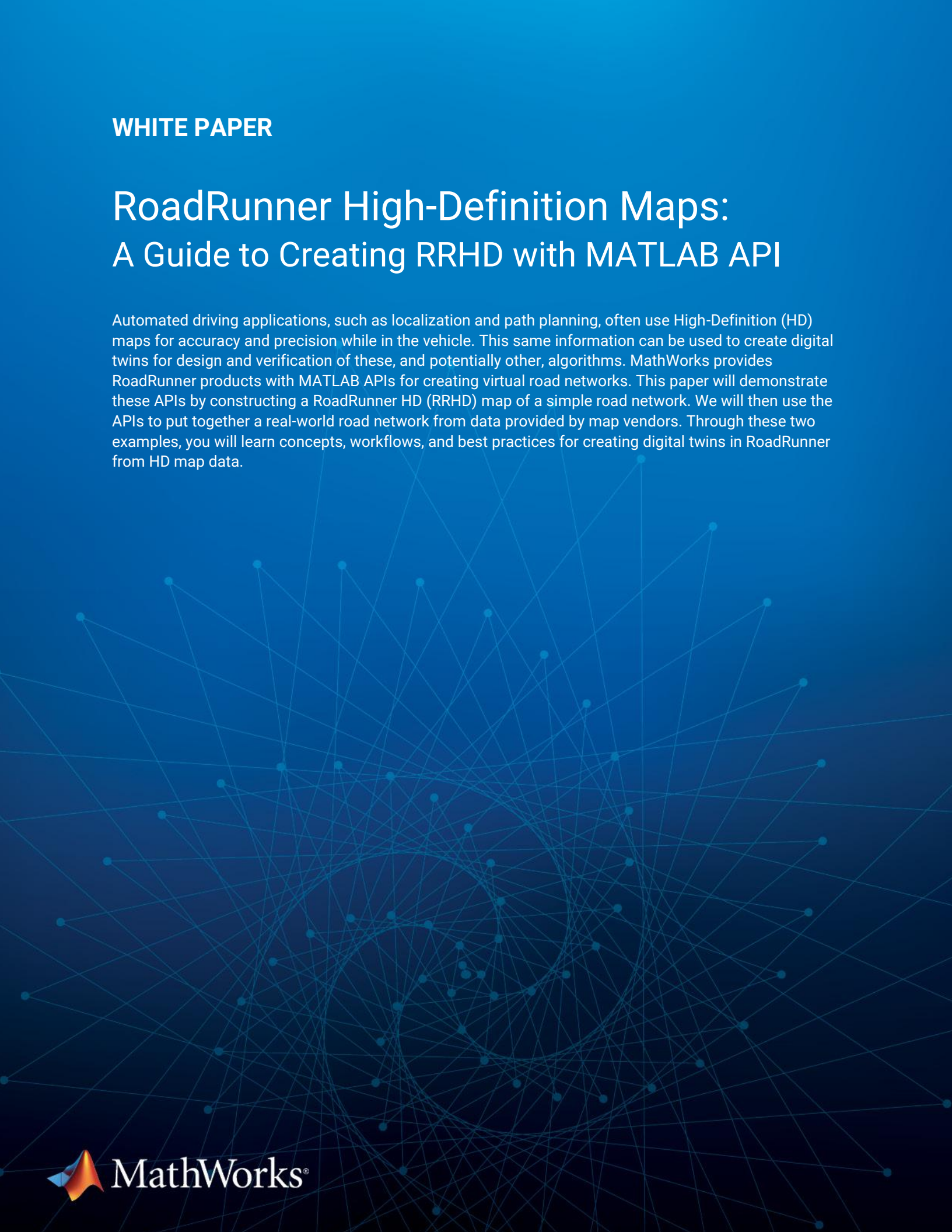
WHITE PAPER

RoadRunner High-Definition Maps:
A Guide to Creating RRHD with MATLAB API

Automated driving applications, such as localization and path planning, often use High-Definition (HD) maps for accuracy and precision while in the vehicle. This same information can be used to create digital twins for design and verification of these, and potentially other, algorithms. MathWorks provides RoadRunner products with MATLAB APIs for creating virtual road networks. This paper will demonstrate these APIs by constructing a RoadRunner HD (RRHD) map of a simple road network. We will then use the APIs to put together a real-world road network from data provided by map vendors. Through these two examples, you will learn concepts, workflows, and best practices for creating digital twins in RoadRunner from HD map data.



MathWorks®

Contents

Introduction

High-definition (HD) maps play an important role in Automated Driving. HD maps provide centimeter accuracy to help algorithms understand where the vehicle is in relation to known elements in the real world. Many localization and path planning algorithms rely on this data while the vehicle is in motion. At the same time, development of other vehicle algorithms often relies on simulation for testing and validation before deployment onto a vehicle in the real world. Using accurate, known, virtual environments in which to test allows algorithm development to advance more quickly than only testing in-situ. These digital twins can help developers find and fix problems faster and easier because the environment in which an incident has occurred, or an algorithm is known to struggle can be used repeatedly and reliably. Digital twins also allow engineers to test rare edge cases that may be too dangerous for real-world testing or are difficult to recreate in-situ.

HD maps contain most of the relevant information required to create accurate virtual representations of real-world road networks, including signs, traffic lights, buildings, foliage, and many other permanent stationary objects relevant to the roadway. However, many simulators do not directly ingest HD maps. Rather, they look to standards like OpenDRIVE or FBX to encode the information for their consumption.

MathWorks provides RoadRunner, an interactive editor, for scene and scenario generation. RoadRunner also provides an API for users to interact with it through the MATLAB language and Google's Remote Procedure Call (gRPC) with protocol buffers (protobuf). Users can quickly and easily create scenes and scenarios from scratch, or from existing sources such as custom HD maps, Standard Definition (SD) maps, satellite imagery, and sensor data from vehicle drives. RoadRunner also provides a means of generating OpenDRIVE, FBX, and other standard file formats from any scene. Therefore, the user can quickly and easily create digital twins for use in many available simulators.

Through MATLAB and gRPC protobuf APIs, users can create and manipulate an object called RoadRunner High-Definition (RRHD) map. When a user pulls in data from a map vendor such as TomTom, HERE-HD, Zenrin ZD, DMP, or MapMyInda, the information is encoded in the RRHD map object. If a user has sensor data from which they would like to recreate a scene, they can convert that information by using the RRHD map. If the user has a scene in RoadRunner and wants to enhance the scene with additional information on buildings, foliage, and other static objects obtained through other sources like aerial imagery or OpenStreetMaps, the information can be easily transformed to RRHD and merge with an an existing scene or road network. Additionally, RRHD maps can provide the necessary (ground truth) information for localization and path planning algorithms and V2X applications.

Often, the user will interact with the RoadRunner user interface to manually create a scene, whether from scratch or from existing data such as a supported HD map. However, the user may also elect to interact with RoadRunner through the provided APIs. For instance, the user may wish to create a scene and a large number of variations on that scene. Alternatively, the user may have access to data or map vendor for which RoadRunner does not support direct import.

This paper will discuss how to build up a RoadRunner High-Definition map using MATLAB APIs. We will provide a detailed example of building a roadway from scratch, highlighting the features for each of the RRHD map properties. Then, we will demonstrate how to create a digital twin for a segment of roadway from HD map data. This example assumes the user has programmatic access to all necessary data from the HD map of choice. Throughout the paper, we will address how to avoid common pitfalls and provide best practices.

Creating RRHD

RRHD is a data model created specifically to translate HD Map information into RoadRunner ingestible information. It codifies much of the static information available in HD Maps, but it does not accept dynamic information such as traffic and weather. The power of RRHD is that if the one has programmatic access to the information in an HD map, it is relatively straightforward to translate it and create a scene in RoadRunner.

Despite the fact there is no standard for defining HD maps, they have a few basic traits in common. They use the concept of connectivity through nodes and edges to define the underlying road graph network. This connectivity brings together the more detailed pieces of the road defined in other layers of the map. HD maps use the concept of geometry to precisely define the sizes and shapes of the roads, lanes, markings, signs, and objects they codify. Often, HD maps will bundle lanes together in a lane group. This helps define the roadway. Lane markings and borders are connected to lanes, and objects like signs, signals and barriers also connect to lanes or lane groups.

RRHD also utilizes these concepts. The biggest difference between RRHD and most HD maps is that RRHD omits the graph nodes and edges. Instead, RRHD defines connectivity through lanes via end-to-end and side-to-side connections. This removes a layer of complexity from the RRHD map. If the user wishes to create only a portion of a roadway, however, they will be best served by using the HD map graph network to extract the information they need before they create the RRHD map.

MATLAB provides methods to define the RoadRunner High-definition map. The RRHD map is encoded as a MATALB object with several properties that are also objects with their own properties. Many of these property objects share the same base properties. This section will describe these properties and how to effectively use them to create a simple RRHD map for a generic section of road.

We will first create a simple example roadway to demonstrate the various properties of the RRHD map. This road will contain five road segments (North, West, South, East, and North/South) with one junction created by roads crossing (between segments 4 and 5), and the other junction explicitly defined using the RRHD map (between 1, 2, 3, and 4).

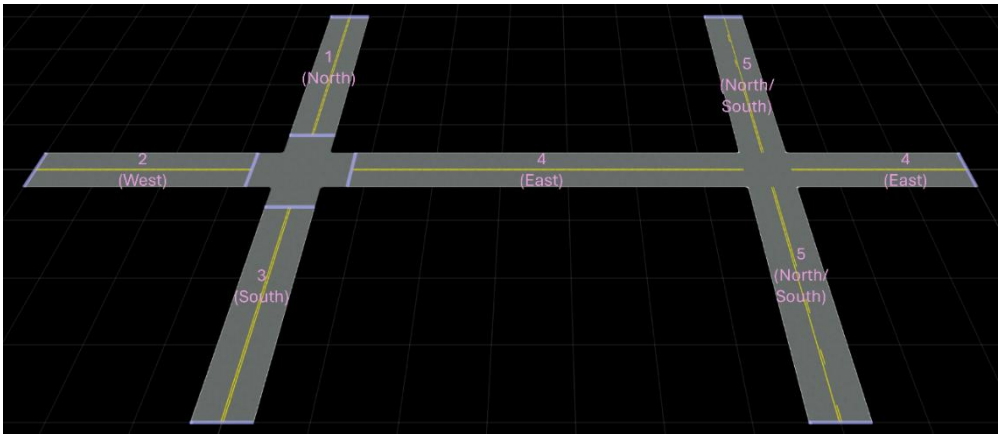


Figure 1: Simple roadway example to demonstrate the various features of the RRHD map.

Create an empty RRHD map

To create an empty RRHD map, call the **roadrunnerHDMap** function. You may provide any or all required information. We will only give an Author here. When working with data from a real-world map, you will

need to provide the GeoReference as the geographic origin of your map. This will be shown in the more detailed **Example: HD Map to RRHD**.

```
rrMap = roadrunnerHDMap(Author="kmcgarri")
```

```
Command Window
>> rrMap = roadrunnerHDMap(Author="kmcgarri")

rrMap =

  roadrunnerHDMap with properties:

                Author: "kmcgarri"
           GeoReference: [0 0]
    GeographicBoundary: []
                 Lanes: [0×1 roadrunner.hdmap.Lane]
           SpeedLimits: [0×1 roadrunner.hdmap.SpeedLimit]
       LaneBoundaries: [0×1 roadrunner.hdmap.LaneBoundary]
           LaneGroups: [0×1 roadrunner.hdmap.LaneGroup]
         LaneMarkings: [0×1 roadrunner.hdmap.LaneMarking]
            Junctions: [0×1 roadrunner.hdmap.Junction]
         BarrierTypes: [0×1 roadrunner.hdmap.BarrierType]
             Barriers: [0×1 roadrunner.hdmap.Barrier]
           SignTypes: [0×1 roadrunner.hdmap.SignType]
               Signs: [0×1 roadrunner.hdmap.Sign]
     StaticObjectTypes: [0×1 roadrunner.hdmap.StaticObjectType]
         StaticObjects: [0×1 roadrunner.hdmap.StaticObject]
   StencilMarkingTypes: [0×1 roadrunner.hdmap.StencilMarkingType]
      StencilMarkings: [0×1 roadrunner.hdmap.StencilMarking]
     CurveMarkingTypes: [0×1 roadrunner.hdmap.CurveMarkingType]
         CurveMarkings: [0×1 roadrunner.hdmap.CurveMarking]
           SignalTypes: [0×1 roadrunner.hdmap.SignalType]
               Signals: [0×1 roadrunner.hdmap.Signal]
```

Figure 2: RRHD map structure

References

An important concept for RRHD maps is the Reference. Many components of the RRHD map, (e.g., Lane Groups, Lanes, Lane Boundaries, Signs, Junctions, Barriers, Static Objects, Stencil Markings, Curved Markings, and Signals) have references to other objects created in the RRHD map. A reference object is the simplest object in the RRHD map. It contains the ID of the object being referenced.

Create a Sample Reference

As an example, assume we have created a lane object for the east bound lane in the lane group on the west side of a junction. The lane group needs the reference object to that lane.

```
EastBoundLaneRefW = roadrunner.hdmap.Reference(ID="LnGrW_EastBnd")
```

```
Command Window
>> EastBoundLaneRefW = roadrunner.hdmap.Reference(ID="LnGrW_EastBnd")

EastBoundLaneRefW =

  Reference with properties:

    ID: "LnGrW_EastBnd"
```

Figure 3: RRHD map Reference object structure

Even though there are no other properties, the ID is expected as a name, value pair. The IDs must be given as a string scalar (`"LnGrW_EastBnd", "546776"`) or character vector (`'LnGrW_EastBnd', '546776'`).

Pitfall

Every object lane group, lane, lane marking, sign, junction, barrier, etc.) must have a unique ID. If the IDs are not unique, RoadRunner will only create the first encountered instance of the object with a given ID in the RRHD map.

Recommendation

HD maps often provide unique IDs for each object. Use those IDs when automatically creating scenes from HD maps.

Aligned References

Another important concept for RRHD is an aligned reference. When creating lanes and lane groups, the logical and geometric alignment of these objects with respect to each other is important. Alignment is defined relative to the object receiving the reference. As will be shown below, the lane group object has a property called Lanes, which is an array of Aligned References referring to the lanes that make up the lane group.

Create a Sample Aligned Reference

Assume we now want to connect our east bound lane to the lane group. As will be discussed in **Lane Groups** the Lanes property contains a list of Aligned References to all the lanes belonging to the lane group. We will add the Aligned Reference to the lane group object **`LaneGroupW`**.

```
LaneGroupW.Lanes(end+1) =
roadrunner.hdmap.AlignedReference(Reference=EastBoundLaneRefW,
```

```
Command Window
>> roadrunner.hdmap.AlignedReference(Reference=EastBoundLaneRefW, Alignment="Forward")

ans =

  AlignedReference with properties:

    Reference: [1×1 roadrunner.hdmap.Reference]
    Alignment: Forward

>> LaneGroupW.Lanes(end)

ans =

  AlignedReference with properties:

    Reference: [1×1 roadrunner.hdmap.Reference]
    Alignment: Forward
```

Figure 4: RRHD AlignedReference object structure.

The Reference property receives the Reference object we created in the **References** section above. We defined the alignment property to be Forward because (as we'll see in the **Lanes** section below) we defined the lanes to have the same orientation as the lane group.

A Note on Alignment

The orientation of a lane or lane group is defined by the geometry of the object. Geometry is defined as an array of (x, y, z) coordinates. As we will see in the **Lane Lane** Groups section, we can define the lane group geometry to run, e.g., west to east (left to right along the x-axis) or we can define it to run east to west (right to left along the x-axis). If we elect to align the lane group west to east, we could define the geometry as [-50, 0; 50, 0]. If we elect to align the lane group east to west, we could define the geometry as [50, 0; -50, 0]. For the right lane within the lane group to be `"Forward"` aligned we would make sure the x-coordinates match those of the lane group: [-50, -1.8; 50, -1.8] for west-to-east lane group orientation, or [50, 1.8; -50, 1.8] for east to west. *(Note: The concept of right (and left) is defined with respect to the orientation of the lane group.)* If the orientation cannot be defined as `"Forward"`, it must be defined as `"Backward"`. See **Figure 5** for an example of forward and backward alignment.

