RoadRunner Scenario

User's Guide
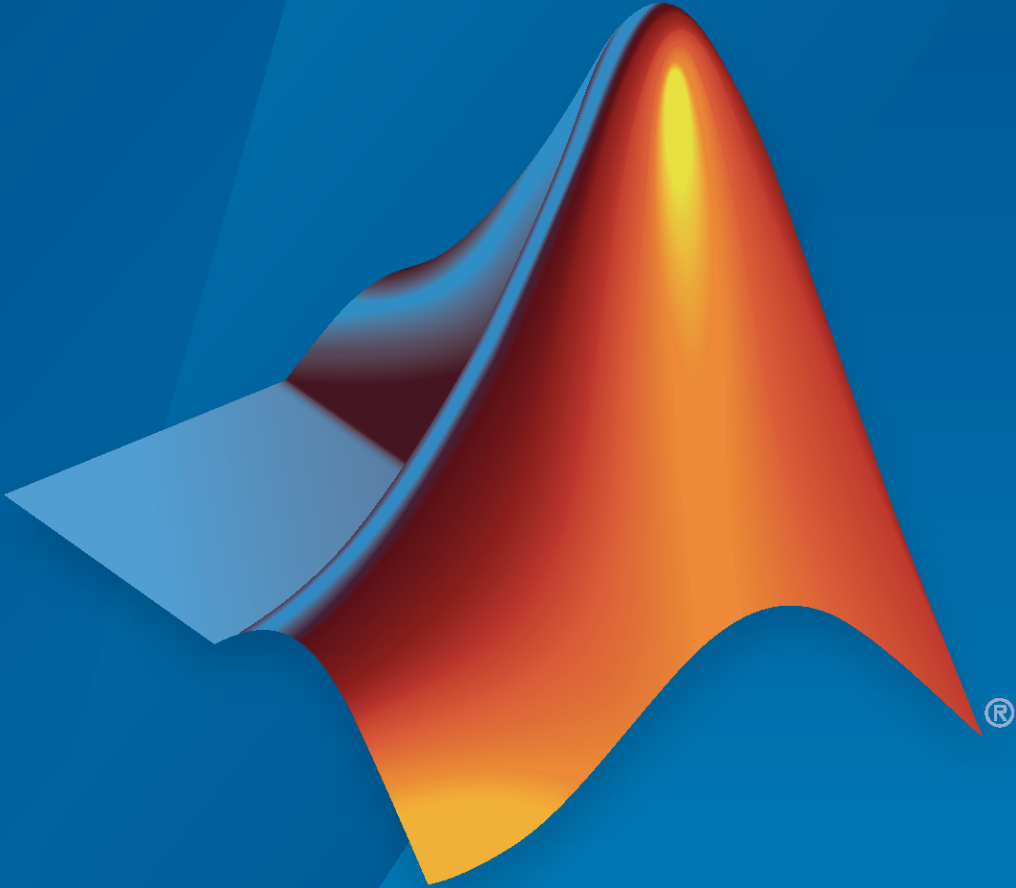
MathWorks®

How to Contact MathWorks

	Latest news:	www.mathworks.com
	Sales and services:	www.mathworks.com/sales_and_services
	User community:	www.mathworks.com/matlabcentral
	Technical support:	www.mathworks.com/support/contact_us
	Phone:	508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Revision History

Contents

Design and Simulate Scenarios

3

Programmatic Scenario Interfaces

4

Export Scenarios

5

Simulate Scenarios with CARLA

6

Get Started with RoadRunner Scenario

RoadRunner Scenario Product Description

Create and play back scenarios for automated driving simulation

RoadRunner Scenario is an interactive editor that lets you design scenarios for simulating and testing automated driving systems. Place vehicles and paths, define logic, and parameterize scenarios. You can then simulate the scenario in the editor. Choose from built-in vehicle actors or design your own using MATLAB® and Simulink® or CARLA.

