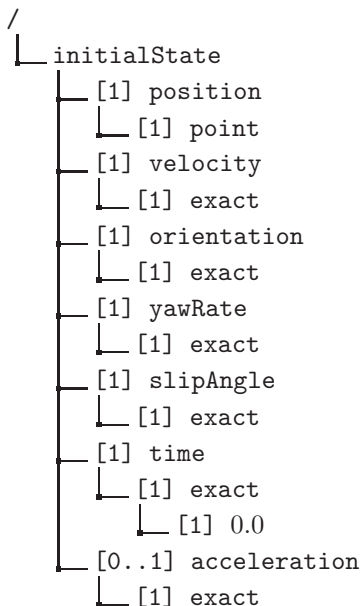


Figure 18: Element *initial state* of a planning problem

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. 19.

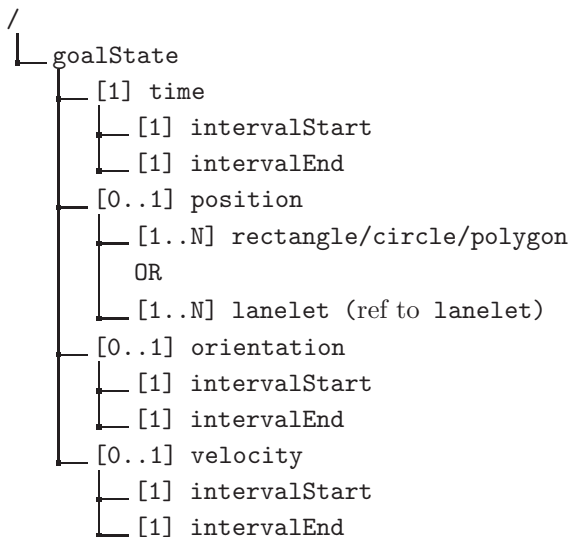


Figure 19: Element *goal state* of a planning problem

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* website¹⁴. 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. Manzingler. 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.

¹⁴commonroad.in.tum.de