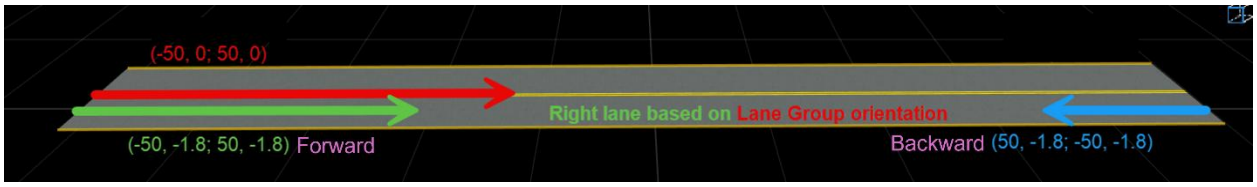


Figure 5: Examples of lane orientations with respect to the lane group. The red arrow shows the lane group orientation if we defined it west to east. The green arrow shows the orientation of the right lane if we defined it running west to east – it would receive the "Forward" designation. The blue arrow shows the orientation of the right lane if we define it as running east to west – it would receive the "Backward" designation with respect to the lane group.

Pitfall

Not providing the proper alignment will lead to strange renderings of the roadway.

Recommendation

If the final result does not look right, start by checking the alignment between lanes and lane groups, and between lanes and lane markings.

Lane Groups

High-definition maps often model roads as a collection of lane groups with connections between them. Lane groups are often defined as a single object, using the concept of geometry to define the location of the center of the lane group. The type of road being modelled will often determine whether lane groups consist of lanes with travel allowed in both directions (Lane Groups 1, 2 and 3 in **Figure 6**), or only in a single direction (Lane Group 4 in **Figure 6**). Single direction lane groups are used most frequently for roads with physical dividers between lanes of a given direction, such as highways and interstates. Roads without a physical divider will often have a single lane group for all lanes with travel allowed in both directions. The Lane Group object in RRHD references the lanes that create the group.

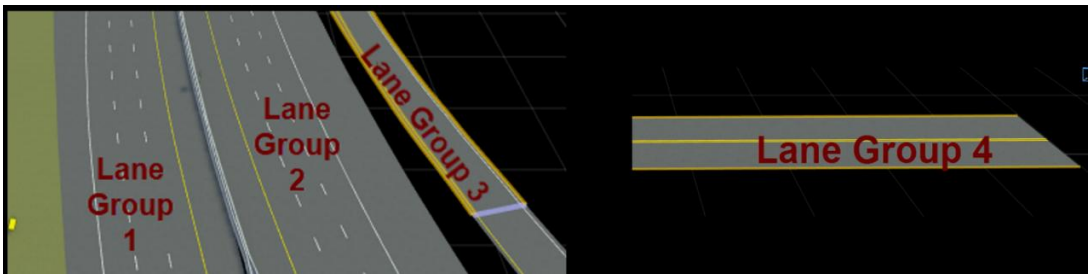


Figure 6: Lane groups for single-direction travel (Lane Groups 1, 2, and 3), and lane groups for bi-directional travel (Lane Group 4).

Create a Simple Lane Group

The snippet of MATLAB code below shows how to define a RRHD lane group object. In this example, we will create a short segment of road that runs "West to East" or left to right along the x-axis. Lane Group geometry may be defined as a 2D element vector (x, y) or as a 3D element vector (x, y, z). If only the x and y coordinates are provided, z is assumed to be zero. All inputs to the Lane Group method are provided as name, value pairs.

```
LaneGroupW = roadrunner.hdmap.LaneGroup(Geometry=[-40, 0; -7.5, 0],
ID="LnGrW")
```

```
Command Window
>> LaneGroupW = roadrunner.hdmap.LaneGroup(Geometry=[-40, 0; -7.5, 0], ID="LnGrW")

LaneGroupW =

  LaneGroup with properties:

         ID: "LnGrW"
   Geometry: [2×2 double]
      Lanes: [0×1 roadrunner.hdmap.AlignedReference]
```

Figure 7: Example of RRHD LaneGroup object.

Lane Groups have one more property besides Geometry and ID. They have the Lane property, which is an array of aligned references to the lanes within the lane group, described in section **Aligned References**.

For our example lane group, we will define two lanes, an east bound lane and a west bound lane. (We will create these lanes in **Lanes**.) Assuming we have already created these lanes and defined their References (EastBoundLaneRefW and WestBoundLaneRefW), the following code shows how to add the aligned references to those lanes to the Lane Group. HD maps will almost always align the lanes and the lane groups in the same direction, though this is not always the case. In this example, we also define the lanes in the same direction as the lane group.

```
LaneGroupW.Lanes(end+1) =
roadrunner.hdmap.AlignedReference(Reference=EastBoundLaneRefW,
Alignment="Forward")
LaneGroupW.Lanes(end+1) =
roadrunner.hdmap.AlignedReference(Reference=WestBoundLaneRefW,
Alignment="Forward")
```

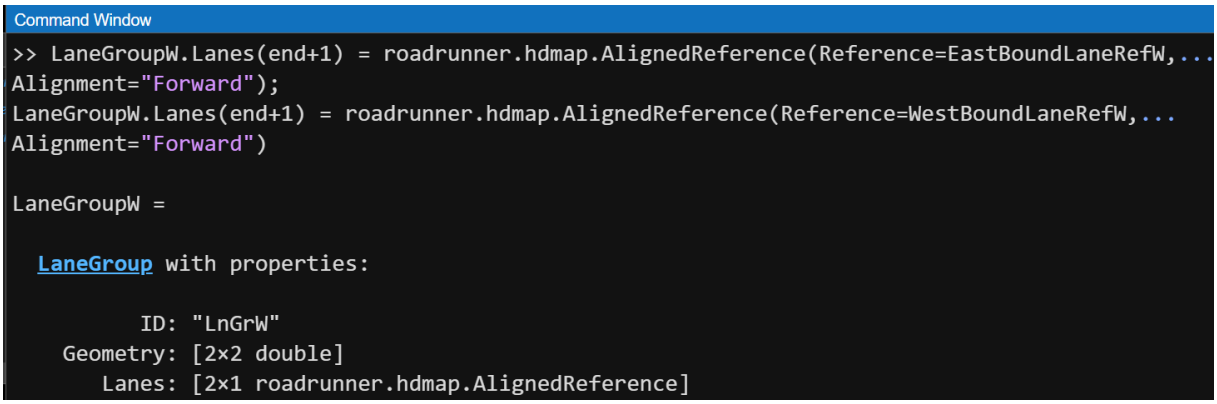


*Figure 8: Note we have now added two AlignedReference objects to the Lanes property in the RRHD LaneGroup property. (Compare to **Figure 7**.)*

Once the lane group has been fully assembled, add it to the RRHD map.

```
rrMap.LaneGroups(end+1) = LaneGroupWE
```

Note: MATLAB best practice is to pre-allocate memory for arrays. Because we may not know a priori how many objects of interest we will have, especially if we are creating a real-world road from another HD map, we will instead allow the array to grow as we build it up.

Lanes

In RRHD, lane objects represent many different parts of the roadway. Aside from representing the portion of the road in which vehicles move, they also represent the shoulders, bike lanes, emergency stopping, medians, curbs, sidewalks, street-side parking, road boarders, entrance and exit ramps, turning lanes, and restricted lanes. For this reason, lanes are the most complex and information dense portion of the RRHD map.

The Lanes object defines a lane through its geometry, boundaries, type and travel direction, references its markings, connects to other lanes, allows users to define metadata, and contains parametric data

such as speed limits. In many ways, Lanes can be thought of as the most fundamental part of the RRHD map. Without lanes, no part of the road network will render in the RoadRunner canvas when the map is imported.

Defining connections between lanes helps with proper rendering of the roadway. RRHD considers two types of lane connections. Lanes within a lane group are connected through their boundaries. This connection is a side-to-side connection. RRHD uses the language of Left and Right, but the definitions of Left and Right depend on the orientation of the lane and the lane group rather than the direction of travel. Lanes in different lane groups are connected through the concept of predecessors and successors. Whether a lane precedes or succeeds another lane depends on each lane's alignment. Regardless of travel direction or global alignment, a lane is a predecessor if it attaches at the starting point, as defined by the geometry, of the lane of interest. A lane is a successor if it attaches at the end point.

Create a Simple Lane

Lane geometry and a unique ID are the bare minimum requirements to define a lane. The lane geometry is given relative to the center of the lane. However, only providing the lane center will not define the lane's width. That is done by defining the **Lane** Boundaries. Travel direction, lane type, and speed limit, while not required, have implications when using the RRHD map to create scenarios. This information might be used by algorithms to determine whether a vehicle is illegally traveling in a lane, either by entering a lane not designated for travel or by driving against the travel direction. Additionally, this information might be used by the simulator when setting up the scenario to help define a vehicle's trajectory.

We will start by defining the east bound lane for the lane group defined in **Create a Simple Lane** Group. The East bound lane's travel direction aligns with the orientation of the road.

```
EastBoundLaneW = roadrunner.hdmap.Lane(Geometry=[-40,-1.8;-7.5,-1.8],
ID="LnGrW_EastBnd", LaneType="Driving", TravelDirection="Forward")
```

```
Command Window
>> EastBoundLaneW = roadrunner.hdmap.Lane(Geometry=[-40, -1.8; -7.5, -1.8],...
 ID="LnGrW_EastBnd", LaneType="Driving", TravelDirection="Forward")

EastBoundLaneW =

  Lane with properties:

                      ID: "LnGrW_EastBnd"
                Geometry: [2×2 double]
           TravelDirection: Forward
         LeftLaneBoundary: [0×0 roadrunner.hdmap.AlignedReference]
        RightLaneBoundary: [0×0 roadrunner.hdmap.AlignedReference]
            Predecessors: [0×1 roadrunner.hdmap.AlignedReference]
              Successors: [0×1 roadrunner.hdmap.AlignedReference]
                LaneType: Driving
                Metadata: [0×1 roadrunner.hdmap.Metadata]
     ParametricAttributes: [0×1 roadrunner.hdmap.ParametricAttribution]
```

Figure 9: Example of RRHD map Lane object. Note only the information passed into the method call has been assigned.

At this point, the lane group has only one lane. We can add other lanes, and for this example, we will add a west bound lane to make the lane group a two-way surface street. For this lane, we define the direction of travel as backward because it goes against the orientation of the lane, which is still defined as west to east.

```
WestBoundLaneW = roadrunner.hdmap.Lane(Geometry=[-40,1.8;-7.5,1.8],
ID="LnGrW_WestBnd", LaneType="Driving", TravelDirection="Backward")
```

```
Command Window
>> WestBoundLaneW = roadrunner.hdmap.Lane(Geometry=[-40, 1.8; -7.5, 1.8],...
ID="LnGrW_WestBnd", LaneType="Driving", TravelDirection="Backward")

WestBoundLaneW =

  Lane with properties:

                     ID: "LnGrW_WestBnd"
               Geometry: [2×2 double]
        TravelDirection: Backward
       LeftLaneBoundary: [0×0 roadrunner.hdmap.AlignedReference]
      RightLaneBoundary: [0×0 roadrunner.hdmap.AlignedReference]
           Predecessors: [0×1 roadrunner.hdmap.AlignedReference]
             Successors: [0×1 roadrunner.hdmap.AlignedReference]
               LaneType: Driving
               Metadata: [0×1 roadrunner.hdmap.Metadata]
    ParametricAttributes: [0×1 roadrunner.hdmap.ParametricAttribution]
```

Figure 10: The second lane. Note, there are no connections or boundaries assigned yet.

So far, the lanes have no widths nor any connections. We need to define the lane boundaries for both lanes. In general, there will be N+1 lane boundaries needed for N lanes defined in a lane group. In this case, there are 2 lanes, so we need 3 boundaries: a center boundary shared by both lanes, an east-bound side boundary, and a west-bound side boundary. Assuming we have created the LaneBoundary objects for these three boundaries (see Create a Simple Lane Boundary) we can use the convenience methods for the Lanes object to define the left and right boundaries of each lane. *Note: the center boundary is shared between the two lanes.*

Pitfall

Left and right are determined by the orientation of the lane geometry, NOT relative to the driving direction.

For example, the center boundary is the LEFT boundary for the east-bound lane, which is how a driver in this lane would define it. However, the center boundary is the RIGHT boundary for the west-bound lane, even though a driver would see this as the left boundary for this lane. If we were to define the lane orientations to be in the opposite direction, the definitions of left and right for each boundary would switch.

```
EastBoundLaneW.leftBoundary("CenterLineW", Alignment="Forward")
WestBoundLaneW.rightBoundary("CenterLineW", Alignment="Forward")
EastBoundLaneW.rightBoundary("EastBoundSideLine", Alignment="Forward")
WestBoundLaneW.leftBoundary("WestBoundSideLine", Alignment="Forward")
```

Command Window

```
>> EastBoundLaneW.leftBoundary("CenterLineW", Alignment="Forward");
WestBoundLaneW.rightBoundary("CenterLineW", Alignment="Forward");
EastBoundLaneW.rightBoundary("EastBoundSideLine", Alignment="Forward");
WestBoundLaneW.leftBoundary("WestBoundSideLine", Alignment="Forward");
>> EastBoundLaneW

EastBoundLaneW =

  Lane with properties:

                     ID: "LnGrW_EastBnd"
               Geometry: [2×2 double]
         TravelDirection: Forward
        LeftLaneBoundary: [1×1 roadrunner.hdmap.AlignedReference]
       RightLaneBoundary: [1×1 roadrunner.hdmap.AlignedReference]
           Predecessors: [0×1 roadrunner.hdmap.AlignedReference]
             Successors: [0×1 roadrunner.hdmap.AlignedReference]
               LaneType: Driving
               Metadata: [0×1 roadrunner.hdmap.Metadata]
    ParametricAttributes: [0×1 roadrunner.hdmap.ParametricAttribution]
>> WestBoundLaneW

WestBoundLaneW =

  Lane with properties:

                     ID: "LnGrW_WestBnd"
               Geometry: [2×2 double]
         TravelDirection: Backward
        LeftLaneBoundary: [1×1 roadrunner.hdmap.AlignedReference]
       RightLaneBoundary: [1×1 roadrunner.hdmap.AlignedReference]
           Predecessors: [0×1 roadrunner.hdmap.AlignedReference]
             Successors: [0×1 roadrunner.hdmap.AlignedReference]
               LaneType: Driving
               Metadata: [0×1 roadrunner.hdmap.Metadata]
    ParametricAttributes: [0×1 roadrunner.hdmap.ParametricAttribution]
```

Figure 11: Both lanes each have a left and right boundary.

If there were more lane groups with lanes, we would look into linking the lane groups together using addSuccessor and addPredecessor convenience methods for the Lanes object. We will cover this in detail when we go through a more complex example in **Example: HD Map to RRHD**.

Should we want to include a speed limit, we can add a reference to the speed limit object in the Parametric Attributes object of the Lanes object. We will discuss this object further in the **Parametric Attributes** section.

Finally, you may add other information to the lane through the metadata object. We will discuss this object further in the **Error! Reference Source not Found** section.

As the last step, add the finished lanes to the RRHD map.

```
rrMap.Lanes(end+1) = EastBoundLaneWE
rrMap.Lanes(end+1) = WestBoundLaneWE
```

Lane Boundaries

As discussed in the **Lanes** section, lane widths and lane connections within a lane group are defined through references to Lane Boundaries objects. Lane Boundaries require a unique ID and geometry. If lane markings exist at the boundaries, they can be added using the Parametric Attributes object, which will be discussed further in the **Parametric Attributes** section.