RoadRunner Scenario supports in-editor playback for visualizing scenarios and connecting to other simulators for cosimulation. Scenarios can be exported to ASAM OpenSCENARIO®. These exported scenarios can then be used in any OpenSCENARIO compliant simulators and players, including esmini.

The RoadRunner API lets you change the parameters of the scenario to create many variants for automated tests. You can use the API to automate workflows like placing scenarios in different scenes, simulating scenarios, and exporting.

RoadRunner Scenario Fundamentals

RoadRunner Scenario is an interactive editor that enables you to design scenarios for simulating and testing automated driving systems. You can place vehicles, define their paths and interactions in the scenario, and then simulate the scenario in the editor. You can also generate multiple variations of scenarios programmatically, export them to ASAM OpenSCENARIO, and simulate and visualize them in automated driving simulators and players.

If you design an actor with autonomous behavior in an external simulator such as CARLA, you can assign that actor behavior to a vehicle in your scenario. You can then cosimulate that actor in RoadRunner Scenario and the external simulator. You can also cosimulate actors designed in MATLAB and Simulink and analyze simulation results using MATLAB and Simulink tools.

Scenes vs. Scenarios

RoadRunner Scenario is an add-on product that requires an addition to your RoadRunner license. The RoadRunner base product enables you to design scenes, which are composed of static elements, such as roads, lanes, terrain, and traffic signals. This figure shows a sample intersection scene with the static elements that you can edit.
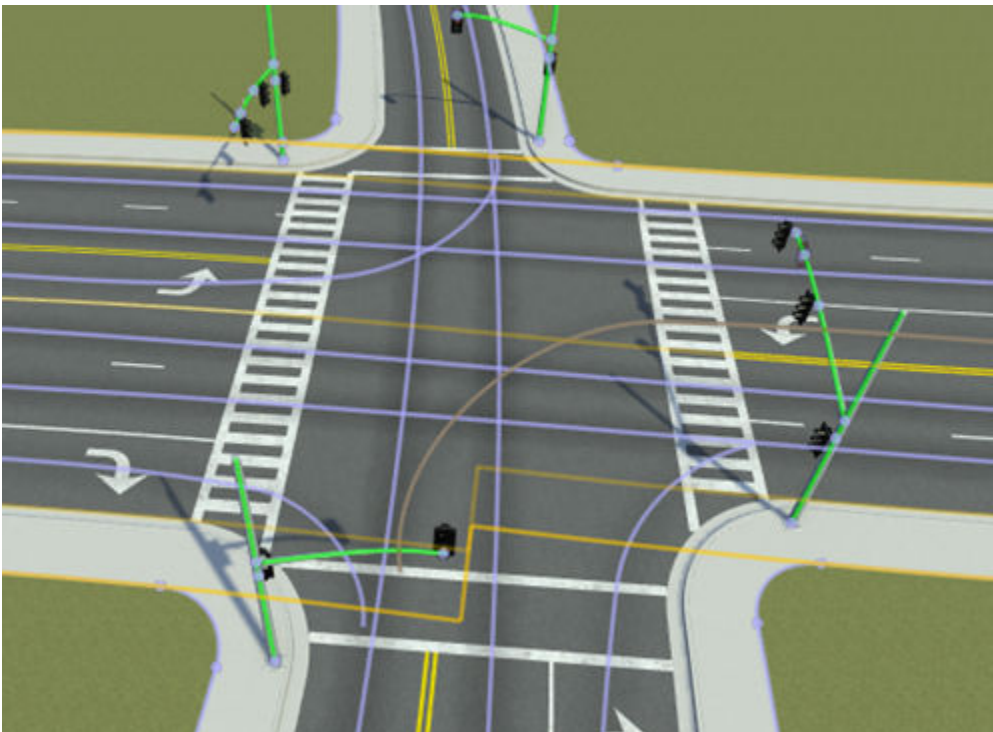
RoadRunner Scenario enables you to design scenarios, which are composed of dynamic elements such as moving vehicles. Scenarios are built on top of scenes: to create a scenario, you must first create a scene. This image shows a sample scenario built off of the scene from the previous image.
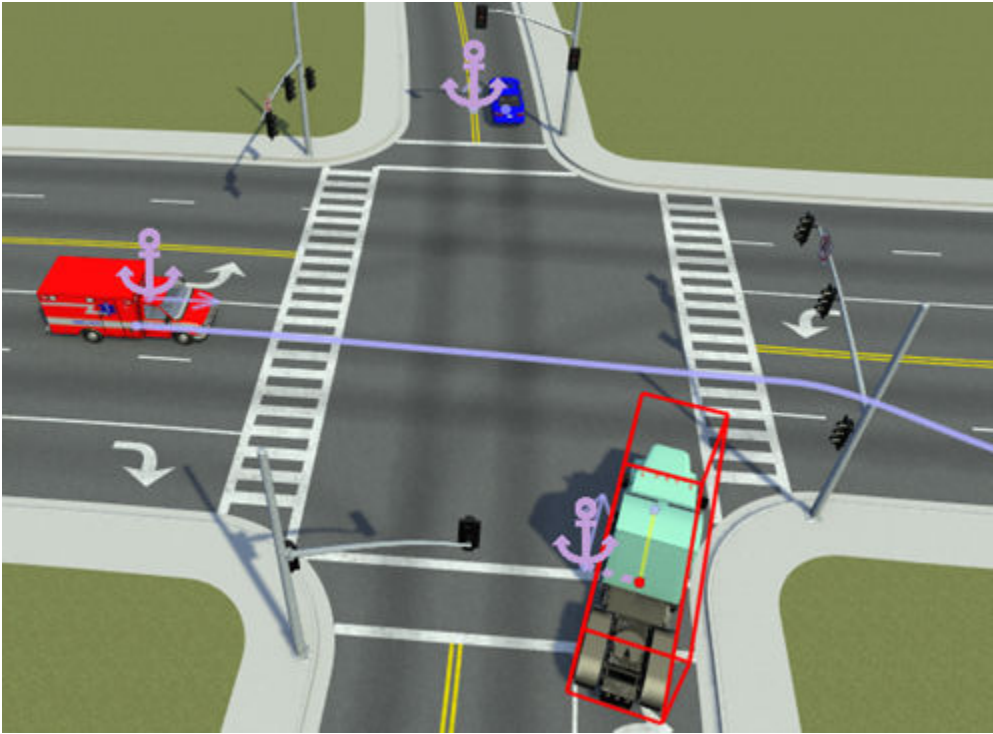
In RoadRunner, only the static elements are visible and editable. In RoadRunner Scenario, the scene is visible but only the dynamic elements are editable. To switch between scene and scenario editing, use the **Scene Editing** (RoadRunner) and **Scenario Editing** (RoadRunner Scenario) toggle in the upper-right corner of the application.
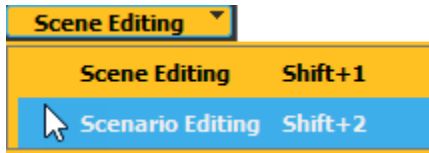
This toggle is available only if you have a RoadRunner Scenario license. Otherwise, RoadRunner is by default in scene editing mode.

