YET ANOTHER LABELING FORMAT?
YES, BUT THE ONLY FORMAT YOU WILL EVER NEED.

OPEN LABELING FORMAT

PROJECT DETAILS

Name	Whitepaper / Technical Specification
Doc. Status	Draft
Author / Owner	HAGL
Document ID	-

DATE	MODIFIED BY	DESCRIPTION
10.01.2020	CS	Draft for version 0.5

HELLA Aglaia Mobile Vision GmbH
A member of the HELLA group
Ullsteinstraße 140
12109 Berlin / Germany

phone +49 (0) 30 2000 429-0
fax +49 (0) 30 2000 429-109
mail info@hella-aglaia.com
www.hella-aglaia.com

Deutsche Bank AG
IBAN: DE 07 4167 0027 0609 8420 00
BIC SWIFT: DEUTDE3B416
AG Berlin-Charlottenburg
HRB 66976 B
VAT-ID No.: DE 194710861
EORI No.: DE 5106826
Managing Director: Kay Talmi

1 ABSTRACT

The rise of deep learning within supervised machine-learning enabled tremendous achievements in visual perception tasks. However, these approaches are critically dependent on the availability of huge amounts of labeled sensor data. This means that humans or reference algorithms have to identify and store thousands, millions or even billions of data samples that are then fed into training, validation and testing pipelines.

The machine learning and visual perception communities have access to several (multi-) sensor datasets but almost all of them introduce new representations on how to store these labels. With the Open Labeling Format (OLF), we propose a unified (not only) automotive sensor labeling format aiming a consolidation of most available formats.

2 THE FORMAT LANDSCAPE

Datasets for visual perception tasks usually bring their own format. An increasing number of automotive-related datasets have been released throughout the past few years:

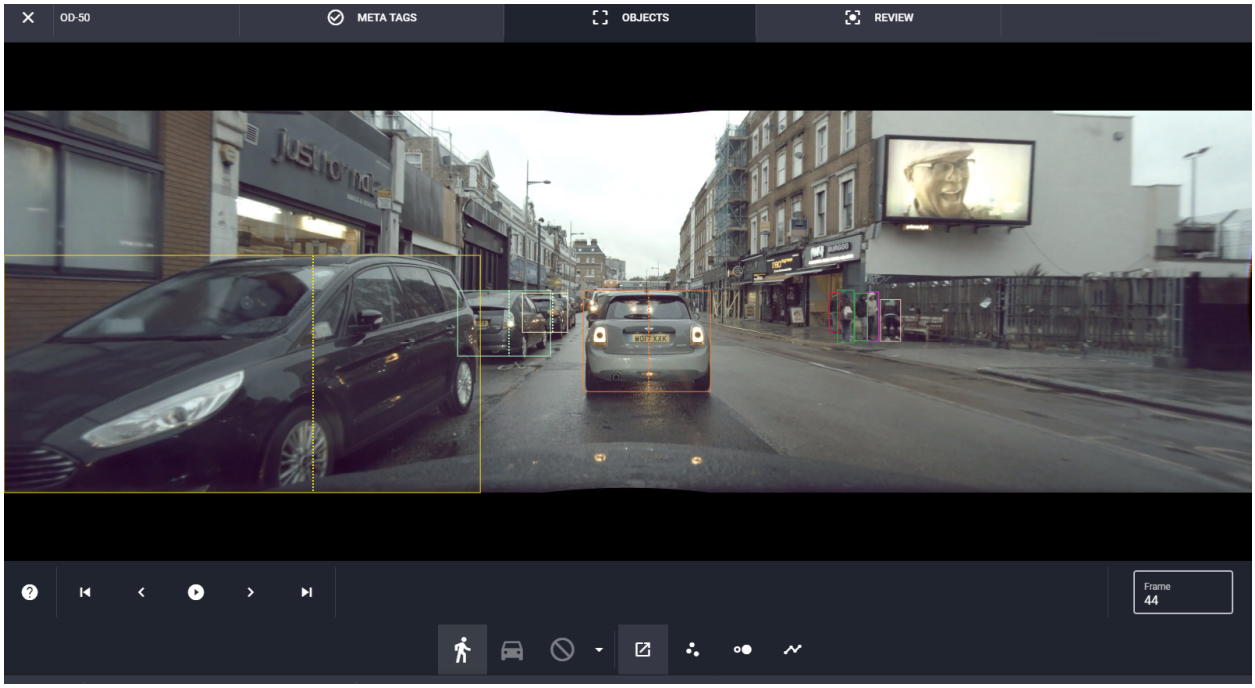
Dataset	Year	Sensor	Labeling Format
KITTI	2013	Cam, Lidar	Proprietary
CityScapes	2016	Cam	Proprietary
Mapillary Vistas	2017	Cam	Proprietary
Oxford RobotCar Dataset	2017	Cam, Lidar	Proprietary
ApolloScape	2018	Cam. Lidar	Proprietary
Berkeley Deep Drive	2018	Cam	Proprietary[1]
Comma 2K19	2018	Cam	Proprietary[2]
HD1K Benchmark Suite	2018	Cam	Proprietary
nuScenes	2018	Cam, Lidar, Radar	Proprietary
A*3D	2019	Cam, Lidar	Proprietary
AEV Autonomous Driving Dataset	2019	Cam, Lidar	Proprietary
Brno Urban Dataset	2019	Cam, Lidar	Proprietary
Lyft Level 5	2019	Cam, Lidar	nuScenes
Oxford Radar RobotCar Dataset	2019	Cam, Lidar, Radar	Proprietary[3]
Waymo Open Dataset	2019	Cam, Lidar	Proprietary

[1] Using format of own labeling tool “Scalabel”

[2] Only GNSS recordings, no object labels

[3] Some parts are shared with Oxford RobotCar Dataset

3 OLF: OVERVIEW



Any proprietary format may be suitable to represent label information like 2D bounding boxes around cars and pedestrians, attributes or other details. Since it is obvious that the value of a dataset comes from the data itself and not from the way this data is stored, the machine learning community has widely accepted the continuous adaption of their data engineering tools after releases of new datasets. However, these recurring adaptions neither foster robustness nor efficiency of these tools. They also slow down the data preparation step and thus the machine learning process as a whole. Moreover, from a functional point of view, as the different datasets support a varying subset of the same sensors, it seems straightforward that a consolidation of available labeling formats into a common standard makes sense.

This being said we are proud to present the solution to this problem: OLF – Open Labeling Format. Yet another labeling format, but the only labeling format you will need.

3.1 KEY-FACTS

- OLF can be used for classification, object detection, keypoint detection and pixel-level segmentation tasks.
  No more need for multiple labeling files with different structures. One does it all.
- OLF supports multi-dimensional object-level labeling.
  The format contains several 2-D and 3-D primitives for creating object labels within 2D or 3D space.
- OLF is sensor-agnostic.
  Labels for camera, Lidar, Radar and other sensor data can be stored, even simultaneously for the same point in time.
- OLF models relationship between objects.
  Within OLF objects can form sibling, child or parent relations enabling complex scene representations.
- OLF supports multiple media files.
  One OLF file can be used for one to multiple media files like image files, video files or laser point clouds.
- OLF supports temporal sparse labeling.
  By defining a list of points in time (or frames) that hold label information, labels for an arbitrary temporal subset within the media files can be stored.
- OLF supports auxiliary information.
  Links to external references like ego-motion, calibration and GNSS can be stored and mapped to the media files.