Create a Simple Lane Boundary

For our current lane group, we have two lanes. They will share the center lane boundary, and each will have its own side boundary. We will create lane boundary objects here. Assuming the lane marking references have been created, we can add the lane marking references to the parametric attributes property of the boundary.

```
cntrLaneBndry = roadrunner.hdmap.LaneBoundary(ID="CenterLineW",
Geometry=[-40, 0; -7.5, 0])
cntrLaneBndry.ParametricAttributes =
roadrunner.hdmap.ParametricAttribution(Span=[0, 1],
MarkingReference=sdyMarkRef)


ebsLaneBndry = roadrunner.hdmap.LaneBoundary(ID="EastBoundSideLine",
Geometry=[-40, -3.6; -7.5, -3.6])
ebsLaneBndry.ParametricAttributes =
roadrunner.hdmap.ParametricAttribution(Span=[0, 1],
MarkingReference=sswMarkRef)


wbsLaneBndry = roadrunner.hdmap.LaneBoundary(ID="WestBoundSideLine",
Geometry=[-40, 3.6; -7.5, 3.6])
wbsLaneBndry.ParametricAttributes =
roadrunner.hdmap.ParametricAttribution(Span=[0, 1],
MarkingReference=sswMarkRef)
```

```
Command Window
>> wbsLaneBndry = roadrunner.hdmap.LaneBoundary(ID="WestBoundSideLine",...
Geometry=[-40, 3.6; -7.5, 3.6]);
wbsLaneBndry.ParametricAttributes = roadrunner.hdmap.ParametricAttribution(Span=[0, 1],...
MarkingReference=sswMarkRef)

wbsLaneBndry =

  LaneBoundary with properties:

                      ID: "WestBoundSideLine"
                Geometry: [2×2 double]
    ParametricAttributes: [1×1 roadrunner.hdmap.ParametricAttribution]
```

Figure 12: RRHD LaneBoundary object.

Now that the boundaries exist, the Lanes objects can refer to them for connecting lanes within the lane group, defining the width of the lanes, and adding lane markings to the road.

Once the lane boundaries are finalized, we add them to the RRHD map.

```
rrMap.LaneBoundaries(end+1) = cntrLaneBndry
rrMap.LaneBoundaries(end+1) = ebsLaneBndry
rrMap.LaneBoundaries(end+1) = WbsLaneBndry
```

Parametric Attributes

Parametric Attributes objects hold information related to lanes and lane boundaries that utilize the concept of location along the lane or boundary, also called span. Lane Markings, Speed Limits and Traffic Signals all fall under this category. Regardless of which attribute is defined, the basic definition of this object is the same.

Most often, Parametric Attributes will span the entire length of a lane or lane boundary. For example, the lane marking between lanes with opposite travel directions will likely have dashed yellow the entire length of the lane group. There are times, however, when this might not hold. For example, on curvy or hilly roads, the center lane markings might change from dashed to solid-dashed or dashed-solid several times along the length of the lane boundary. This is when the concept of span comes into play.

The Span property is defined as a fraction of length along the lane or boundary. Therefore, The Span start should never be less than zero, and the span end should never be greater than one.

Pitfall

While you can define multiple speed limits along a lane, RoadRunner will only apply the fastest limit to the lane.

Recommendation

If you need to define multiple speed limits, break the lane (and corresponding lane group) into separate objects for each segment and provide the speed limits for each object.

Creating a Parametric Attribute

We already demonstrated how to define a lane marking along the lane boundaries in **Create a Simple Lane Boundary**. For that example, the marking spans the entire length of the boundary. If instead we want to define a double solid yellow center line for a first segment of the lane boundary, a solid-dashed yellow center line for a second segment, and a dashed-solid yellow center line for the last segment, assuming the references to those three lane markings exist, we would do the following.

```
cntrLaneBndry.ParametricAttributes(end+1) =
roadrunner.hdmap.ParametricAttribution(Span=[0, 0.3],
MarkingReference=doubleSolidYellowMarkRef)

cntrLaneBndry.ParametricAttributes(end+1) =
roadrunner.hdmap.ParametricAttribution(Span=[0.3, 0.72],
MarkingReference=SolidDashedYellowMarkRef)

cntrLaneBndry.ParametricAttributes(end+1) =
roadrunner.hdmap.ParametricAttribution(Span=[0.72, 1],
MarkingReference=DashedSolidYellowMarkRef)
```

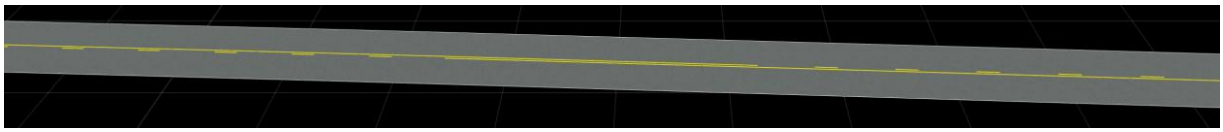


Figure 13: Note the lane marking along this stretch of road has been defined in three segments: Solid-Dashed, Double Solid, Dashed-Solid.

Pitfall

RoadRunner does not enforce the span end for attribute n to match the start of attribute n+1. Any gaps will have no markings present. E.g., if span 1 ended at 0.3 and span 2 started at 0.36, the distance between 0.3 and 0.36 along the road would have no markings of any kind.

Recommendation

When creating a script, warn if the sum is not continuous.

To add speed limits or Signals along a roadway, we would follow the same process calling the `SpeedLimitReference` or `SignalReference` name, value pair.

Metadata

The Metadata object allows the user to attach any information to a lane that is not already defined by RRHD. This information is useful for many things, including exporting OpenDRIVE user data and providing information relevant to ADAS or Automated Driving algorithms. For example, an algorithm identifies the lanes using a numbering system counting left to right along the direction of travel. Each lane in RRHD must have a unique identifier, so the lane number cannot be used as the ID. Therefore, we can give the Lane Number as metadata. Now, during testing, the lane number can be obtained at any point along the trajectory and compared to the value the algorithm selected. The metadata of an object can be found in the side panel Metadata tab in the RoadRunner application when the object (lane, barrier, sign, etc.) is selected.

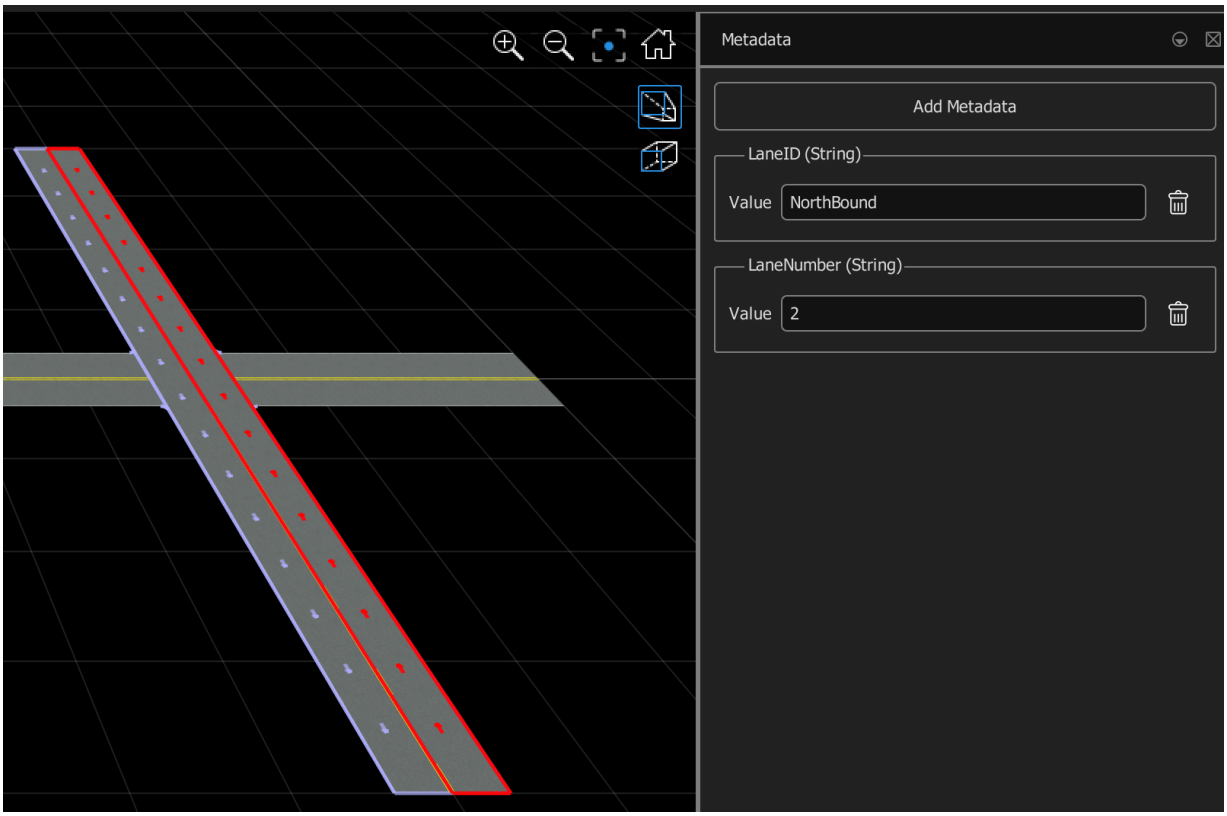


Figure 14: Metadata information display for a lane in RoadRunner app.

Create a Simple Metadata

```
LaneNum1 = roadrunner.hdmap.Metadata(Name="LaneNumber", Value="1")
```

```
Command Window
>> LaneNum1 = roadrunner.hdmap.Metadata(Name="LaneNumber", Value="1")

LaneNum1 =

  Metadata with properties:

      Name: "LaneNumber"
     Value: "1"
```

Figure 15: RRHD metadata object.

Pitfall

For Metadata both Value and Name must be specified as a string scalar or char vector. Value cannot take on numerical values.

Junctions

The final portion of the roadway that can be defined using RRHD is a junction. In general, RoadRunner does not need junctions to be independently defined. When two lane groups cross one another, RoadRunner will form a junction with standard geometry based on the angles of the lane groups with respect to one another.

If, for some reason, RoadRunner does not create the junction in a reasonable way, you may use the Junctions object to define a junction. Junctions are the most complicated object in the RRHD map. As shown in the next section, the onus is on the user to define every aspect of the junction.

Creating a Simple Junction

In this example, we will add a junction to the end of our lane group. It will represent the junction created by a road running west to east intersecting a road running north to south. The Junction breaks these two roads into four, a road on the west, a road on the east, a road to the north, and a road to the south of the junction. Each road has two lanes, with opposing travel directions, and three lane markings. We assume all of this has been set up per the examples above. Our final result will look something like this:

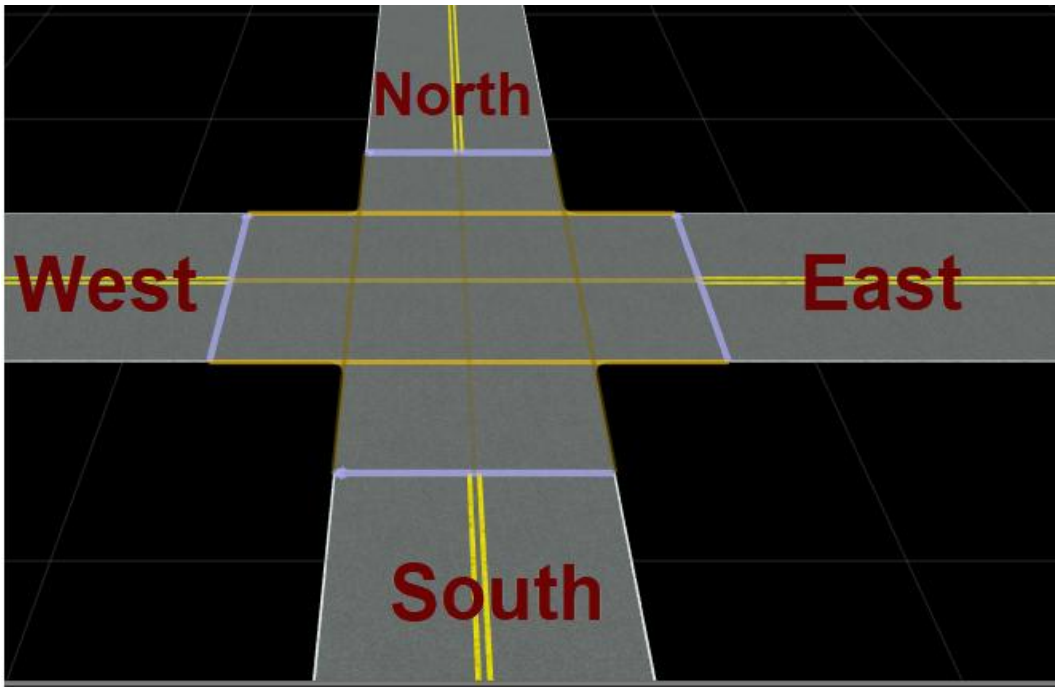


Figure 16: Road definition used to create the junction object.

Junction geometry is defined using a special RRHD MultiPolygon object. A MultiPolygon object has a RRHD Polygons object as its property. The Polygons object has two properties. One is an ExteriorRing, the other is InteriorRIngs. ExteriorRing defines the outer geometry of the junction as an Nx3 array, like any other geometry definition described in the previous sections. InteriorRings defines inner geometry that should be removed from the junction. Because there might be multiple interior rings, this property should be defined as a cell array where each cell contains an Nx3 array defining the geometry of the interior ring. For example, creating a roundabout would require the exterior ring to define the outermost edge of the roundabout while the interior ring defines the inner edge of the roundabout.

For our simple junction, we will only define the exterior ring.

```
junctGeom = [-3.6,  7.5, 0; -5.0,  5.0, 0; -7.5,  3.6, 0; ...
             -7.5, -3.6, 0; -5.0, -5.0, 0; -3.6, -7.5, 0; ...
              3.6, -7.5, 0;  5.0, -5.0, 0;  7.5, -3.6, 0; ...
              7.5,  3.6, 0;  5.0,  5.0, 0;  3.6,  7.5, 0; ...
             -3.6,  7.5, 0]
junctPolygn = roadrunner.hdmap.Polygon(ExteriorRing=junctGeom)
junctMultiPolygn = roadrunner.hdmap.MultiPolygon(polygons=junctPolygn)
centerJunct = roadrunner.hdmap.Junction(Geometry=junctMultiPolygn,
ID="TestJunction")
```

```
Command Window
>> junctGeom = [-3.6,  7.5, 0; -5.0,  5.0, 0; ...
            -7.5,  3.6, 0; -7.5, -3.6, 0; ...
            -5.0, -5.0, 0; -3.6, -7.5, 0; ...
             3.6, -7.5, 0;  5.0, -5.0, 0; ...
             7.5, -3.6, 0;  7.5,  3.6, 0; ...
             5.0,  5.0, 0;  3.6,  7.5, 0; ...
            -3.6,  7.5, 0];
junctPolygn = roadrunner.hdmap.Polygon(ExteriorRing=junctGeom);
junctMultiPolygn = roadrunner.hdmap.MultiPolygon(polygons=junctPolygn);
centerJunct = roadrunner.hdmap.Junction(Geometry=junctMultiPolygn, ID="TestJunction")

centerJunct =

  Junction with properties:

               ID: "TestJunction"
         Geometry: [1×1 roadrunner.hdmap.MultiPolygon]
            Lanes: [0×1 roadrunner.hdmap.Reference]
   Configurations: [0×1 roadrunner.hdmap.JunctionConfiguration]
```

Figure 17: RRHD junction object.

```
>> centerJunct.Geometry

ans =

  MultiPolygon with properties:

    Polygons: [1×1 roadrunner.hdmap.Polygon]

>> centerJunct.Geometry.Polygons

ans =

  Polygon with properties:

     ExteriorRing: [13×3 double]
    InteriorRings: {}
```

Figure 18: RRHD MultiPolygon and Polygon objects used to define Junction geometry.