Design and Simulate Scenarios

In RoadRunner Scenario, the scenarios you design are composed of actors, which are the objects in motion within a scenario. Vehicles are one type of actor and a key component of driving scenarios. The **Vehicles** folder of the **Library Browser** contains a variety of vehicle assets that you can drag into your scenario.
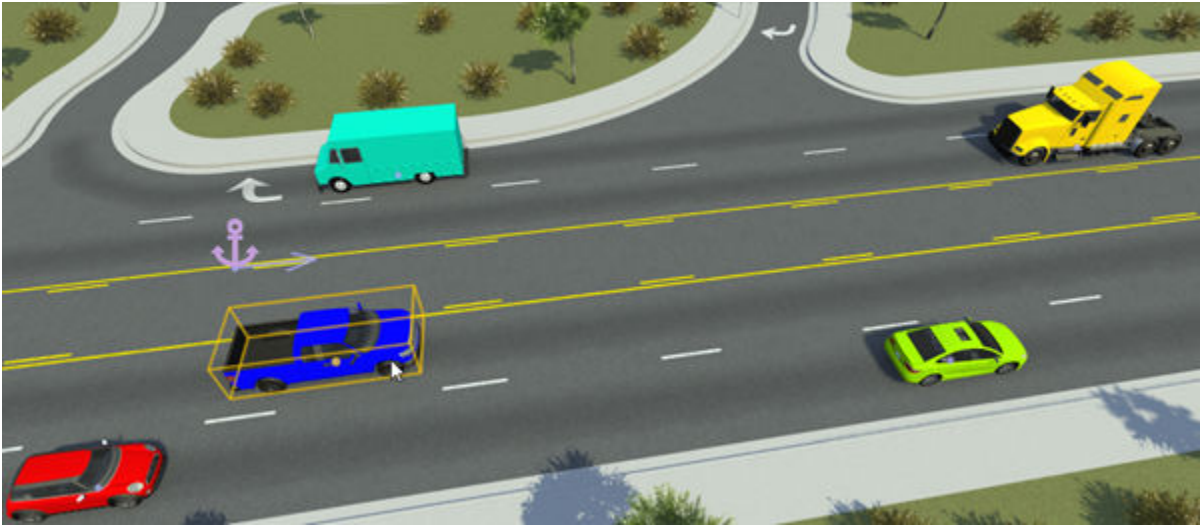
By selecting an actor and right-clicking within the scenario, you can define paths for the vehicle to follow. By default, paths snap to lanes, enabling you to quickly create complex paths such as turns and lane changes. For additional flexibility, you can change the shape of paths by modifying tangent waypoints, shift paths laterally within a lane, and specify free-form paths that go off-road or disobey traffic laws.

If you do not specify a path, then actor fall back on their built-in behavior, which for vehicles added to roads is to drive in their current lane at a constant speed.

To define the interactions between actors, you use the graphic **Logic** editor, which is available in the **2D Editor** pane.