- ☐ OLF supports custom coordinate systems.
  Labels can be stored relative to a specified 2-D or 3D- coordinate system.
- ☐ OLF supports local and global attribute hierarchies of arbitrary depth.
  It is possible to assign single value or hierarchical attributes to an entire point in time (or frame) or to a single object.
- ☐ OLF supports IDs.
  For every object or attribute unique identifiers can be assigned.
- ☐ OLF supports history information.
  Every label can be extended with information about its creation and modification allowing monitoring of complex labeling tasks.

Additionally, the Open Labeling Format comes with several advantages:

- It is **flexible** and designed for optional support of **automotive** datasets.
- It is **readable**, since it is based on JSON.[4]
- It is **checkable** using our supplied JSON Schema[5] (draft 0.7).
- It is **open** and **free** to use.
- It is **easy** to use, since we offer a growing set of converters and scripts for existing formats.

4 OLF: DETAILS

A valid OLF file consists of 8 mandatory JSON objects that serve different purposes:

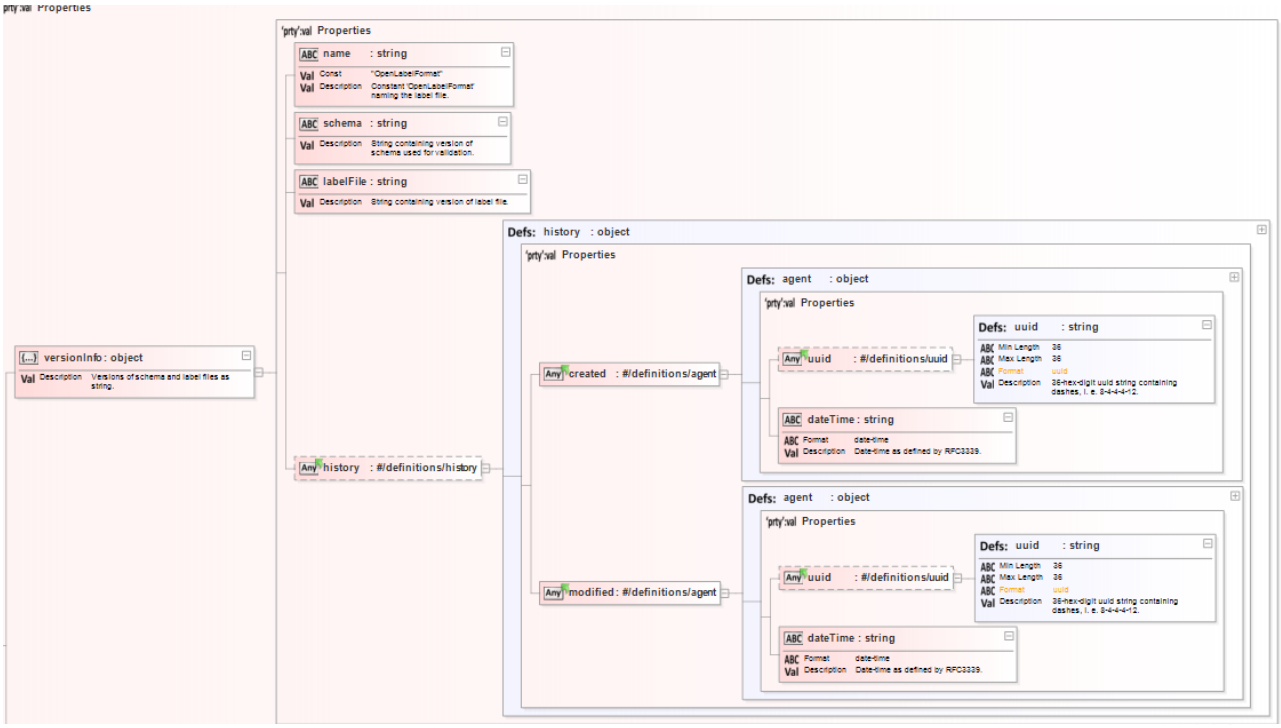
Object	Description
versionInfo	Stores information about the schema version and the version of the actual label file
projectInfo	Stores global information about the entire labeling project
odometryInfo	Stores information about referenced ego-motion data
calibrationInfo	Stores information about coordinate systems and referenced calibration data
positionInfo	Stores information about referenced position data like GNSS or inertial data
mediaInfo	Stores information about media files to be labeled
metaInfo	Stores global attributes valid in specified ranges of media
objectInfo	Stores attributes and shapes bound to a specific object in specified ranges of media

In the following sections, these objects will be described in detail.

[4] https://en.wikipedia.org/wiki/JSON

[5] https://json-schema.org/

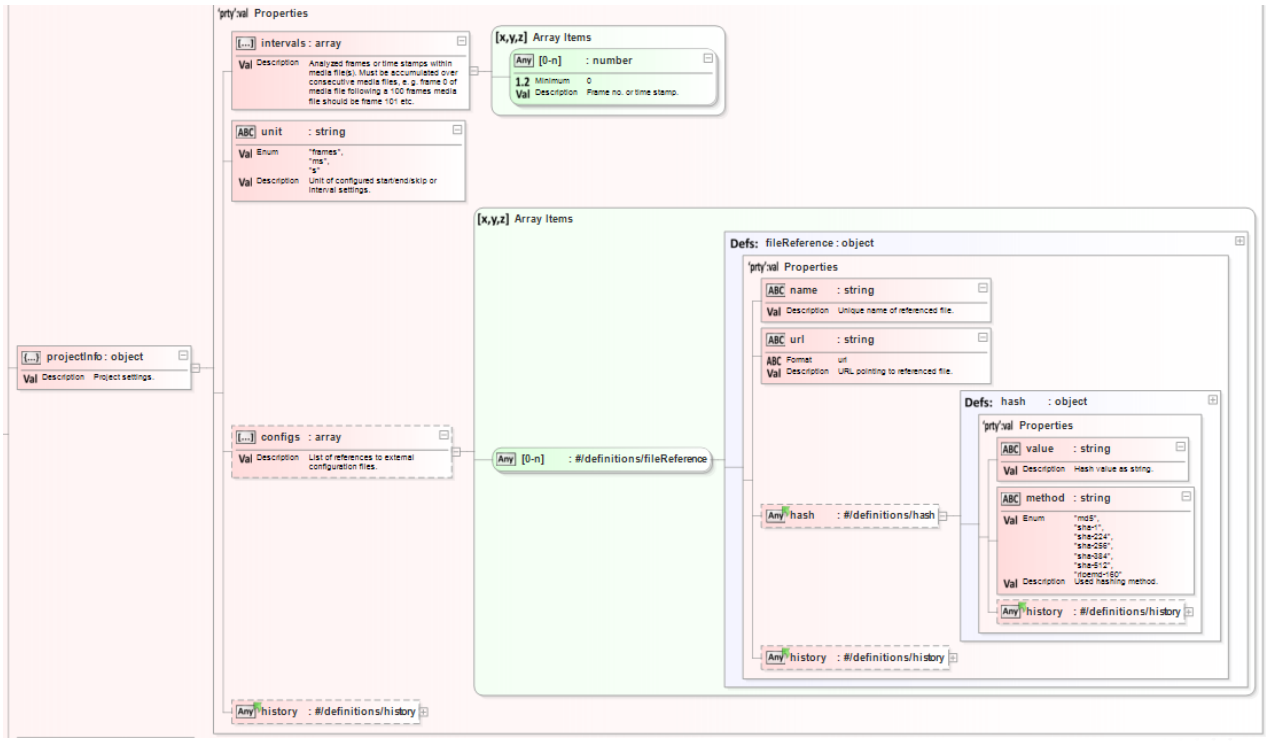
4.1 VERSIONINFO



The **versionInfo** object consists of 3 mandatory key-value pairs and an optional history object. The key **name** must have the string value "OpenLabelingFormat" and identifies the file as an OLF. The key's **schema** and **labelFile** must have string values specifying the used versions of schema and OLF file. To track changes within the OLF file history, information can optionally be stored under the key **history**. The following description of the value object is valid for all other keys named **history** throughout the schema and this document:

This object consists of two mandatory keys, named **created** and **modified**. Both are composed of an optional key **uuid** and a mandatory key **dateTime**. **uuid** expects a string value of length 36 containing 32 hex digits and 4 separating dashes, i. e. 8-4-4-4-12. It can be used to store a unique identifier of a person or machine that **created** or **modified** the OLF file. **dateTime** expects a string value for date and time represented according to RFC3339.