Next, the user must define all the connection lanes inside the junction. For our simple junction there are twelve total connecting lanes we must create: N2N, S2S, W2W, E2E, N2E, N2W, S2E, S2W, W2S, W2N,

E2S, E2N. It is also important to define the curvature of the lanes. RoadRunner does not automatically calculate the curvature given a junction geometry.

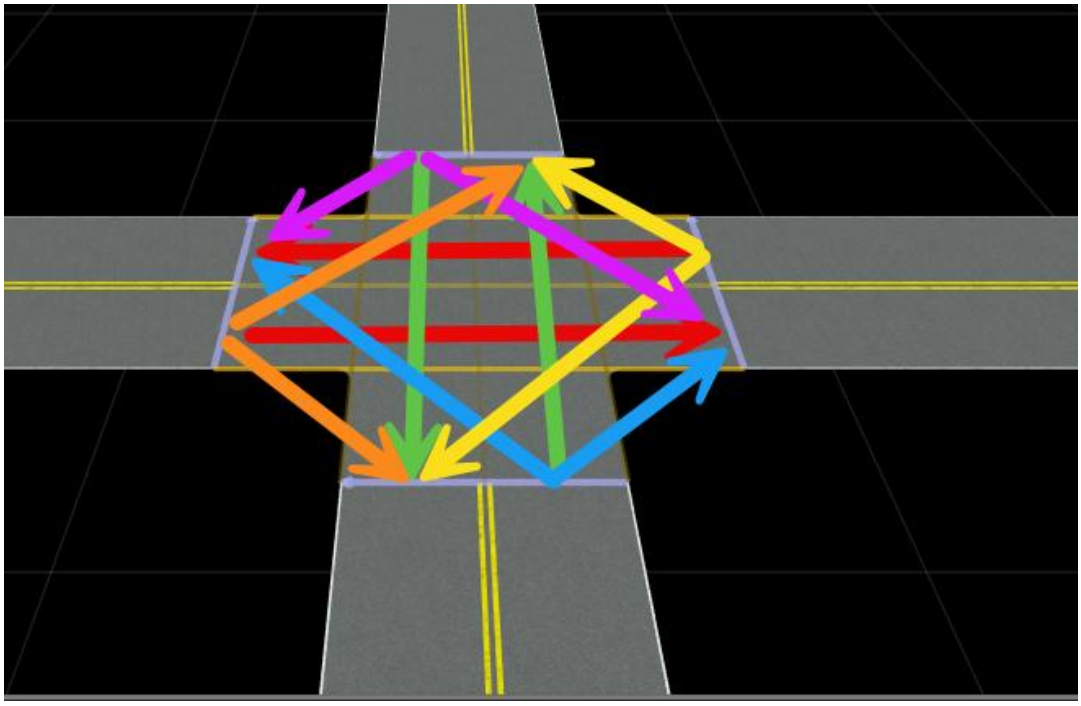


Figure 19: All Junction object lane connections. The user must define each of these lane connections individually.

Assuming we have defined all the lanes per the discussion in the **Lane** section, we can add the lane references to the junction. For sake of brevity, we only add four lanes here, but the other eight will be added in exactly the same way.

```
centerJunct.Lanes(end+1) = LaneN2NBndRef
centerJunct.Lanes(end+1) = LaneS2SBndRef
centerJunct.Lanes(end+1) = LaneW2WBndRef
centerJunct.Lanes(end+1) = LaneE2EBndRef
```

```
centerJunct =

  Junction with properties:

                ID: "TestJunction"
          Geometry: [1×1 roadrunner.hdmap.MultiPolygon]
             Lanes: [4×1 roadrunner.hdmap.Reference]
    Configurations: [0×1 roadrunner.hdmap.JunctionConfiguration]
```

Figure 20: RRHD Junction object with inner lane connections defined.

Junction Configurations

The final property of a junction is the Junction Configuration object. This object provides information about which lanes through a junction are allowed which actions during a signal phase (e.g. red light, green light, flashing red, etc.). The signal phase is defined using a Phase Object. For each phase, the allowed motion for each lane in the junction is defined in a Junction Lane State object.

For example, if there is a traffic light controlling each entrance into the junction we created in the section above, we can define the phases for the light controlling the west and east bound traffic.

```
EWTrafficLight =
roadrunner.hdmap.JunctionConfiguration(ID="Junction1EWLight",
Name="EastWestTrafficLight")
```

```
Command Window
>> EWTrafficLight = roadrunner.hdmap.JunctionConfiguration(ID="Junction1EWLight",...
Name="EastWestTrafficLight")

EWTrafficLight =

  JunctionConfiguration with properties:

        ID: "Junction1EWLight"
      Name: "EastWestTrafficLight"
    Phases: [0×1 roadrunner.hdmap.Phase]
```

Figure 21: RRHD JunctionConfiguration object.

```
EWRedPhase    = roadrunner.hdmap.Phase(ID="EWRedLight", Time=20)
EWYellowPhase = roadrunner.hdmap.Phase(ID="EWYellowLight", Time=5)
EWGreenPhase  = roadrunner.hdmap.Phase(ID="EWGreenLight", Time=15)
```

```
Command Window
>> EWRedPhase = roadrunner.hdmap.Phase(ID="EWRedLight", Time=20);
EWYellowPhase = roadrunner.hdmap.Phase(ID="EWYellowLight", Time=5);
EWGreenPhase = roadrunner.hdmap.Phase(ID="EWGreenLight", Time=15)

EWGreenPhase =

  Phase with properties:

                  ID: "EWGreenLight"
                Name: ""
                Time: 15
    JunctionLaneStates: [0×1 roadrunner.hdmap.JunctionLaneState]
```

Figure 22: RRHD Phase object.

```
EWGreenPhase.JunctionLaneStates(end+1) =
roadrunner.hdmap.JunctionLaneState(LaneID=LaneE2EBndRef,
State="GoAlways")
EWGreenPhase.JunctionLaneStates(end+1)=
roadrunner.hdmap.JunctionLaneState(LaneID=LaneE2EBndRef,
State="GoAlways") EWGreenPhase.JunctionLaneStates(end+1)=
roadrunner.hdmap.JunctionLaneState(LaneID=LaneE2EBndRef, State="Yield")
EWGreenPhase.JunctionLaneStates(end+1)=
roadrunner.hdmap.JunctionLaneState(LaneID=LaneN2NBndRef, State="Stop")
```

```
EWGreenPhase =

  Phase with properties:

                  ID: "EWGreenLight"
                Name: ""
                Time: 15
    JunctionLaneStates: [4×1 roadrunner.hdmap.JunctionLaneState]
```

Figure 23: RRHD Phase object with JunctionLaneStates defined.

```
JunctionLaneState with properties:

   LaneID: [1×1 roadrunner.hdmap.Reference]
    State: "GoAlways"
```

Figure 24: RRHD JunctionLaneState object.

```
EWTrafficLight.Phases(end+1) = EWRedPhase
EWTrafficLight.Phases(end+1) = EWGreenPhase
EWTrafficLight.Phases(end+1) = EWYellowPhase
```

```
EWTrafficLight =

  JunctionConfiguration with properties:

        ID: "Junction1EWLight"
      Name: "EastWestTrafficLight"
    Phases: [3×1 roadrunner.hdmap.Phase]
```

Figure 25: RRHD JunctionConfiguration object with defined Phases.

In this example, we have not considered all possible combinations of phase and lane states, but it is mostly a matter of good bookkeeping to satisfy the combinatorics.

Once the junction is complete, add it to the RRHD map.

```
rrMap.Junctions = centerJunct
```

Lane Markings and Other Asset Types

RRHD has properties that refer to RoadRunner Assets.

Road markings are categorized as Lane Markings, Stencil Marking Types, and Curved Marking Types. Lane Markings refer to an asset in RoadRunner that represents the marking between lanes on the road. Curved marking types also refer to assets in RoadRunner that represent markings along the road. The difference between these two marking types is that Lane Markings automatically follow the geometry of the lane boundary that refers to the marking. Curved markings are independent of lane boundaries. A typical use case for curved markings is parking spot marking. RRHD does not have a parking lot or parking spot object, but parking lots can be created using lanes and curved markings. Stencil marking types refer to other assets in RoadRunner that represent roadway markings such as zebra crossings, exclusion zone diagonals and chevrons, turn arrows painted on lanes, markings at an interstate split, etc.

Static objects are categorized as Barrier Types, Sign Types, and Static Object Types. These objects refer to assets in RoadRunner that represent the different kinds of barriers erected along the roadway, the signs encountered along the road, and any other static objects one might expect to find on the side or in the middle of the road. This includes Jersey Barriers, Guard rails, Noise Walls, Stop signs, Freeway signs Speed Limit signs, buildings, trees, traffic cones, etc.

Creating a Simple Asset Type

Regardless of what kind of asset type you wish to define, all are defined in exactly the same way. Start by defining a Relative Asset Path object, then create the asset type.

For a solid double yellow lane marking, we would do the following:

```
sdyPath =
roadrunner.hdmap.RelativeAssetPath(AssetPath="Assets/Markings/SolidDouble
Yellow.rrlms")
solidDbleYllw = roadrunner.hdmap.LaneMarking(ID="SolidDoubleYellow",
AssetPath=sdyPath)
```

```
Command Window
>> sdyPath = roadrunner.hdmap.RelativeAssetPath(AssetPath="Assets/Markings/SolidDoubleYellow.rrlms")
solidDbleYllw = roadrunner.hdmap.LaneMarking(ID="SolidDoubleYellow", AssetPath=sdyPath)

solidDbleYllw =

  LaneMarking with properties:

          ID: "SolidDoubleYellow"
   AssetPath: [1×1 roadrunner.hdmap.RelativeAssetPath]
```

Figure 26: RRHD LaneMarking object.

Pitfall

For Lane Markings, Sign Types, Static Object Types, Stencil Marking Types, and Curved Marking Types, the property is AssetPath. For Barriers, the property is ExtrusionPath, even though it expects the same input data.

Once the lane marking, or other Asset Type has been created, add it, as appropriate, to the RRHD map.

```
rrMap.LaneMarkings(end+1) = solidDbleYllw
```

Note About Relative Asset Paths

Relative Asset Paths are paths relative to your RoadRunner project directory leading to the asset you care about. In the above example, we made a solid double yellow lane marking. No matter what kind of asset you are looking for, you can find them in one of two ways. In the RoadRunner app find the "Library Browser" pane (see **Figure 27**). There you will find the folder structure for all assets in the RoadRunner project. Drill into the structure to find the asset you want. In the Attributes pane, the asset and the file name will appear in the preview.

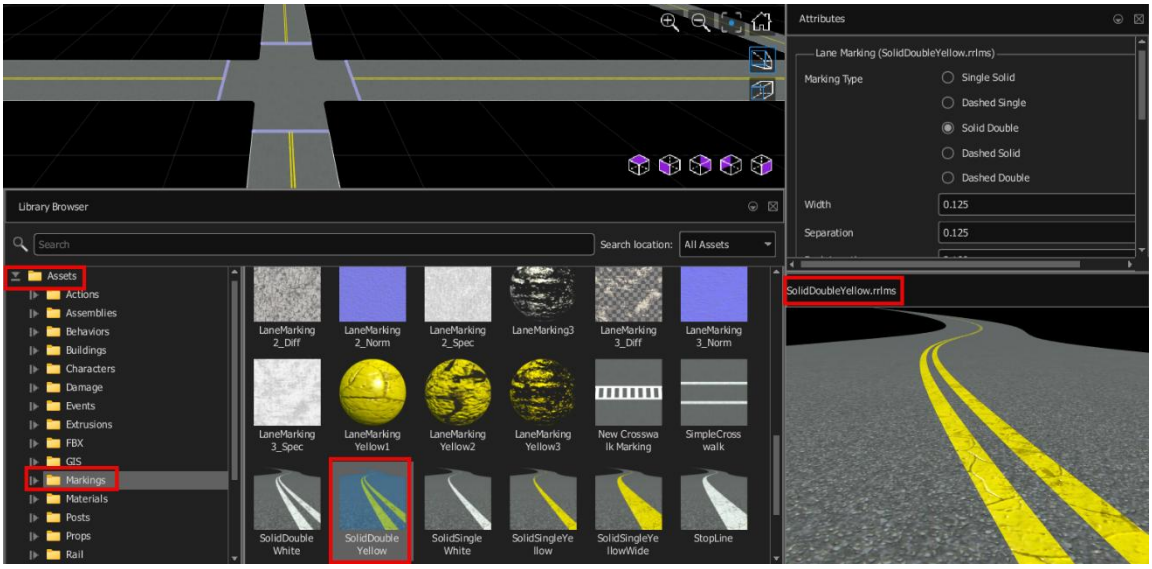


Figure 27: Find assets in the RoadRunner Library Browser pane.

The second way to find these assets is to open a window explorer and navigate to your RoadRunner project directory (see **Figure 28**). Look through the directories for the asset you want. If the file name ends with ".rrmeta", you should remove that from the name when you enter it into the relative asset path.

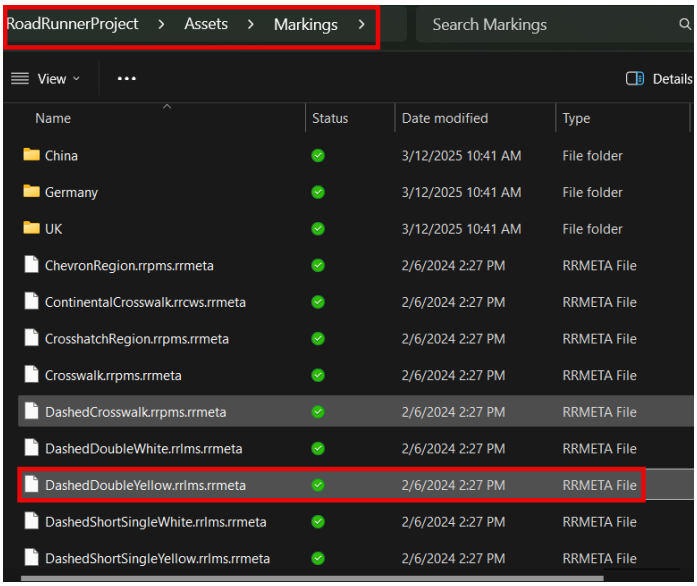


Figure 28: Use Windows explorer to find asset files in the RoadRunner Project Assets directory.

Most directory names in the Assets directory are self-explanatory. There are a few exceptions as follows:

- Barriers are found in “Assets\Extrusions”
- Stop and Yield signs with poles, intersection-spanning Traffic Lights, and common Electricity poles can be found in “Assets\Assemblies” *NOTE: these are not signs, they are static objects.*
- Stop and Yield signs without poles can be found in “Assets\Signs” *Note: These are signs.*
- “Assets\Props” contains many of the pieces required to build up trusses for overhead freeway signs, poles for other signs, construction elements, etc.

Assets

Now that we have defined the asset types, we can place those assets throughout the RRHD map.

Lane Markings are treated slightly differently from the other assets. They do not have a corresponding independent object like Barriers, Static Objects, Stencil Markings, and Curved Markings. Instead, they become a Parametric Attribute for a Lane Boundary.

Using the double solid yellow line marking reference (sdyMarkRef) we created above; we can add it to the center lane boundary object (cntrLaneBndry) we created in the **Create a Simple Lane Boundary** section above.

```
cntrLaneBndry.ParametricAttributes =
roadrunner.hdmap.ParametricAttribution(Span=[0, 1],
MarkingReference=sdyMarkRef)
```

```
ParametricAttribution with properties:

                Span: [0 1]
    MarkingReference: [1×1 roadrunner.hdmap.MarkingReference]
  SpeedLimitReference: [0×0 roadrunner.hdmap.SpeedLimitReference]
     SignalReference: [0×0 roadrunner.hdmap.SignalReference]
```

Figure 29: RRHD ParametricAttribution object.

Creating a Barrier, Sign, Static Object, Stencil Marking, or Curved Marking happens in nearly identical ways. All objects require ID, Geometry, Reference, and Metadata. Barriers and Curved Markings have extra properties regarding orientations. We will not cover those here.