The scenario logic defined in this editor consists of a series of actions with optional conditions that trigger those actions. Actions and conditions can also occur in parallel, enabling you to build complex scenarios containing multiple actors that have different goals. For example, this sample logic scenario shows a vehicle changing lanes and changing speeds when it reaches a certain distance from another vehicle.
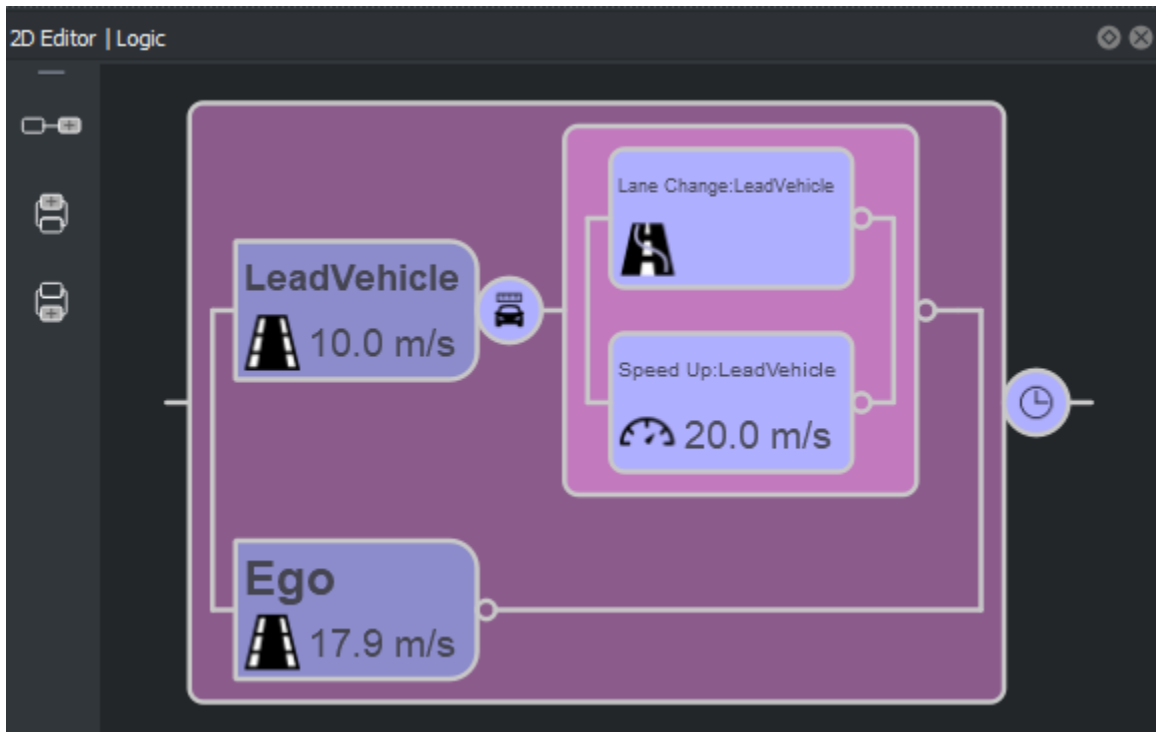
To test your scenarios, RoadRunner Scenario provides an in-editor simulation tool. From the toolbar, select the **Simulation Tool** to enable simulation controls. Using the controls in the **Simulation** pane, you can control the pacing and step size of the scenarios.
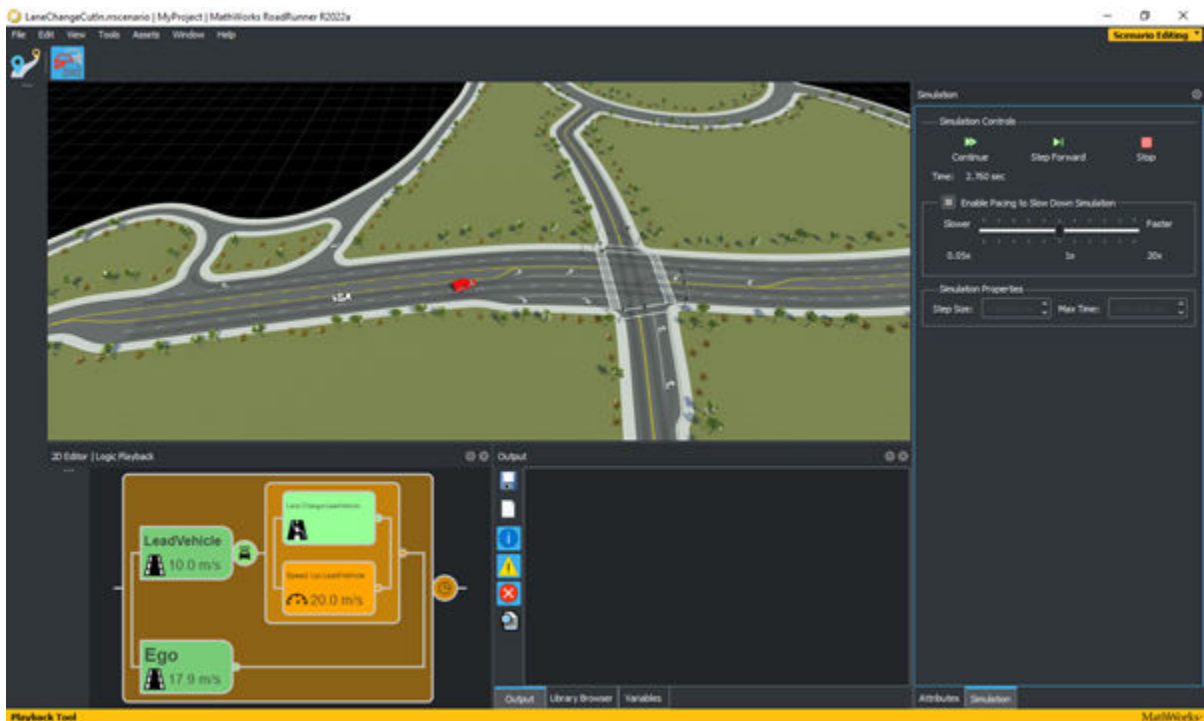
During the design and simulation process, RoadRunner Scenario provides validation feedback on your scenarios. This feedback comes in the form of visual indicators while designing and reports in the **Output** pane after simulation. Use this validation feedback to fix validation errors early, instead of waiting to catch them during the more time-consuming export process.