4.2 PROJECTINFO



The most important key here is **intervals**. Its value is a number array containing subsequent offsets relative to the beginning of the first media file. These offsets specify the points in time where labeling has been done and can be entered as frame numbers or time codes. Please note that the unit used has to be set as a string ("frames", "ms" or "s") as value for the key **unit**.

Example: 1 media file with 10 frames, every 2nd frame labeled

intervals	**0**	**2**	**4**	**6**	**8**

Example: 2 media files with 10 frames each, every 3rd frame labeled, starting 2nd media file at beginning

intervals	**0**	**3**	**6**	**9**	**10**	**13**	**16**	**19**

Example: 3 media files with 1 s duration each, every 500 ms labeled

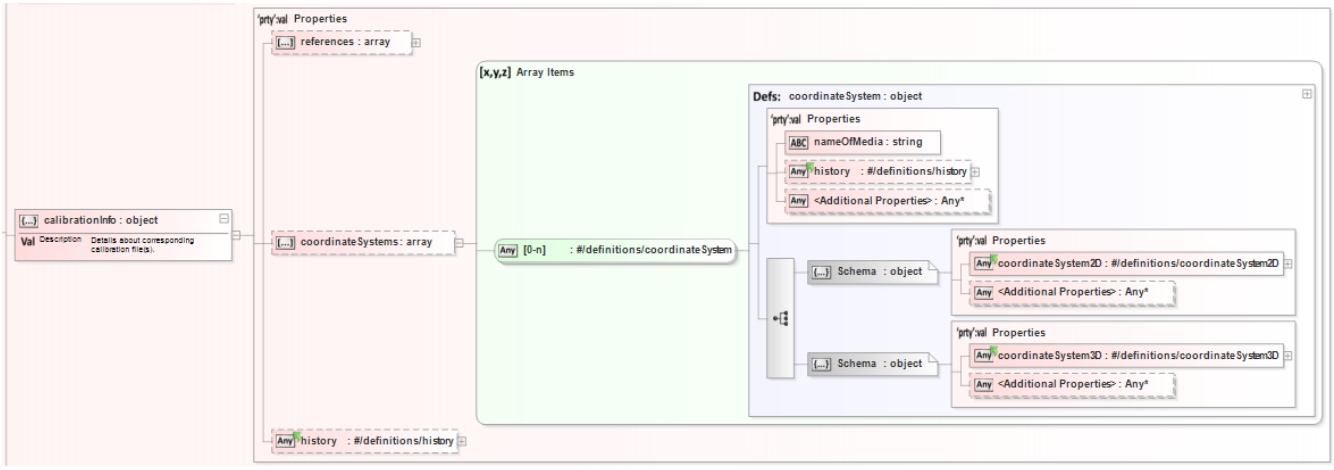
intervals	**0**	**500**	**1000**	**1500**	**2000**	**2500**	**3000**

The optional key **configs** holds an array of objects as value. Each of these objects represents a reference to an external file that might be used for the configuration of the labeling process but is not an elementary part of the label information itself. The reference object consists of two mandatory keys, **name** and **url**. Both expect string values. **name** should be assigned a unique name while **url** holds the absolute path to the referenced file. The optional key **hash** allows to store information about the integrity of the referenced file. While the key **value** holds the hash value itself, the key **method** stores the used hashing method and must be one of "md5", "sha-1", "sha-224", "sha-256", "sha-384", "sha-512" or "ripemd-160".

4.3 ODOMETRYINFO

Depending on the layout of the labeling task, it can be beneficial to include ego-motion data in the visualization or processing pipeline of the labeling tool. Hence, an array of file references can be assigned to the optional key **references** of this object. The only difference to the file references described in the last section is that every ego-motion file is exactly valid for one media file and this mapping to the corresponding media file has to be stored as string value of the key **nameOfMedia**.