Assume we want to add a barrier along the east bound lane of the road west of the junction. First, we create the barrier type object for our guardrail.

```
barrierType = roadrunner.hdmap.BarrierType(ID="GUARDRAIL",
ExtrusionPath=roadrunner.hdmap.RelativeAssetPath(AssetPath=
"Assets\Extrusions\GuardRail.rrext"))
```

```
Command Window
>> barrierType = roadrunner.hdmap.BarrierType(ID="GUARDRAIL",...
 ExtrusionPath=roadrunner.hdmap.RelativeAssetPath(AssetPath = "Assets\Extrusions\GuardRail.rrext"))

barrierType =

  BarrierType with properties:

              ID: "GUARDRAIL"
    ExtrusionPath: [1×1 roadrunner.hdmap.RelativeAssetPath]
```

Figure 30: RRHD BarrierType object.

Then, we can create a reference to this barrier and create the barrier about 0.2m beyond the edge of the East bound outer lane boundary.

```
barRef  = roadrunner.hdmap.Reference(ID = "GUARDRAIL")
barrier = roadrunner.hdmap.Barrier(ID="EB_LGW_Bar",
BarrierTypeReference=barRef, Geometry=EastBoundLaneW.Geometry-2)
```

```
Command Window
>> barrier = roadrunner.hdmap.Barrier(ID="EB_LGW_Bar", BarrierTypeReference = barRef,...
Geometry = EastBoundLaneW.Geometry-2)

barrier =

  Barrier with properties:

                    ID: "EB_LGW_Bar"
              Geometry: [2×2 double]
   BarrierTypeReference: [1×1 roadrunner.hdmap.Reference]
          FlipLaterally: 0
              Metadata: [0×1 roadrunner.hdmap.Metadata]
```

Figure 31: RRHD Barrier Object with BarrierTypeReference.

Once the lane marking, or other Asset Type has been created, add it, as appropriate, to the RRHD map.

```
rrMap.LaneMarkings(end+1) = solidDbleYllw
```

Note About Relative Asset Paths

Relative Asset Paths are paths relative to your RoadRunner project directory leading to the asset you care about. In the above example, we made a solid double yellow lane marking. No matter what kind of asset you are looking for, you can find them in one of two ways. In the RoadRunner app find the “Library Browser” pane (see **Figure 27**). There you will find the folder structure for all assets in the RoadRunner project. Drill into the structure to find the asset you want. In the Attributes pane, the asset and the file name will appear in the preview.

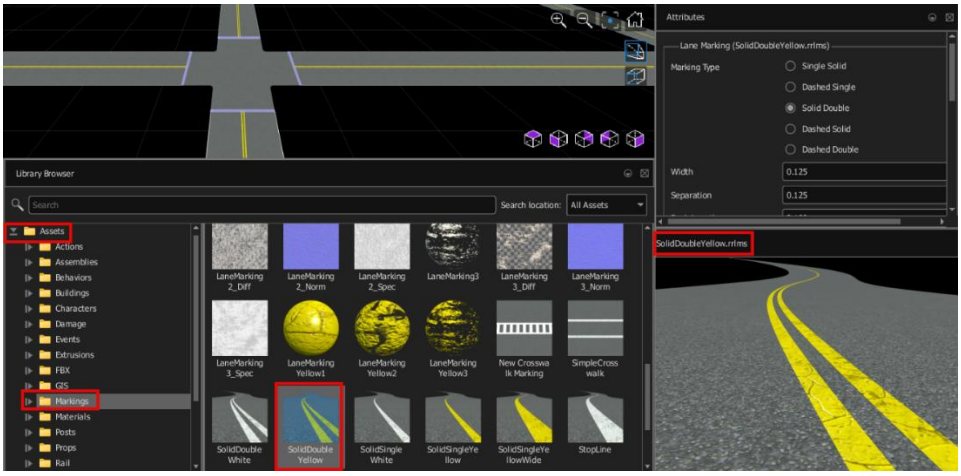


Figure 32: Find assets in the RoadRunner Library Browser pane.

The second way to find these assets is to open a window explorer and navigate to your RoadRunner project directory (see **Figure 28**). Look through the directories for the asset you want. If the file name ends with “.rrmeta”, you should remove that from the name when you enter it into the relative asset path.

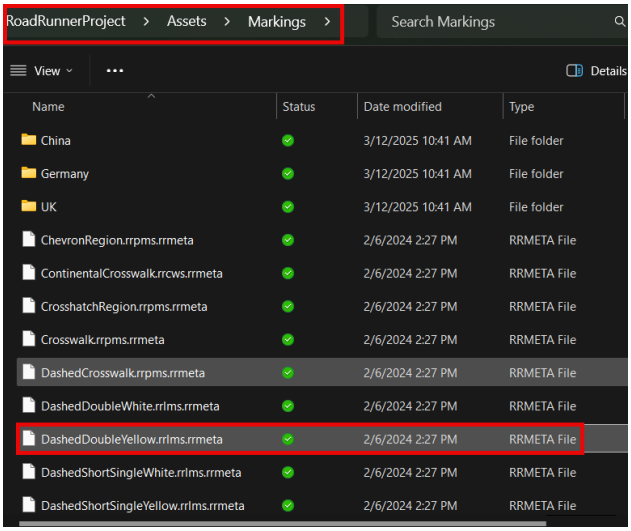


Figure 33: Use Windows explorer to find asset files in the RoadRunner Project Assets directory.

Most directory names in the Assets directory are self-explanatory. There are a few exceptions as follows:

- Barriers are found in “Assets\Extrusions”
- Stop and Yield signs with poles, intersection-spanning Traffic Lights, and common Electricity poles can be found in “Assets\Assemblies” *NOTE: these are not signs, they are static objects.*
- Stop and Yield signs without poles can be found in “Assets\Signs” *Note: These are signs.*
- “Assets\Props” contains many of the pieces required to build up trusses for overhead freeway signs, poles for other signs, construction elements, etc.

Assets

Now that we have defined the asset types, we can place those assets throughout the RRHD map.

Lane Markings are treated slightly differently from the other assets. They do not have a corresponding independent object like Barriers, Static Objects, Stencil Markings, and Curved Markings. Instead, they become a Parametric Attribute for a Lane Boundary.

Using the double solid yellow line marking reference (`sdyMarkRef`) we created above; we can add it to the center lane boundary object (`cntrLaneBndry`) we created in the **Create a Simple Lane Boundary** section above.

```
cntrLaneBndry.ParametricAttributes =
roadrunner.hdmap.ParametricAttribution(Span=[0, 1],
MarkingReference=sdyMarkRef)
```

```
ParametricAttribution with properties:

                Span: [0 1]
    MarkingReference: [1×1 roadrunner.hdmap.MarkingReference]
  SpeedLimitReference: [0×0 roadrunner.hdmap.SpeedLimitReference]
     SignalReference: [0×0 roadrunner.hdmap.SignalReference]
```

Figure 34: RRHD ParametricAttribution object.

Creating a Barrier, Sign, Static Object, Stencil Marking, or Curved Marking happens in nearly identical ways. All objects require ID, Geometry, Reference, and Metadata. Barriers and Curved Markings have extra properties regarding orientations. We will not cover those here.

Assume we want to add a barrier along the east bound lane of the road west of the junction. First, we create the barrier type object for our guardrail.

```
barrierType = roadrunner.hdmap.BarrierType(ID="GUARDRAIL",
ExtrusionPath=roadrunner.hdmap.RelativeAssetPath(AssetPath=
"Assets\Extrusions\GuardRail.rrext"))
```

```
Command Window
>> barrierType = roadrunner.hdmap.BarrierType(ID="GUARDRAIL",...
 ExtrusionPath=roadrunner.hdmap.RelativeAssetPath(AssetPath = "Assets\Extrusions\GuardRail.rrext"))

barrierType =

  BarrierType with properties:

              ID: "GUARDRAIL"
    ExtrusionPath: [1×1 roadrunner.hdmap.RelativeAssetPath]
```

Figure 35: RRHD BarrierType object.

Then, we can create a reference to this barrier and create the barrier about 0.2m beyond the edge of the East bound outer lane boundary.

```
barRef  = roadrunner.hdmap.Reference(ID = "GUARDRAIL")
barrier = roadrunner.hdmap.Barrier(ID="EB_LGW_Bar",
BarrierTypeReference=barRef, Geometry=EastBoundLaneW.Geometry-2)
```

```
Command Window
>> barrier = roadrunner.hdmap.Barrier(ID="EB_LGW_Bar", BarrierTypeReference = barRef,...
Geometry = EastBoundLaneW.Geometry-2)

barrier =

  Barrier with properties:

                    ID: "EB_LGW_Bar"
              Geometry: [2×2 double]
   BarrierTypeReference: [1×1 roadrunner.hdmap.Reference]
          FlipLaterally: 0
              Metadata: [0×1 roadrunner.hdmap.Metadata]
```

Figure 36: RRHD Barrier Object with BarrierTypeReference.

Pitfall

Object reference type names are similar, but not identical to the object being referenced, i.e., for Barriers, the reference type property is BarrierTypeReference, for Signs it is SignTypeReference, for Static objects, it is ObjectTypeReference, and for the Stencil and Curved Markings, it is MarkingTypeReference.

Once the static object is finished, add it to the RRHD map appropriately.

```
rrMap.Barriers(end+1) = barrier
```

Create the Scene in RoadRunner

Once you have built up the RRHD map, it is useful to save it as an asset in the RoadRunner project directory. This will allow you to import the RRHD map easily into the RoadRunner application. You may

do that in two ways. Either programmatically, or by navigating to where you saved it in the Library Browser of the Application and dragging and dropping into the canvas.

(The file locations will be unique to the user's computer.)

```
rrInstallLoc = "C:\Program Files\RoadRunner R2025a\bin\win64"
rrProjLoc    = "C:\Users\userName\RRProject"
rrhdFile     = rrProjLoc + "\Assets\SimpleJunction.rrhd"

write(rrMap, rrhdFile)
rrApp = roadrunner(rrProjLoc, "InstallationFolder", rrInstallDir)
rrApp.importScene(rrhdFile, "RoadRunner HD Map")
```

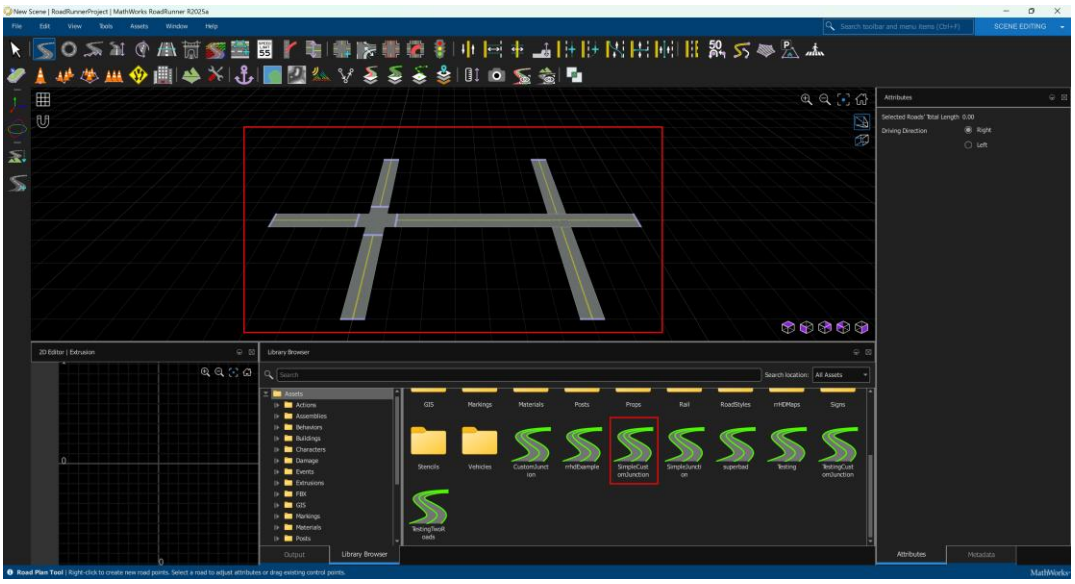


Figure 37: RoadRunner's visualization of the RRHD map we created.

Pitfall

RoadRunner does not build the crossroads on the right with a junction. To change this default behavior, you may specify build options for the **`importScene`** function.

```
OLGopts           = enableOverlapGroupsOptions
OLGopts.IsEnabled = false
rrhdBuildOpts     = roadrunnerHDMapBuildOptions
rrhdBuildOpts.EnableOverlapGroupsOptions = OLGopts
rrApp.importScene(rrhdFileNameAndLoc, "RoadRunner HD Map",
roadrunnerHDMapImportOptions(buildOptions=rrhdBuildOpts))
```

Compare our previous RRHD build:

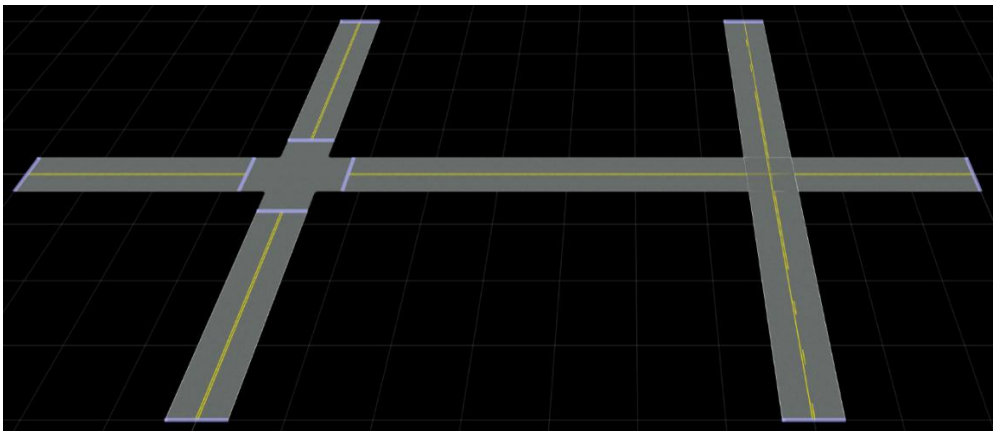


Figure 38: Road network defined in our RRHD map, built by RoadRunner with default build options.

To the build with the enable overlap group options set to false:

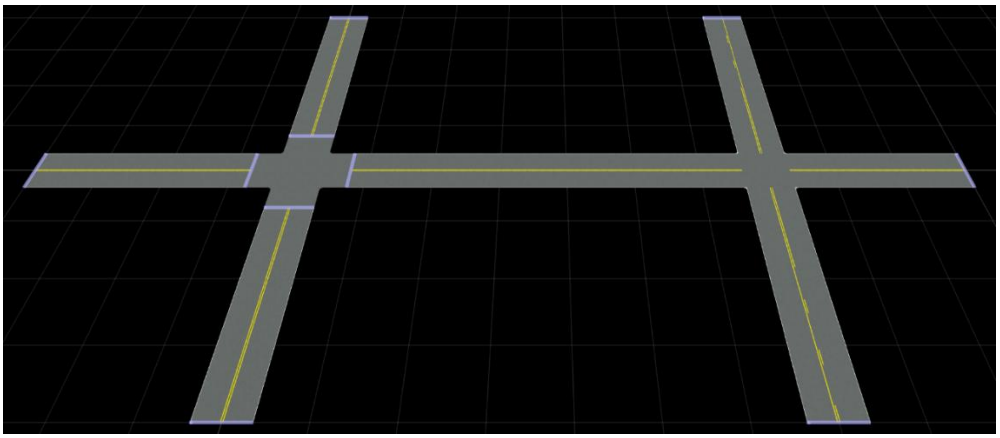


Figure 39: Road network defined in our RRHD map, built by RoadRunner with group build option set to false.

Recommended Workflow to Build RRHD Map

While there is no required way to assemble an RRHD map, we have found the easiest way is to follow the organization of the HD map from which the data is being drawn. **Figure 35** shows the flowchart enumerated in the list below with our recommended workflow to create a script to automatically build up an RRHD map from HD map data.

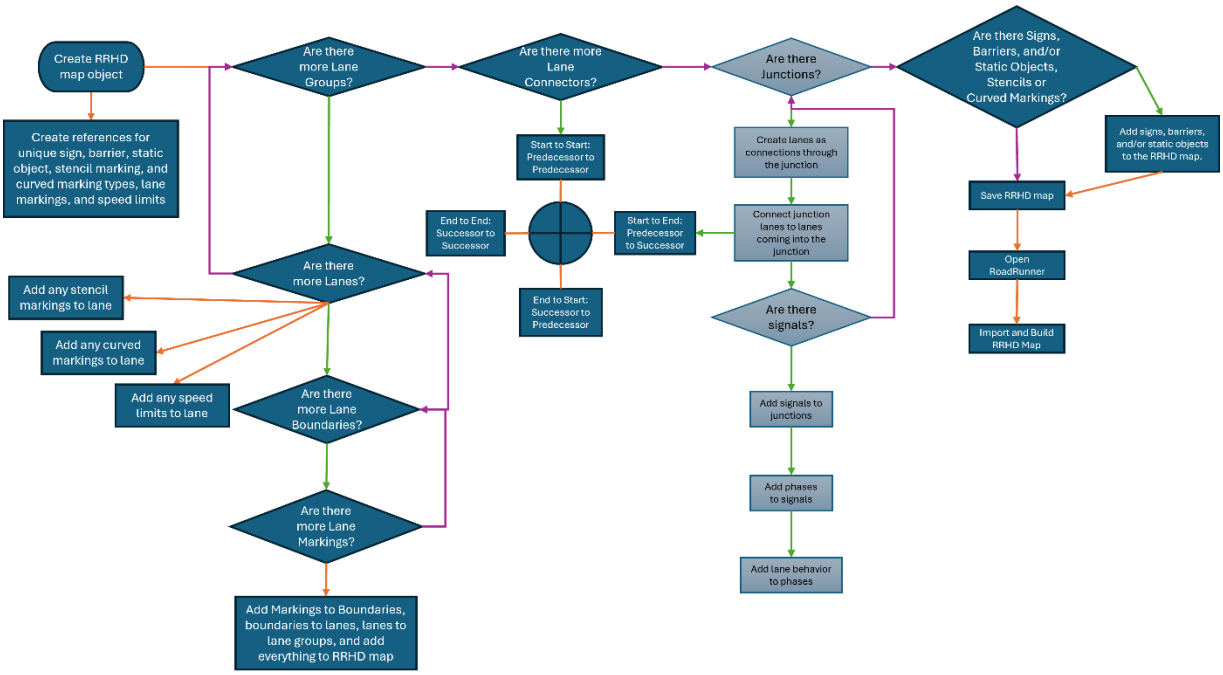


Figure 40: Flowchart to help create a script to build an RRHD map from HD map data.

1. Create a roadrunner HD map object
2. If necessary, convert the geometry of your various objects from (Lat, Lon, Alt) to (x, y, z) – this can be done as you proceed through the steps below
3. Add the unique assets found within your map to the RRHD map: Lane Markings, Stencil Marking Types, Curved Marking Types, Barrier Types, Sign Types, Static Object Types, Speed Limits, Signal Types. These references will be used throughout the next step to add these assets in multiple places throughout the RRHD map. E.g., Double Solid Yellow lines will likely be used in many places, but you only need one LaneMarking reference for this asset.
4. Loop through the lane groups in your HD map object
    a. Loop through the lanes in your lane group
        i. Add the connectivity to the lanes on the left and/or right side of the current lane
        ii. Add lane boundaries to left and right side of lane
            1. Include lane markings (parametric references)
            2. Add lane boundaries to RRHD map
        iii. Add Speed limits to the lane (parametric references)
        iv. Add current lane to the RRHD Map
    b. Add current lane to lane group
    c. Add lane group to RRHD map
5. Loop through the Lane Group connection information from the HD Map provider
    a. Loop through the lane groups associated with the connection
        i. Loop through the lanes within the lane groups
            1. Find and add the predecessors and successors for each lane within the lane group
6. Loop through stencil markings and add to rrhd object
7. Loop through curve markings and add to rrhd object

8. Loop through signs and add to rrhd object
9. Loop through barriers and add to rrhd object
10. Loop through static objects and add to rrhd object
11. Create junctions if necessary
12. Save RRHD map as an asset in your RoadRunner project directory
13. Create an instance of the RoadRunner application and connect it to a RoadRunner object in MATLAB
14. Import and build the scene created by RRHD

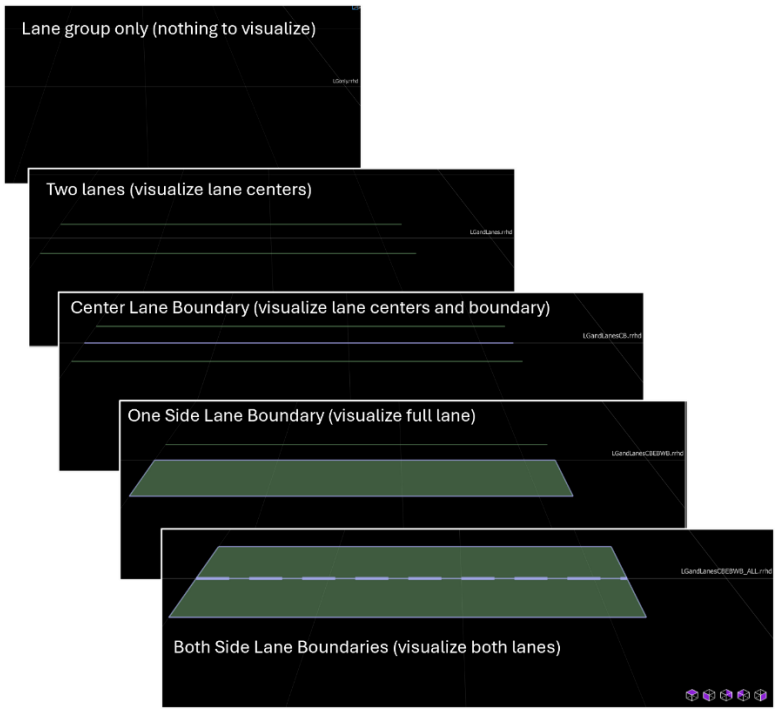


Figure 41: Step-by-step buildup of an RRHD map.

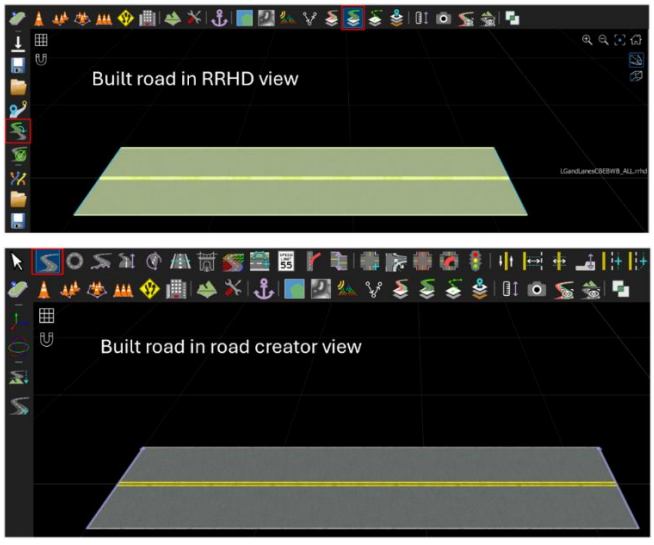


Figure 42: A built road in RRHD and road creator views.

Example: HD Map to RRHD

In this section, we will take a real-world location, convert the desired portion of the roadway into the RRHD map, and import it into RoadRunner. We are interested in recreating a stretch of Interstate 95 as it passes near Boston, Massachusetts. We do not want all the side streets, or even all of the major thoroughfares that intersect I95. We do, however, want to have the entrance and exit ramps to those side streets and thoroughfares. Using a depth-first-search algorithm through the links and nodes coming from our HD map provider of choice, we will select (more or less) the parts of the roadways highlighted in **Figure 38**.

Figure 43: Section of I95 to recreate using RRHD. Most, but not all, red segments will be imported.

It is beyond the scope of this paper to discuss the procedure for selecting the roadway from the HD map provider, but assuming this information can be obtained, and the necessary data can be extracted from the HD map, we can begin to construct the RRHD map.

To begin, as part of the HD map data processing, we extract the various assets (barriers, lane markings, and signs) unique to this section of road. We also extract the various lane type definitions used by the HD map provider.

barrierIds — 3×1 string

	1
1	JERSEY_BARRIER
2	GUARDRAIL
3	WALL

laneTypeIds — 23×1 string

	1
1	LANE_TYPE_UNKNOWN
2	REGULAR
3	HIGH_OCCUPANCY_VEHICLE
4	DRIVABLE_SHOULDER
5	BICYCLE
6	PARKING
7	REVERSIBLE
8	EXPRESS
9	ACCELERATION
10	DECELERATION
11	AUXILIARY
12	SLOW
13	PASSING
14	REGULATED_ACCESS
15	TURN
16	CENTER_TURN
17	CENTRE_TURN
18	TRUCK_PARKING
19	SHOULDER
20	VARIABLE_DRIVING
21	DRIVABLE_PARKING
22	OTHER
23	BUS

markingIds — 13×1 string

	1
1	SOLIDYELLOW
2	DOUBLESOLIDYELLOW
3	DASHEDYELLOW
4	DOUBLEDASHEDYELLOW
5	DASHEDSOLIDYELLOW
6	SOLIDDASHEDYELLOW
7	SOLIDWHITE
8	DOUBLESOLIDWHITE
9	DASHEDWHITE
10	DOUBLEDASHEDWHITE
11	DASHEDSOLIDWHITE
12	SOLIDDASHEDWHITE
13	NONE

signIds — 21×1 string

	1
1	BLACK_DIAMOND
2	BLACK_RECTANGLE
3	BLUE_SIGN_SHAPE_OTHER
4	BLUE_RECTANGLE
5	BROWN_RECTANGLE
6	GREEN_RECTANGLE
7	GREEN_SIGN_SHAPE_OTHER
8	RED_OCTAGON
9	RED_RECTANGLE
10	RED_TRIANGLE
11	SIGN_COLOR_OTHER_DIAMOND
12	SIGN_COLOR_OTHER_RECTANGLE
13	SIGN_COLOR_OTHER_TRIANGLE
14	SIGN_COLOR_OTHER_SIGN_SHAPE_OTHER
15	WHITE_CIRCLE
16	WHITE_RECTANGLE
17	WHITE_SIGN_SHAPE_OTHER
18	YELLOW_CIRCLE
19	YELLOW_DIAMOND
20	YELLOW_RECTANGLE
21	YELLOW_SIGN_SHAPE_OTHER
22	
23	

Figure 44: The unique barriers, lane markings, signs and lane type definitions provided by the HD map provider.

This allows us to find the corresponding asset files and lane type definitions in RoadRunner. We need a proper mapping between these definitions in RoadRunner and the HD map before we can automate the process.

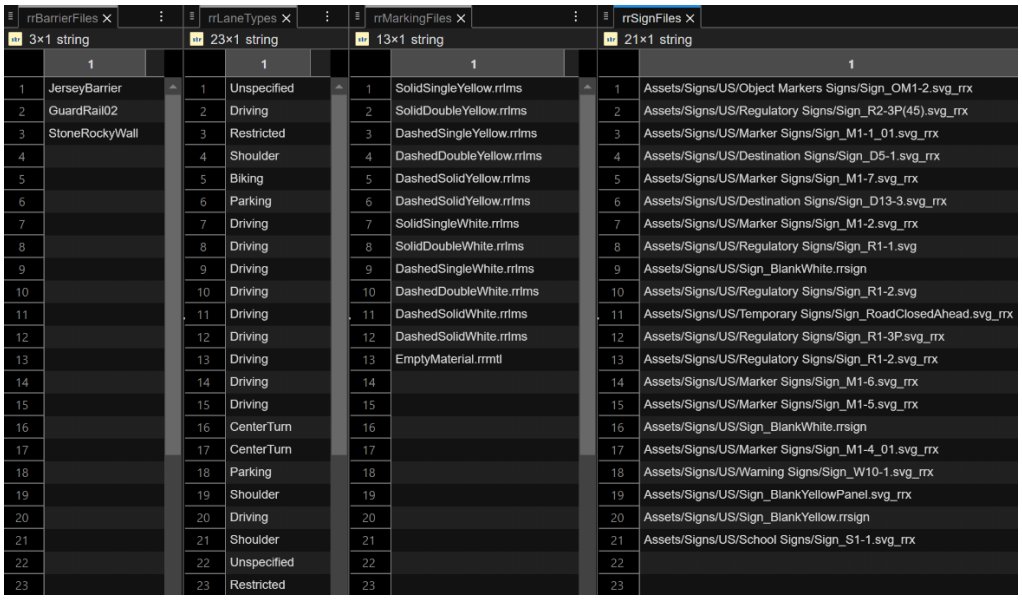


Figure 45: RoadRunner asset files and Lane type definitions to map to those provided by the HD map provider.

Next, we need the data from the HD map. We separated out the signs and barriers of interest from the lane groups, lanes, and marking of interest. The map we used contains information about lane group connections, which makes defining the lane processors and successor much easier than having to write the connection algorithm ourselves. The image below shows the top-level organization of the data. The data format is a relatively simplified structure from the HD map structure. The searchTable contains information about the subset of links found through our depth-first-search algorithm. The signs and barriers of interest are those signs and barriers that exist along the route we have defined in the searchTable.

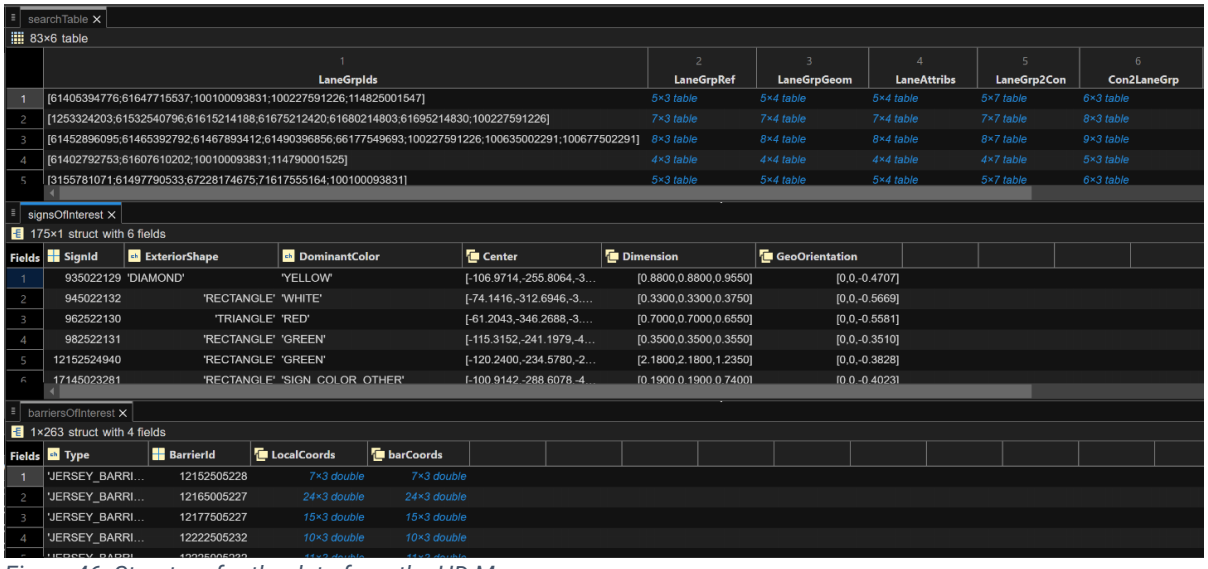


Figure 46: Structure for the data from the HD Map

Finally, we need to know the geo center for the map and the extents of the map area. We provide this information in the geoCenter and geoBounds variables. The geoCenter is given as latitude, longitude, and altitude. The geoBounds are the (x, y, z) extents of the map size in meters. This can be easily calculated from the HD map data.