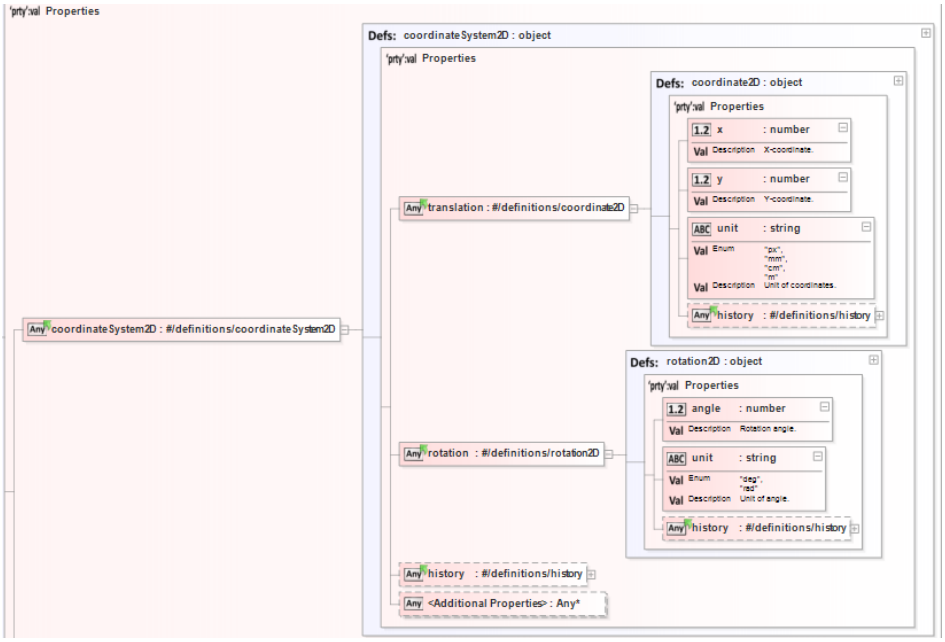
4.4 CALIBRATIONINFO

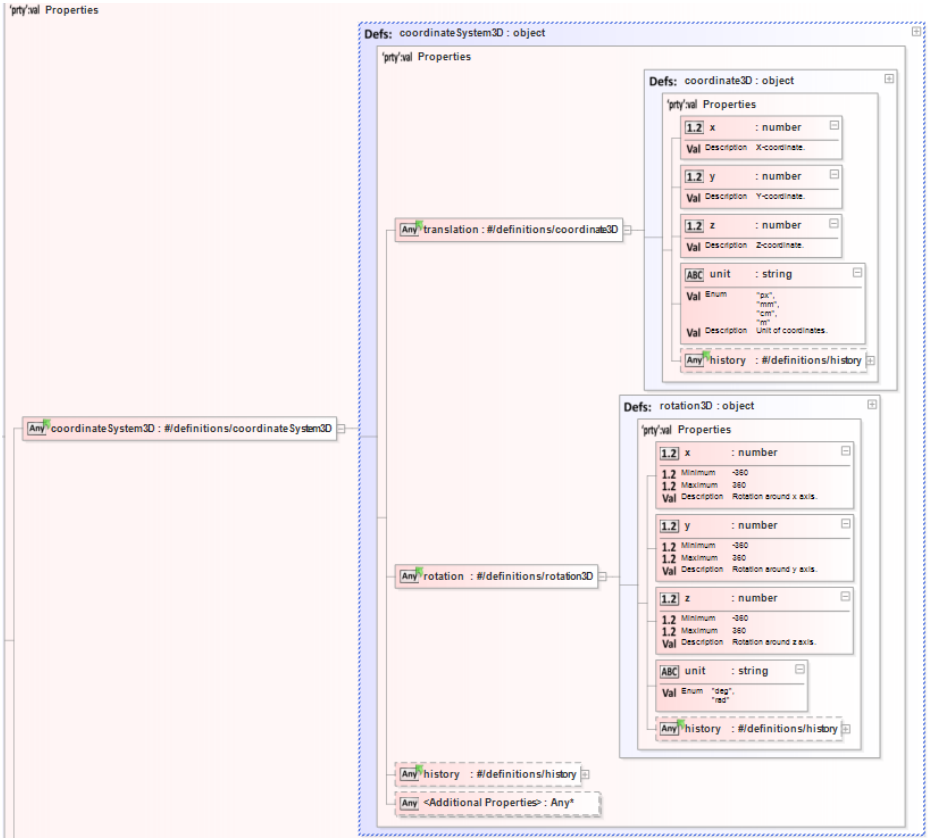


Similar to ego-motion data, the incorporation of sensor-calibration data can sometimes be beneficial for the labeling tasks. For this purpose, the object **calibrationInfo** provides the optional key **references** which allows external file references in the same manner as in **odometryInfo**.

Additionally, it might be necessary to label objects in 2D or 3D space but relative to a given coordinate system. Thus, the optional key **coordinateSystems** is supplied which allows to store one coordinate system per media file. Value of the key is an array of objects, where each object has a mandatory key **nameOfMedia** as unique reference to a media file and either a 2D or a 3D coordinate system object.

Both objects define the keys **translation** and **rotation**. However, for the 2D case the values are a 2D coordinate object (**x**, **y**) and a 2D rotation object (**angle**), while for the 3D case 3D coordinate objects (**x**, **y**, **z**) and 3D rotation objects (**x**, **y**, **z**) are used. Please note that every coordinate and rotation object also contains the key **unit** to specify the unit, i. e. “deg”, “rad” for angles and “px”, “mm”, “cm”, “m” for coordinates.

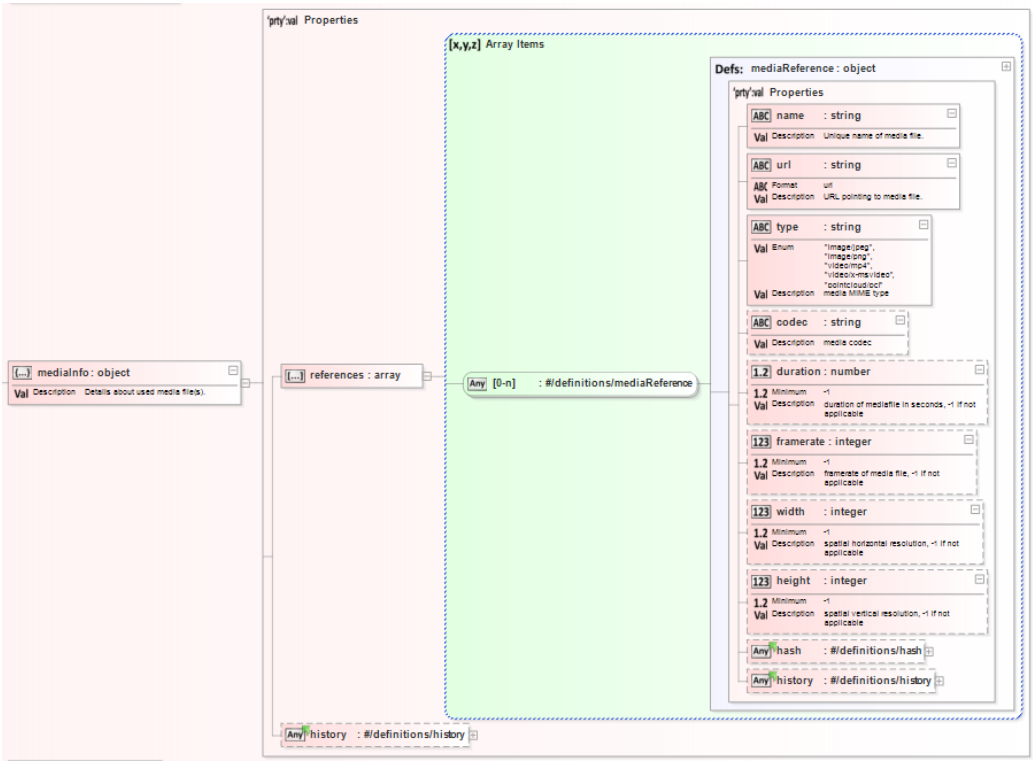




4.5 POSITIONINFO

The structure of this object is the same as for **odometryInfo**. The information that should be maintained here are references to GNSS and/or inertial data files.

4.6 MEDIAINFO



This object links the label information to the actual media files. Its mandatory key **references** expects an array of media reference objects. These objects require the specification of a unique **name** (preferably UUID) and an absolute **url** to the location of the file as string values. The last mandatory key is **type** which must be set to the MIME type of the media file, e. g. “image/jpeg”, “image/png”, “video/mp4”, “video/x-msvideo”. For media types without a known MIME type. like point-clouds in a *.pcl file, the use of self-defined MIME types like “pointcloud/pcl” is suggested.