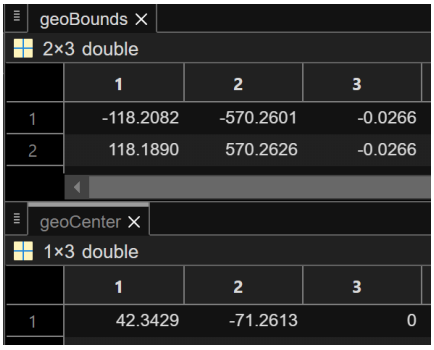


Figure 47: Geo center and map extents

There are also several variables the user must provide. To create the RRHD map, the user should provide the name of the map Author. To save the RRHD map for import, the user should provide a file name, and it must have the '.rrhd' extension. To open the RoadRunner application programmatically, the user needs to provide the locations of the RoadRunner project and installation directory. The installation directory is likely in the default location, but the user may elect to save it in a different place, or they must access it through a network drive. The project directory does not have a default location. The user must specify the location the first time they open RoadRunner.

```
21      %%%%%%%%%%%%%%%%%%%%%%%%%%
22      % User Defined quantities %
23      %%%%%%%%%%%%%%%%%%%%%%%%%%
24      mapAuthor    = "kmcgarri";                                          % map author
25      rrhdFileName = "rrhdExample.rrhd";                                  % file name for RRHD map
26      rrProjLoc    = "C:\Users\kmcgarri\OneDrive - MathWorks\RoadRunnerProject\"; % RoadRunner project directory
27      rrInstallDir = "c:\Program Files\RoadRunner R2025a\bin\win64";      % RoadRunner install directory
```

Figure 48: User defined quantities for creating an RRHD map.

Now that we have all the data we need, we can start to build up the map.

First, we must create an RRHD object.

```
34      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
35      % create RoadRunner HD map %
36      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
37      rrMap = roadrunnerHDMap(Author=mapAuthor, GeoReference=geoCenter(1:2), GeographicBoundary=geoBounds);
```

Figure 49: Step 1 ***Create a RoadRunner HD map object****.*

```
rrMap =

  roadrunnerHDMap with properties:

                Author: "kmcgarri"
           GeoReference: [42.3429 -71.2613]
      GeographicBoundary: [2×3 double]
                 Lanes: [0×1 roadrunner.hdmap.Lane]
            SpeedLimits: [0×1 roadrunner.hdmap.SpeedLimit]
         LaneBoundaries: [0×1 roadrunner.hdmap.LaneBoundary]
             LaneGroups: [0×1 roadrunner.hdmap.LaneGroup]
           LaneMarkings: [0×1 roadrunner.hdmap.LaneMarking]
              Junctions: [0×1 roadrunner.hdmap.Junction]
           BarrierTypes: [0×1 roadrunner.hdmap.BarrierType]
               Barriers: [0×1 roadrunner.hdmap.Barrier]
             SignTypes: [0×1 roadrunner.hdmap.SignType]
                 Signs: [0×1 roadrunner.hdmap.Sign]
      StaticObjectTypes: [0×1 roadrunner.hdmap.StaticObjectType]
          StaticObjects: [0×1 roadrunner.hdmap.StaticObject]
    StencilMarkingTypes: [0×1 roadrunner.hdmap.StencilMarkingType]
        StencilMarkings: [0×1 roadrunner.hdmap.StencilMarking]
      CurveMarkingTypes: [0×1 roadrunner.hdmap.CurveMarkingType]
          CurveMarkings: [0×1 roadrunner.hdmap.CurveMarking]
            SignalTypes: [0×1 roadrunner.hdmap.SignalType]
                Signals: [0×1 roadrunner.hdmap.Signal]
```

Figure 50: RRHD map. The user defined quantities have been added to the object.

Our rrMap object is empty except for the Author, GeoReference, and GeographicBoundary properties we passed into the function call.

Next, we will set the Asset Types in the RRHD map. To do this, we created a helper function called setRRAssetTypes. This function requires the RRHD map object, a string indicating which asset we are setting, a vector containing the file names (**Figure 45),** and a vector containing the asset IDs (**Figure 44**). We do this for each of the asset types we want to include in the RRHD map.

```
39        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
40        % add the barrier, marking, and sign types to rrMap %
41        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
42        setRRAssetTypes(rrMap, "Barrier",     rrBarrierFiles, barrierIds);
43        setRRAssetTypes(rrMap, "LaneMarking", rrMarkingFiles, markingIds);
44        setRRAssetTypes(rrMap, "Sign",        rrSignFiles,    signIds);
```

Figure 51: Step 3 ***add the unique assets found within your map to the rrhd map****: Lane Markings, Stencil Marking Types, Curved Marking Types, Barrier Types, Sign Types, Static Object Types, Speed Limits, Signal Types. These references will be used throughout the next step to add these assets in multiple places throughout the RRHD map. E.g., Double Solid Yellow lines will likely be used in many places, but you only need one LaneMarking reference for this asset. setRRAssetTypes is a user-created function (source code shown in* ***Figure 52****) to help define the various asset types used in this example.*

The important part of the helper function defines a relative asset path, the asset type, and inserts the asset type reference into the RRHD map.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% loop over assets used in this map %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for px = 1:numel(assetIds)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % create RelativeAssetPath for assets %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    relAssetPath = roadrunner.hdmap.RelativeAssetPath(AssetPath = assetFiles(px));
    rrAssetType  = roadrunner.hdmap.(mapInsert)(ID = assetIds(px));

    %%%%%%%%%%%%%%%%%%%%
    % create assetType %
    %%%%%%%%%%%%%%%%%%%%
    if strcmp(assetType, "Barrier")
        rrAssetType.ExtrusionPath = relAssetPath;
    else
        rrAssetType.AssetPath = relAssetPath;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % add asset to asset type array in RoadRunner HD Map %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    rrMap.(mapInsert + "s")(end+1) = rrAssetType;
end % end loop over lane markings
```

Figure 52: Source code for the function shown in Figure 48 to define various asset types in RRHD. Note: regardless of the asset type, the basic functions calls are the same.

The RRHD map object now contains 13 unique lane markings, 3 unique barrier types, and 21 unique sign types.

```
rrMap =

  roadrunnerHDMap with properties:

                Author: "kmcgarri"
          GeoReference: [42.3429 -71.2613]
     GeographicBoundary: [2×3 double]
                 Lanes: [0×1 roadrunner.hdmap.Lane]
           SpeedLimits: [0×1 roadrunner.hdmap.SpeedLimit]
        LaneBoundaries: [0×1 roadrunner.hdmap.LaneBoundary]
            LaneGroups: [0×1 roadrunner.hdmap.LaneGroup]
          LaneMarkings: [13×1 roadrunner.hdmap.LaneMarking]
             Junctions: [0×1 roadrunner.hdmap.Junction]
          BarrierTypes: [3×1 roadrunner.hdmap.BarrierType]
              Barriers: [0×1 roadrunner.hdmap.Barrier]
             SignTypes: [21×1 roadrunner.hdmap.SignType]
                 Signs: [0×1 roadrunner.hdmap.Sign]
     StaticObjectTypes: [0×1 roadrunner.hdmap.StaticObjectType]
         StaticObjects: [0×1 roadrunner.hdmap.StaticObject]
    StencilMarkingTypes: [0×1 roadrunner.hdmap.StencilMarkingType]
       StencilMarkings: [0×1 roadrunner.hdmap.StencilMarking]
     CurveMarkingTypes: [0×1 roadrunner.hdmap.CurveMarkingType]
         CurveMarkings: [0×1 roadrunner.hdmap.CurveMarking]
           SignalTypes: [0×1 roadrunner.hdmap.SignalType]
               Signals: [0×1 roadrunner.hdmap.Signal]
```

Figure 51: Note, our RRHD object now contains all the asset type definitions.

Because our HD Map provider uses links and nodes as the basis for defining a roadway, we have organized our data by links, which may contain several lane groups. Lane groups may span more than one link, and so we want to make sure we do not double-count any given lane group.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Our data is arranged by links rather than lane groups, therefore it is      %
% possible to encounter a given lane group more than once. We need to keep    %
% track of whetherwe have already added the lane group so we don't double add %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
laneGroupAdded = unique(vertcat(searchTable.LaneGrpIds{:}), "stable");
laneGroupAdded = [laneGroupAdded, false(size(laneGroupAdded))];
```

Figure 52: Variable to keep track of whether we have added the lane group to the RRHD map.

We loop through the rows of our searchTable and extract the lane group and associated lanes attributes information.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% loop over links to find the lane groups and lanes to set up the RRHD map %
% (each row in the searchTable contains link information)                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for bx = 1:numel(searchTable.LaneGrpRef)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % grab the lane group information %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    currLaneGeom = searchTable.LaneGrpGeom{bx};
    currLaneAtrb = searchTable.LaneAttribs{bx};
```

Figure 53: Loop over Links from HD Map. In this example, the map uses links as its base road object. Each link will contain multiple Lane Groups.

```
currLaneGeom =

  5×4 table

    LaneGroupRef    ReferenceGeometry    LaneGeometries    LaneBoundaryGeometries
    ____________    _________________    ______________    ______________________

     61405394776        1×1 struct        {6×1 struct}          {7×1 struct}
     61647715537        1×1 struct        {4×1 struct}          {5×1 struct}
    100100093831        1×1 struct        {6×1 struct}          {7×1 struct}
    100227591226        1×1 struct        {6×1 struct}          {7×1 struct}
    114825001547        1×1 struct        {5×1 struct}          {6×1 struct}
```

Figure 54: In the current link, there are five Lane Groups. The First Lane group has 6 lanes associated with it, and 7 lane boundaries. The other lane groups have N lanes and N+1 Lane Boundaries as defined by the HD Map provider.

```
currLaneAtrb =

  5×4 table

    LaneGroupRef    LaneAttribution    LaneBoundaryAttribution    ParametricAttribution
    ____________    _______________    _______________________    _____________________

     61405394776     {6×1 struct}           {5×1 struct}               {1×1 struct}
     61647715537     {4×1 struct}           {4×1 struct}               {1×1 struct}
    100100093831     {6×1 struct}           {5×1 struct}               {1×1 struct}
    100227591226     {6×1 struct}           {5×1 struct}               {1×1 struct}
    114825001547     {5×1 struct}           {6×1 struct}               {1×1 struct}
```

Figure 55: The Attribution data is organized per the HD Map provider, and there may not always be an attribute explicitly associated with each element in the road. You will need to know the defaults from your HD Map provider to properly assign these attributes.

We loop through the lane groups associated with the link and extract the information from the map data. At this point, it is important to understand how your HD Map provider structures the data, as this will determine how you extract the relevant information and apply it to the correct lanes, boundaries, lane groups, etc.

```
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % loop over lane groups in this link %
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for lgx = 1:size(currLaneGeom, 1)
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % setup the rrhd lane group object %
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                currLGGeom  = currLaneGeom.ReferenceGeometry(lgx).LocalCoords;
                laneGroupID = currLaneGeom.LaneGroupRef(lgx);
```

Figure 56: Step 4 **Loop through the lane groups in your HD map object.**

If we have not already added a given lane group to the RRHD map, we can create a Lane Group object from the information we have extracted.

```
                %%%%%%%%%%%%%%%%%%%%%%%%%%%
                % look for the lane group %
                %%%%%%%%%%%%%%%%%%%%%%%%%%%
                lgax = (laneGroupAdded(:, 1) == laneGroupID);

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % if we have not already added this lane group, add it %
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                if (laneGroupAdded(lgax, 2) == uint64(0))
                    currRRHDLaneGrp = roadrunner.hdmap.LaneGroup(ID = laneGroupID, Geometry = currLGGeom);
```

Figure 57: Determine whether the lane group has been added to the RRHD map.

Now, we loop over the lanes within the current lane group and set up Lane objects for the RRHD map.

```
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % loop over lanes in the current lane group %
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    for lx = 1:numel(currLaneInfo.Geometries)
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        % create a rrhd lane object for the current lane %
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        travelDir    = currLaneAtrb.LaneAttribution{lgx}(lx).TravelDir;
                        laneType     = currLaneAtrb.LaneAttribution{lgx}(lx).LaneType;
                        laneGeom     = currLaneGeom.LaneGeometries{lgx}(lx).LanePathGeometry.LocalCoords;
                        laneID       = num2str(laneGroupID) + "_" + num2str(lx);
                        currRRHDLane = roadrunner.hdmap.Lane(ID = laneID, Geometry = laneGeom,...
                                                             TravelDirection = travelDir, LaneType = laneType);
```

Figure 58: Step 4a **Loop through the lanes in your lane group.**

For a given lane, we will have information on both the left and right lane boundaries. For the first lane in our group, we will make both a left and right boundary object. After that, as we go through our loop over the lanes, we only need to add the right boundary because the right boundary from the previous lane becomes the left boundary of our current lane.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% we only need to find the left lane boundary for the first lane after that, once we %
% have the right lane boundary it becomes the left lane boundary for the next lane   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if lx == 1
    llbx = currLaneInfo.Geometries(lx).LeftLaneBoundaryNumber;
    currRRHDLeftBndry = getRRHDLaneBndryObj(currBndryInfo, currRRHDLane.ID, llbx);
    rrMap.LaneBoundaries(end+1) = currRRHDLeftBndry;
else
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % the right lane boundary from the last lane is the left lane boundary for this lane! %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    currRRHDLeftBndry = currRRHDRightBndry;
end % end if first, else other lanes

%%%%%%%%%%%%%%%%%%%%%%%%%%
% now get right boundary %
%%%%%%%%%%%%%%%%%%%%%%%%%%
rlbx = currLaneInfo.Geometries(lx).RightLaneBoundaryNumber;
currRRHDRightBndry = getRRHDLaneBndryObj(currBndryInfo, currRRHDLane.ID, rlbx);
rrMap.LaneBoundaries(end+1) = currRRHDRightBndry;
```

Figure 59: Step 4a ***Add the connectivity to the lanes on the left and/or right side of the current lane,*** *including lane boundaries, and speed limits (we omit these in this example). Finally, adding the lane to the RRHD map.*

To get the lane boundary objects for the RRHD map, we have created a helper function, getRRHDLaneBndryObj, which requires lane boundary and lane information. This function will find the various lane markings associated with the given lane and the spans along the lane over which these markings apply. Then, it creates a Lane Boundary object to return to the calling function. The salient part of this function is as follows.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create the RRHD lane boundary object %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
currRRHDLaneBndry = roadrunner.hdmap.LaneBoundary(ID = bndryID, Geometry = bndryGeom);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% setting up the rrhd lane boundary parametric attributions object %
% loop over the total number of markings                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
prmAttr = [];
for mx = 1:numel(lmID)
    rrRef     = roadrunner.hdmap.Reference(ID = lmID(mx));
    rrMarkRef = roadrunner.hdmap.MarkingReference(MarkingID = rrRef);
    prmAttr   = [prmAttr; roadrunner.hdmap.ParametricAttribution(Span = markingSpan(mx, :),...
                                                                   MarkingReference = rrMarkRef)];
end % end loop over number of marks for this lane (mx)

currRRHDLaneBndry.ParametricAttributes = prmAttr;
```

Figure 60: Step 4a ***Include lane markings (parametric references)*** *to the RRHD map*

Once we have the right and left boundaries, we can add them to the current lane. We can then add the current lane to the RRHD map.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% set the boundaries in the current lane object %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
currRRHDLane.leftBoundary( currRRHDLeftBndry.ID,  "Alignment", "Forward");
currRRHDLane.rightBoundary(currRRHDRightBndry.ID, "Alignment", "Forward");
rrMap.Lanes(end+1) = currRRHDLane;
```