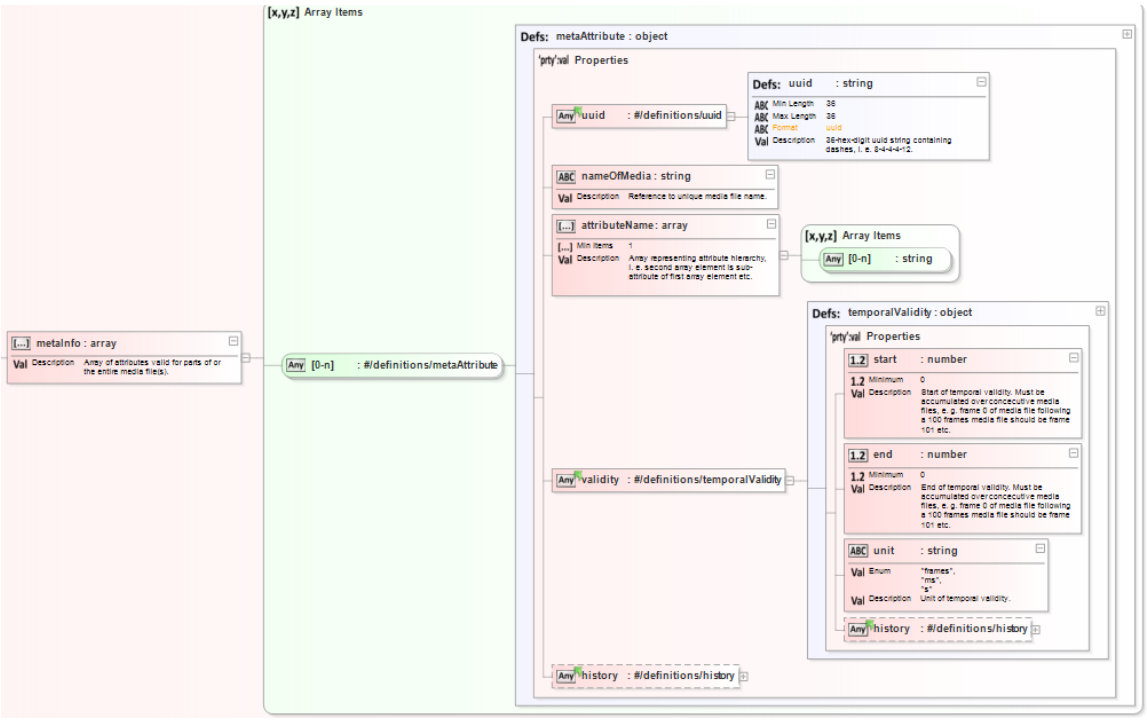
Although the OLF is designed sensor-agnostic, several optional keys are proposed within this object that are mainly relevant for image and video data:

- The **codec** of the media file as a string.
- The **duration** of the media file in seconds, set to -1 if not applicable.

- The **framerate** of the media file, set to -1 if not applicable.
- The **width** of the media file, set to -1 if not applicable.
- The **height** of the media file, set to -1 if not applicable.

As described before, the use of the optional keys **hash** and **history** is advised.

4.7 METAINFO



Attributes that are not combined with or dependent on distinct spatial locations within the media files, should be stored in the array **metaInfo**. Examples are the class of an image or global properties of a scene like weather conditions.

Every array element is an object that specifies one attribute. The key **uuid** uniquely identifies this attribute and should be stored according to the layout of a uuid as described before. **nameOfMedia** is the unique identifier of the media file this attribute belongs to. Again: This can and should be also an uuid but can be any string as long it is unique. **attributeName** represents the actual attribute including its hierarchy. This is simply encoded using an array of strings.

Example: Encode the attribute/class “car”

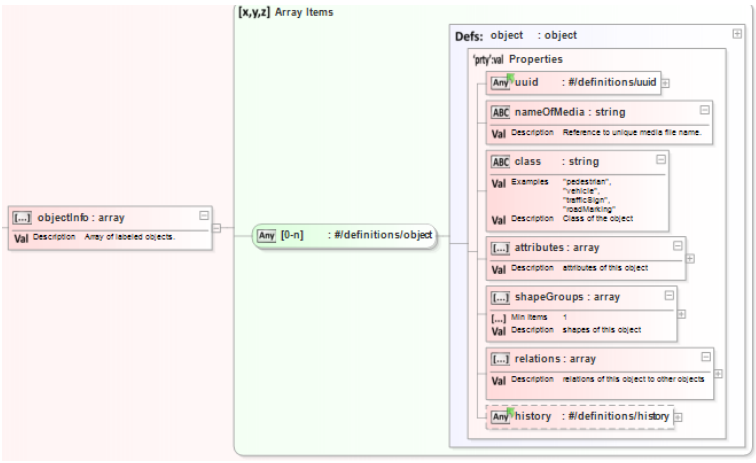
attributeName	**car**

Example: Encode the attribute hierarchy “weather” → “rainy” → “heavy”

attributeName	**weather**	**rainy**	**heavy**

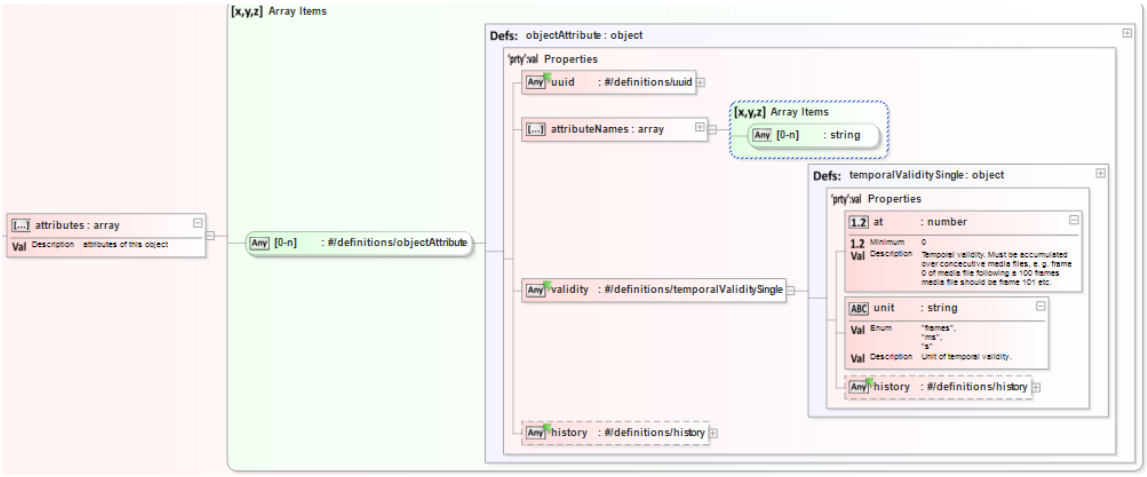
To temporally constrain the spatially global attributes, the key **validity** has to be used. The assigned object expects three keys: **start**, **end** and **unit** defining the temporal location or range of the attribute’s validity. It is important to keep these settings consistent to the **intervals** definition from the **projectInfo** object. Please also note that when defining a validity with e. g. “start” = 100, “end” = 200, “unit” = “frames” then it is assumed that the object is continuously valid between frame 100 and 200. If temporal gaps exist and are critical to be represented then the same attribute has to be defined multiple times with different validities.

4.8 OBJECTINFO



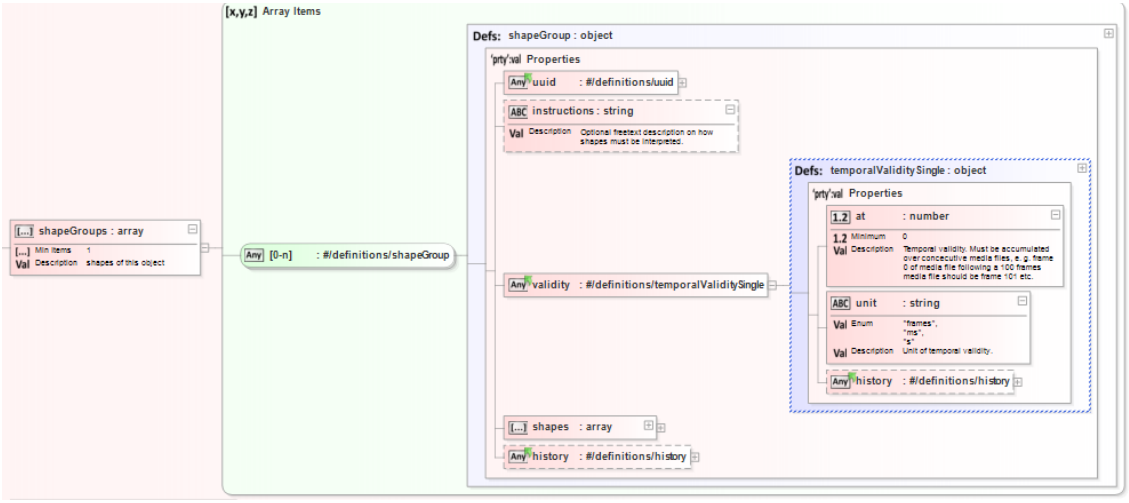
Every JSON object within the **objectInfo** array represents a physical entity visible in parts of one media file. While the keys **uuid**, **nameOfMedia** and **history** follow the same definition as in **metaInfo**, the other keys are described below.

class simply states an object's class as a string value. Examples are "pedestrian", "vehicle", "trafficSign" or "roadMarking". The array **attributes** has been defined similar to **metaInfo** from the previous section. The two differences are a missing link from the attribute to the media since this is done through the object in this case. The other difference is how temporal **validity** is defined.

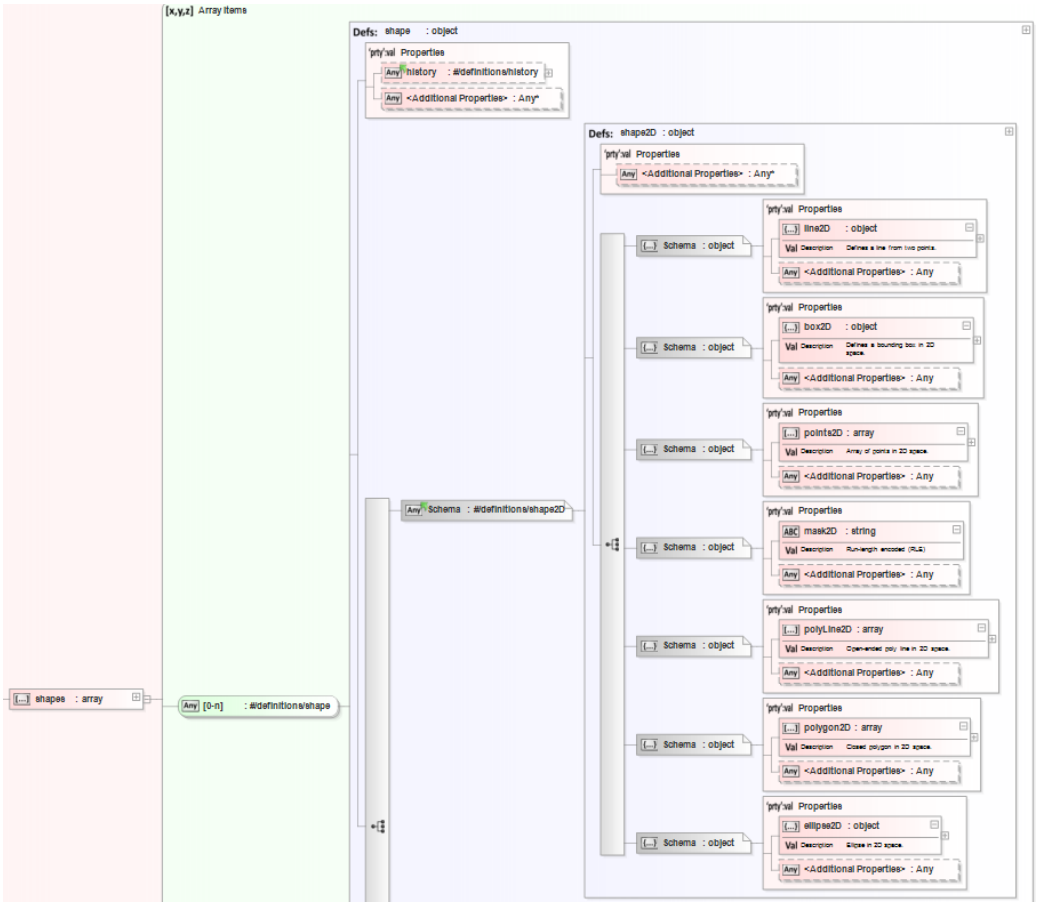


Compared to **metaInfo**, object attributes are valid **at** a specific frame number or time code (**unit** set to "frames", "ms" or "s") instead of a range. Please note that this **validity** also has to be consistent with **intervals** from **projectInfo**.

The key **shapeGroups** holds an array of objects that represent a composition of one or more **shapes**, including **instructions** how to compose them and a its **validity at** a given point in time (**unit** = "frames", "ms" or "s"). While the layout of the mandatory **uuid** of every **shapeGroup** is already known from previous sections, the value of the optional **instructions** is just a string holding free text on how to compose the contained **shapes** to form the desired label.



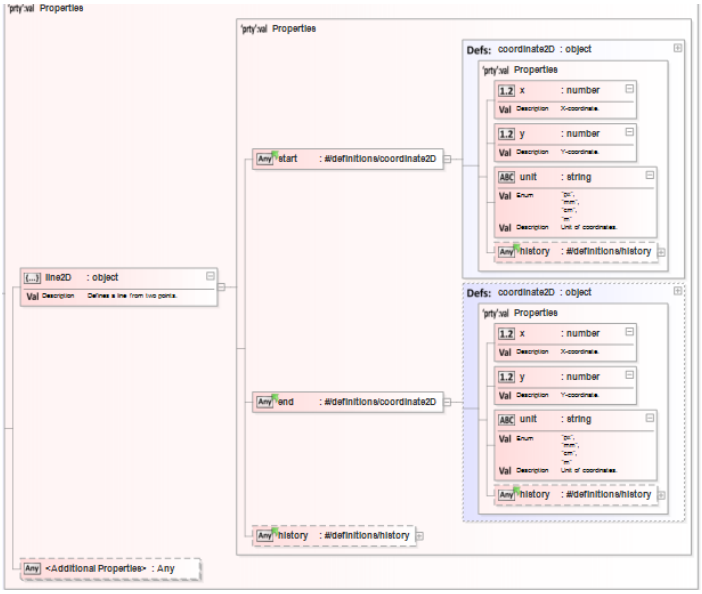
The **shapes** array within the **shapeGroups** array can take an arbitrary combination of 2D shapes and 3D shapes but it must contain at least one shape.



4.8.1 2D SHAPES

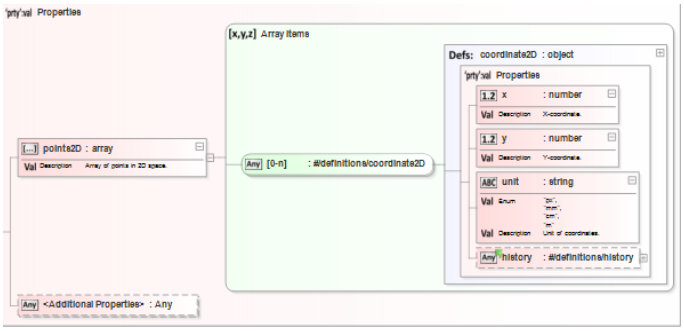
The following 2D shapes exist:

- line2D
  It is defined by a **start** and an **end** 2D-coordinate object. This coordinate object consists of the keys **x**, **y** and **unit**. Number coordinates are assigned to **x** and **y**, **unit** holds the unit as string and can be either “px”, “mm”, “cm” or “m.