Figure 61: Step 4a ***Add lane boundaries to left and ride side of lane.***

Finally, we create an aligned reference for this lane and add it to the current lane group, thus completing the loop over the lanes.

```
136                     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
137                     % get the reference to this lane %
138                     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
139                     laneRef    = roadrunner.hdmap.Reference(ID = currRRHDLane.ID);
140                     allLaneRef = roadrunner.hdmap.AlignedReference(Reference = laneRef, Alignment = "Forward");
141                     currRRHDLaneGrp.Lanes(end+1) = allLaneRef;
142                 end % end loop over lanes in current lane group (lx)
```

*Figure 62: Step 4b **Add current lane to lane group**.*

Now that the lane group has references to all the lanes, which have references to all the lane boundaries, and thus we have lateral connectivity between the lanes within the lane group, we can add the lane group to the RRHD map.

```
144                     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
145                     % add all lane references to the lane group object %
146                     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
147                     rrMap.LaneGroups(end+1) = currRRHDLaneGrp;
```

*Figure 63: Step 4c **Add lane group to RRHD map**.*

The RRHD map object now has 6 lanes, 7 lane boundaries, and 1 lane group. These arrays will continue to grow as we loop over the lane groups and links to build up the map.

```
rrMap =

  roadrunnerHDMap with properties:

                Author: "kmcgarri"
          GeoReference: [42.3429 -71.2613]
    GeographicBoundary: [2×3 double]
                 Lanes: [6×1 roadrunner.hdmap.Lane]
           SpeedLimits: [0×1 roadrunner.hdmap.SpeedLimit]
       LaneBoundaries: [7×1 roadrunner.hdmap.LaneBoundary]
           LaneGroups: [1×1 roadrunner.hdmap.LaneGroup]
         LaneMarkings: [13×1 roadrunner.hdmap.LaneMarking]
            Junctions: [0×1 roadrunner.hdmap.Junction]
         BarrierTypes: [3×1 roadrunner.hdmap.BarrierType]
             Barriers: [0×1 roadrunner.hdmap.Barrier]
            SignTypes: [21×1 roadrunner.hdmap.SignType]
                Signs: [0×1 roadrunner.hdmap.Sign]
     StaticObjectTypes: [0×1 roadrunner.hdmap.StaticObjectType]
         StaticObjects: [0×1 roadrunner.hdmap.StaticObject]
   StencilMarkingTypes: [0×1 roadrunner.hdmap.StencilMarkingType]
       StencilMarkings: [0×1 roadrunner.hdmap.StencilMarking]
     CurveMarkingTypes: [0×1 roadrunner.hdmap.CurveMarkingType]
         CurveMarkings: [0×1 roadrunner.hdmap.CurveMarking]
           SignalTypes: [0×1 roadrunner.hdmap.SignalType]
               Signals: [0×1 roadrunner.hdmap.Signal]
```

Figure 64: RRHD map with lane groups, lanes, and lane boundaries.

After we have made this first pass through the data, we will need to loop through the data again to connect the successors and predecessors for each lane. Without proper connectivity, the RRHD map will not render correctly when imported into RoadRunner. Our HD Map provider gives Lane Group connection information. The next section of code loops through this connection information and finds the lanes to connect.

There are four ways lanes can connect to one another:

1. Both lanes start at the connector
2. Current lane starts and next lane ends at the connector
3. Current lane ends at the connector, and next lane starts at the connector
4. Both lanes end at the connector

Most of the code in this section of the file is to determine which of these four ways the lanes connect. Once we know this, we can define the connections for the RRHD map. Note, both lanes may be Predecessors of one another if they both start at the connection, whereas they would be Successors if they both end at the connection. Knowing the direction of the lane is important at this point.

```
284                         if laneConnInfo.IsStart(currConnLns(ccx))
285                             if laneConnInfo.IsStart(currConnLns(ncx))
286                                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
287                                 % both lanes start at connector %
288                                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
289                                 addPredecessor(rrMap.Lanes(cldx), nextLaneID, "Alignment", "Backward");
290                                 addPredecessor(rrMap.Lanes(nldx), currLaneID, "Alignment", "Backward");
291                             else
292                                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
293                                 % current lane starts and next lane ends at connector %
294                                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
295                                 addPredecessor(rrMap.Lanes(cldx), nextLaneID, "Alignment", "Forward");
296                                 addSuccessor(rrMap.Lanes(nldx),   currLaneID, "Alignment", "Forward");
297                             end
298                         else
299                             if laneConnInfo.IsStart(currConnLns(ncx))
300                                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
301                                 % current lane ends and next lane starts at connector %
302                                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
303                                 addSuccessor(rrMap.Lanes(cldx),   nextLaneID, "Alignment", "Forward");
304                                 addPredecessor(rrMap.Lanes(nldx), currLaneID, "Alignment", "Forward");
305                             else
306                                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
307                                 % both lanes end at connector %
308                                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
309                                 addSuccessor(rrMap.Lanes(cldx), nextLaneID, "Alignment", "Backward");
310                                 addSuccessor(rrMap.Lanes(nldx), currLaneID, "Alignment", "Backward");
311                             end % end if next lane starts, else ends, at connector
312                         end % end if current lane starts, else ends, at connector
```

Figure 65: Step 5 ***Find and add the predecessors and successors for each lane within the lane group****. Note the conditions under which a lane is considered a predecessor compared to a successor of another lane. The conditions have to do with whether a lane starts or ends at a connection.*

```
K>> rrMap.Lanes(1)

ans =

  Lane with properties:

                     ID: "61405394776_1"
               Geometry: [6×3 double]
        TravelDirection: Unspecified
       LeftLaneBoundary: [1×1 roadrunner.hdmap.AlignedReference]
      RightLaneBoundary: [1×1 roadrunner.hdmap.AlignedReference]
           Predecessors: [1×1 roadrunner.hdmap.AlignedReference]
             Successors: [1×1 roadrunner.hdmap.AlignedReference]
               LaneType: Shoulder
               Metadata: [0×1 roadrunner.hdmap.Metadata]
    ParametricAttributes: [0×1 roadrunner.hdmap.ParametricAttribution]
```

Figure 66: The first lane in the RRHD map now has all its connections.

Once we have cared for the lane groups, lanes, and lane boundaries, we can move on to the other objects such as signs and barriers.

Our script loops over the signs first. Each element in the signsOfInterest structure array contains all the necessary information for setting the signs along the roadway. Note, we did not add poles or trusses to these signs, but they could be added programmatically with a little extra coding logic and adding StaticObjectTypes to the RRHD map.

To place the sign in space, we must create a GeoOrientedBoundingBox object that contains information about the location of the center, dimensions, and orientation of the sign.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% get the sign geometry information %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
currGeom                = roadrunner.hdmap.GeoOrientedBoundingBox;
currGeom.Center         = currSign.Center;
currGeom.Dimension      = currSign.Dimension;
currGeom.GeoOrientation = currSign.GeoOrientation;
```

*Figure 67: Step 8 **Loop through signs and add to RRHD object**. Defining the sign geometry information.*

Then, we can create the reference for the sign and finally add the sign to the RRHD map.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% set the reference and the signs in rrhd map %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rrSignTypeRef      = roadrunner.hdmap.Reference(ID = rrRefID);
rrMap.Signs(end+1) = roadrunner.hdmap.Sign(ID = signID, Geometry = currGeom,...
                                           SignTypeReference = rrSignTypeRef);
```

*Figure 68: Step 8 **Loop through signs and add to RRHD object**. Adding Sign to RRHD.*

```
rrMap =

  roadrunnerHDMap with properties:

               Author: "kmcgarri"
         GeoReference: [42.3429 -71.2613]
    GeographicBoundary: [2×3 double]
                Lanes: [666×1 roadrunner.hdmap.Lane]
          SpeedLimits: [0×1 roadrunner.hdmap.SpeedLimit]
       LaneBoundaries: [914×1 roadrunner.hdmap.LaneBoundary]
           LaneGroups: [250×1 roadrunner.hdmap.LaneGroup]
         LaneMarkings: [13×1 roadrunner.hdmap.LaneMarking]
            Junctions: [0×1 roadrunner.hdmap.Junction]
         BarrierTypes: [3×1 roadrunner.hdmap.BarrierType]
             Barriers: [0×1 roadrunner.hdmap.Barrier]
            SignTypes: [21×1 roadrunner.hdmap.SignType]
                Signs: [175×1 roadrunner.hdmap.Sign]
    StaticObjectTypes: [0×1 roadrunner.hdmap.StaticObjectType]
        StaticObjects: [0×1 roadrunner.hdmap.StaticObject]
   StencilMarkingTypes: [0×1 roadrunner.hdmap.StencilMarkingType]
       StencilMarkings: [0×1 roadrunner.hdmap.StencilMarking]
    CurveMarkingTypes: [0×1 roadrunner.hdmap.CurveMarkingType]
        CurveMarkings: [0×1 roadrunner.hdmap.CurveMarking]
          SignalTypes: [0×1 roadrunner.hdmap.SignalType]
              Signals: [0×1 roadrunner.hdmap.Signal]
```

Figure 69: RRHD map with all signs added.

Next, we will loop over all the barriers of interest. Each element in the barriersOfInterest structure array contains all the necessary information for adding barriers along the roadway.

```
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % set the reference and the barriers in rrhd map %
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            barRef  = roadrunner.hdmap.Reference(ID = barriersOfInterest(bx).Type);
            rrMap.Barriers(end+1) = roadrunner.hdmap.Barrier(ID = barriersOfInterest(bx).BarrierId,...
                                                         BarrierTypeReference = barRef,...
                                                         Geometry = barriersOfInterest(bx).LocalCoords);
```

Figure 70: Step 9 ***Loop through barriers and add to RRHD object****.*

```
rrMap =

  roadrunnerHDMap with properties:

               Author: "kmcgarri"
         GeoReference: [42.3429 -71.2613]
    GeographicBoundary: [2×3 double]
                Lanes: [666×1 roadrunner.hdmap.Lane]
          SpeedLimits: [0×1 roadrunner.hdmap.SpeedLimit]
       LaneBoundaries: [914×1 roadrunner.hdmap.LaneBoundary]
           LaneGroups: [250×1 roadrunner.hdmap.LaneGroup]
         LaneMarkings: [13×1 roadrunner.hdmap.LaneMarking]
            Junctions: [0×1 roadrunner.hdmap.Junction]
         BarrierTypes: [3×1 roadrunner.hdmap.BarrierType]
             Barriers: [263×1 roadrunner.hdmap.Barrier]
            SignTypes: [21×1 roadrunner.hdmap.SignType]
                Signs: [175×1 roadrunner.hdmap.Sign]
    StaticObjectTypes: [0×1 roadrunner.hdmap.StaticObjectType]
        StaticObjects: [0×1 roadrunner.hdmap.StaticObject]
   StencilMarkingTypes: [0×1 roadrunner.hdmap.StencilMarkingType]
       StencilMarkings: [0×1 roadrunner.hdmap.StencilMarking]
    CurveMarkingTypes: [0×1 roadrunner.hdmap.CurveMarkingType]
        CurveMarkings: [0×1 roadrunner.hdmap.CurveMarking]
          SignalTypes: [0×1 roadrunner.hdmap.SignalType]
              Signals: [0×1 roadrunner.hdmap.Signal]
```

Figure 71: All barriers added to RRHD map.

If we wanted to add other stationary objects to the scene, we could do that now, or, as in the case of speed limits, which attach to lanes, we could add them as we build up the roadway itself. The pattern to do so is roughly the same as what we have outlined above. Junctions are the only thing that we need to carefully consider. For the most part, we don't need to do anything with junctions. RoadRunner will create junctions automatically when two or more roads come together and cross one another. In this example, we do not have a need to build up junctions.

Finally, we should save the RRHD map we have created, open an instance of RoadRunner, import the RRHD map, and build the road into a scene. From here, you may export the scene for use in any simulator, or you may move to RoadRunner Scenario to add vehicles and maneuvers for testing and validation.

```
361        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
362        % save the map to the output file %
363        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
364        rrhdFileNameAndLoc = rrProjLoc + "Assets\" + rrhdFileName;
365        write(rrMap, rrhdFileNameAndLoc);
366
367        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
368        % open roadrunner application %
369        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
370        rrApp = roadrunner(rrProjLoc, "InstallationFolder", rrInstallDir);
371
372        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
373        % import rrhd map and build scene %
374        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
375        rrApp.importScene(rrhdFileNameAndLoc, "RoadRunner HD Map");
```

Figure 72: Steps 12-14 ***Save RRHD map as an asset in your RoadRunner project directory****, open RoadRunner app, and import RRHD map.*

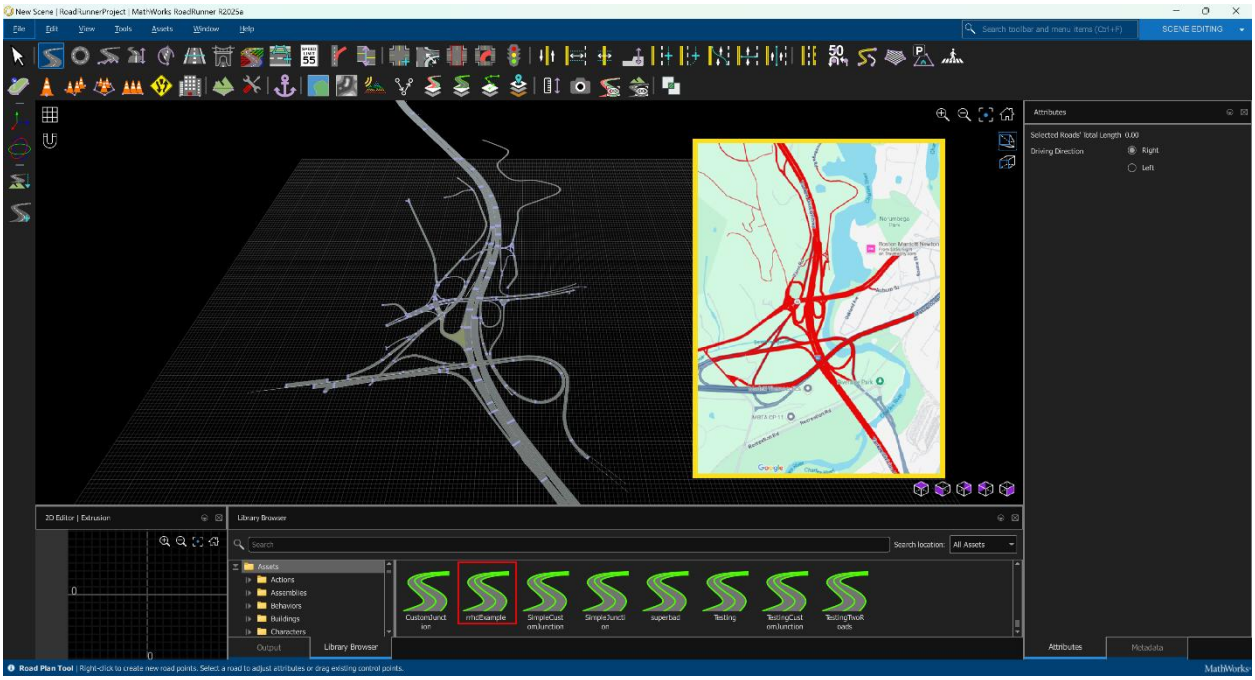


*Figure 73: Scene generated in RoadRunner from the RRHD map. Compare the road created in RoadRunner to the insert image from our original map viewed online (****Figure 38****).*

About the Author

Kim McGarrity is a senior consultant at MathWorks. She works on projects related to ADAS and Automated Driving, focusing on scene and scenario simulations using RoadRunner, and Unreal. Prior to joining MathWorks, Kim was an Algorithm engineer at Continental, GM, and ZF/TRW where she worked on developing ADAS features such as Blind Spot Detection, Trailer Merge Assist, and Valet Parking. Kim holds a BA in Physics from Concordia College, an MSc in Physics from Michigan State University, and a PhD in Statistics and Materials Science from TU Delft. MATLAB has been a strongly utilized tool across all these positions.