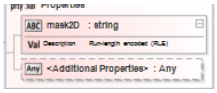


- box2D
  This shape represents a rectangular bounding box. It is defined by the two 2D coordinates already known from above, one specifying the **topLeft** and one the **bottomRight** corner of the box. In addition, a **rotation** object is required which specifies its **angle** as a number and the **unit** ("deg", "rad") as a string value.

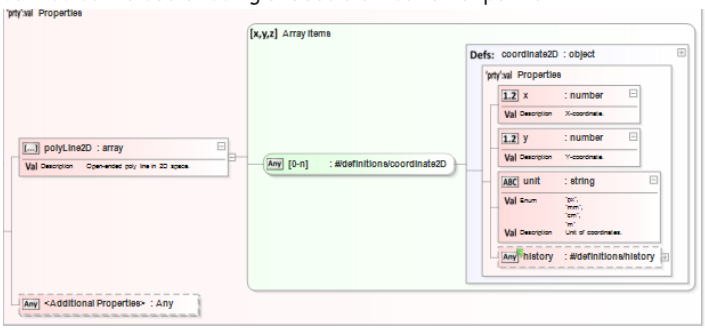
- points2D
  This is an array of 2D coordinates forming an arbitrary distribution of points in a 2D space.



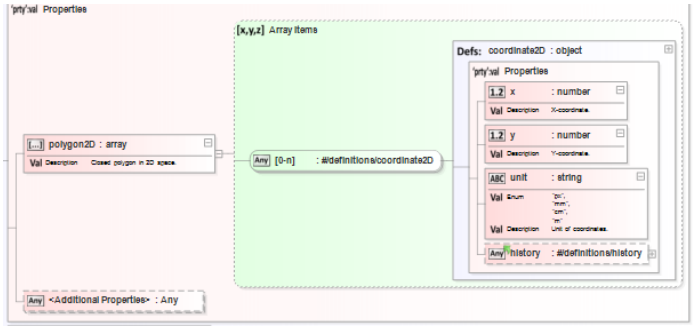
- mask2D
  This shape is intended to be used with pixel masks. When these masks are run-length encoded (RLE) they can be assigned as a string.



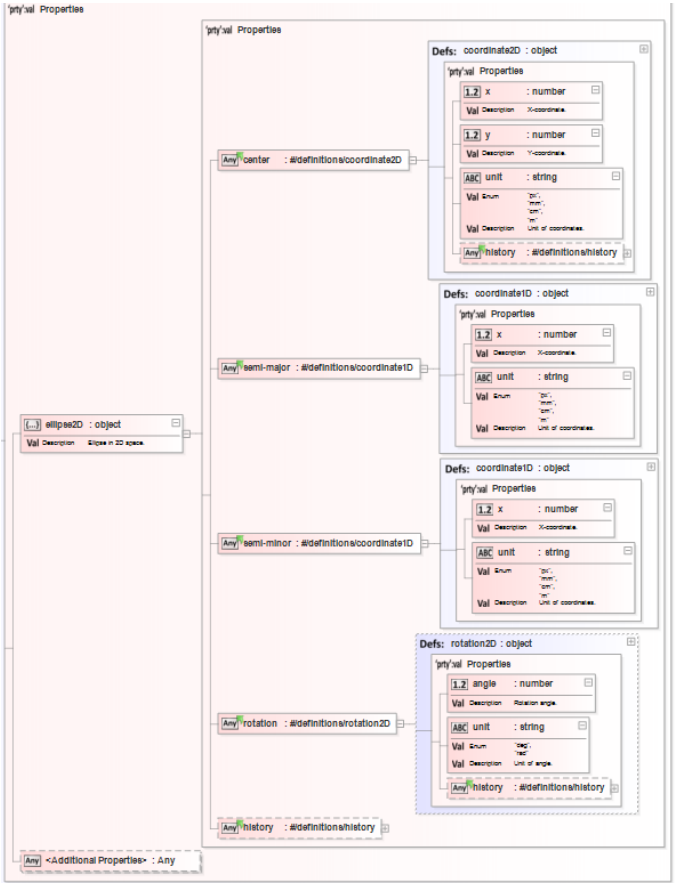
- polyLine2D
  This shape is defined identical to **points2D** but is interpreted differently. In **polyLine2D,** the points are considered connected instead of being a loose distribution of points.



- polygon2D
  This shape is defined identical to **points2D** and **polyLine2D** too, only this time it is required that the first and the last coordinates are identical as well, forming a closed polyline, hence a polygon.

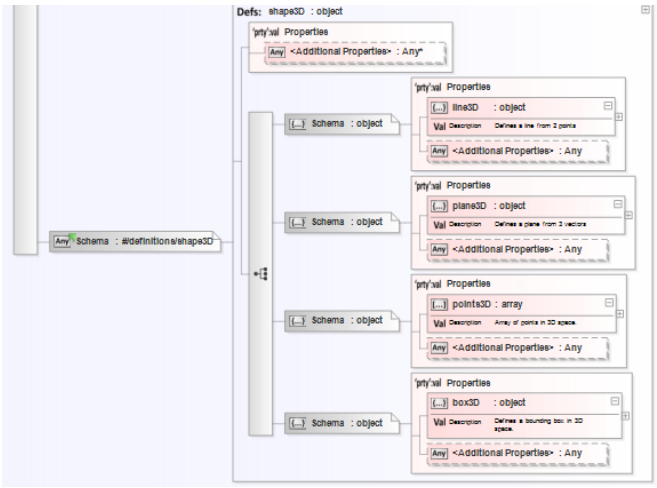


- ellipse2D
  Elliptical or circular shapes in 2D space can be described using **ellipse2D**. It is defined by the component's **center**, **semi-major**, **semi-minor** and **rotation**. **center** is the 2D coordinate of the ellipse's center point, while **semi-major** is a 1D coordinate (length) of the semi-major axis. The length of the semi-minor axis is covered by the 1D coordinate assigned to **semi-minor**. As in **box2D,** the **rotation** object with **angle** and **unit** describes the rotation of the ellipse.

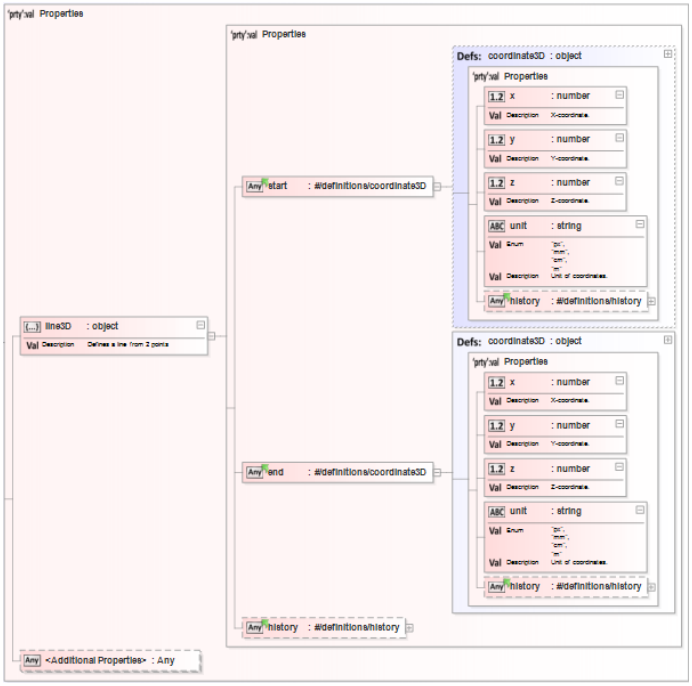


4.8.2 3D SHAPES

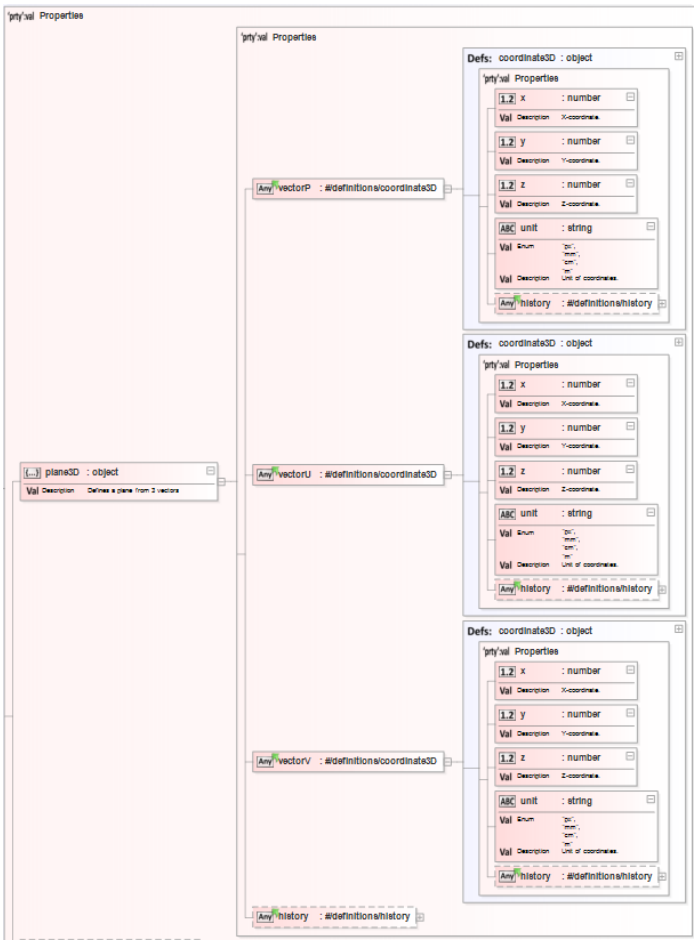
For labeling in 3D space, the following shapes are supported:



- line3D
  In 3D space a line is represented by the two 3D coordinates **start** and **end**, each composed of **x**, **y**, **z** and **unit**.



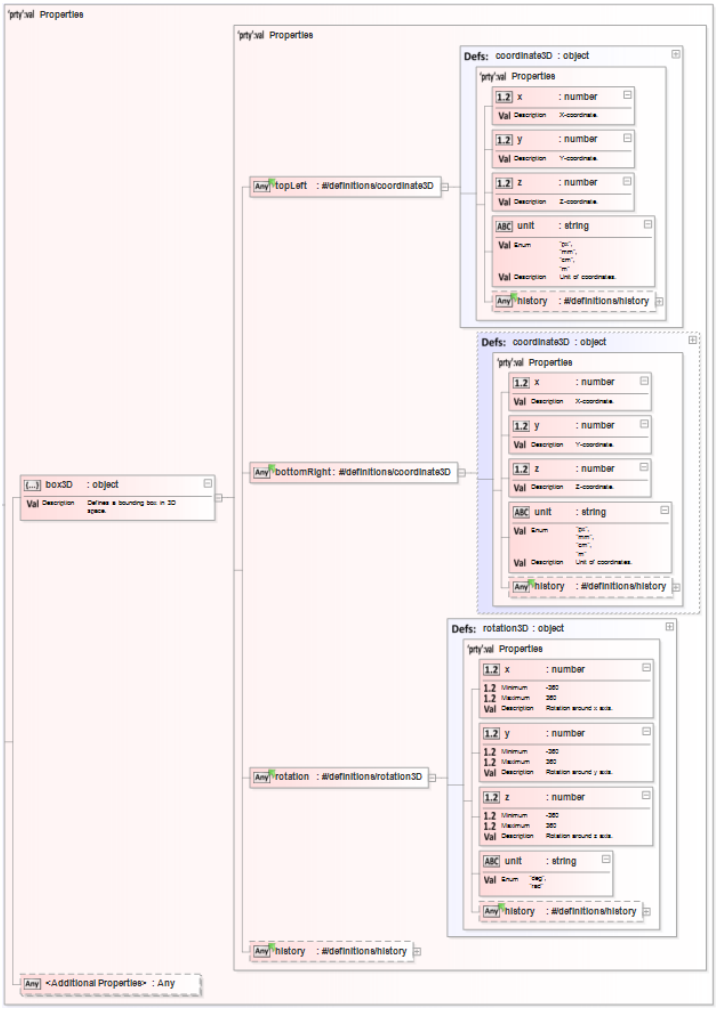
- plane3D



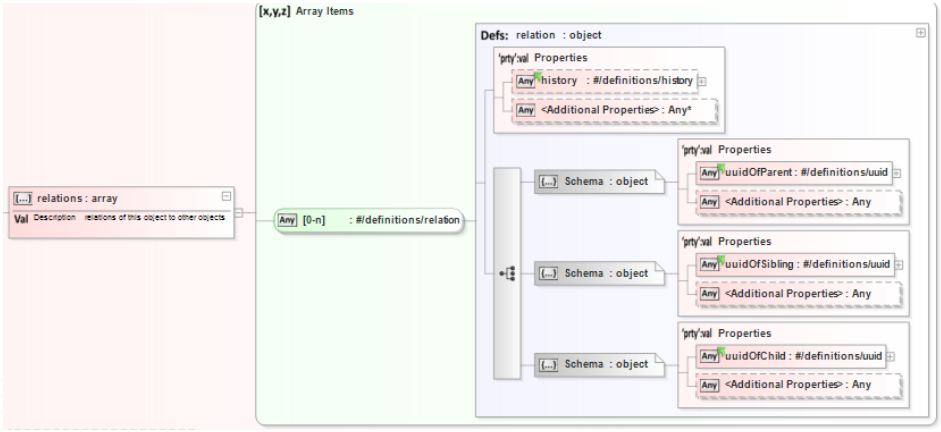
To set a plane into a 3D space, three vectors are defined: **vectorP**, **vectorU** and **vectorV**. Each vector has three elements, named **x**, **y**, **z**. Again, **unit** holds the unit of the values used ("px", "mm", "cm", "m").

box3D

In 3D space the definition of a bounding box is extended compared to **box2D**. **topLeft** and **bottomRight** have an additional (3rd) coordinate and rotation has three angles **x**, **y**, **z** (one per axis).



Finally, every object has an array called **relations**, whose elements are objects with one of the following keys: **uuidOfParent**, **uuidOfSibling**, **uuidOfChild**. The according values are unique identifiers of other objects that have a relationship with the current object.



Example: Object A with uuid [6]= 123 is dependent on object B with uuid [7]= 456. Object C with uuid [8]= 789 is also dependent on object B but not from object C.

	uuid[9]	uuidOfParent	uuidOfSibling	uuidOfChild
Object A	123	456	789	-
Object B	456	-	-	123, 789
Object C	789	456	123	-

5 LICENSE

[6] Not a valid uuid

[7] Not a valid uuid

[8] Not a valid uuid

[9] Not a valid uuid