Export and Import Scenarios

After you design your scenario, you can export it to a supported format such as ASAM OpenSCENARIO V1.x and V2.0. You can use these exported scenarios in automated driving simulators and players. For example, this figure shows a visualization of a scenario exported to ASAM OpenSCENARIO V1.0 in esmini.

If you have scenarios defined in supported formats such as ASAM OpenSCENARIO V1.0, you can import those scenario files into RoadRunner Scenario, modify the scenarios in the editor, and then re-export them.

You can export and import files interactively or programmatically. Using either MATLAB functions or a language-neutral API built on gRPC® technology, you can programmatically export or import hundreds of files in the language of your choice. Use of the MATLAB functions requires Automated Driving Toolbox™.

Generate Scenario Variations

By taking a single scenario and varying certain aspects of it programmatically, you can quickly generate and export hundreds or even thousands of scenarios on which to test autonomous vehicle

algorithms. For example, you can vary the distances between vehicles in a scenario, or vary vehicle speeds, colors, or types.

To create a variable, right-click a scenario attribute in the **Attributes** pane and select **Create Variable**. For example, this figure shows a vehicle speed variable being created.
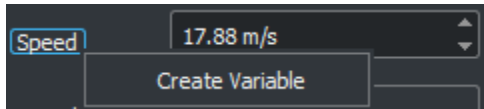
These values appear in the **Variables** pane.
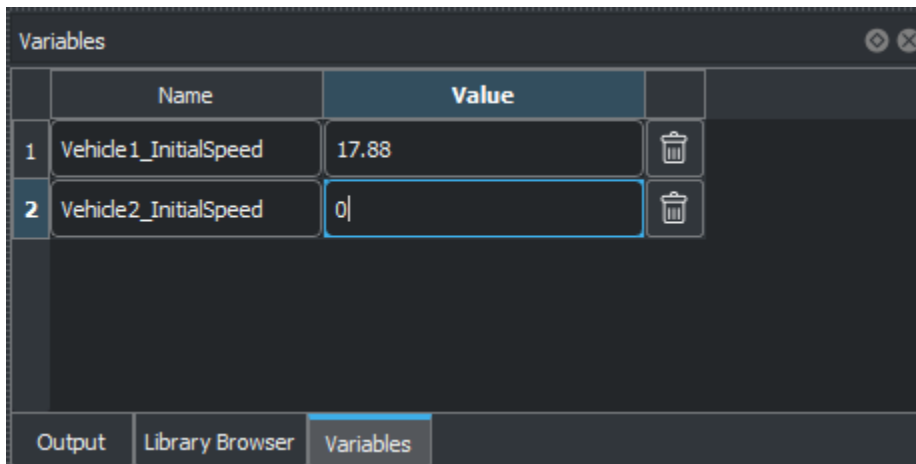
You can then modify these variables using the RoadRunner gRPC API or corresponding MATLAB functions (requires Automated Driving Toolbox) and export multiple variations of a scenario programmatically.

You can also vary scenarios by relocating them within a scene. The anchoring system used in RoadRunner Scenario makes it easy to relocate scenarios. By dragging an anchor point around in a scenario, all objects anchored to that point move with it.

In a scenario, an anchor can be:

- A point on a road or junction.
- A point specifying the location of an actor.
- A path waypoint.

You can assign an anchor to any object in a scenario. For example, this figure shows vehicles anchored to each other, with all vehicles anchored directly or indirectly to a road anchor in lavender.
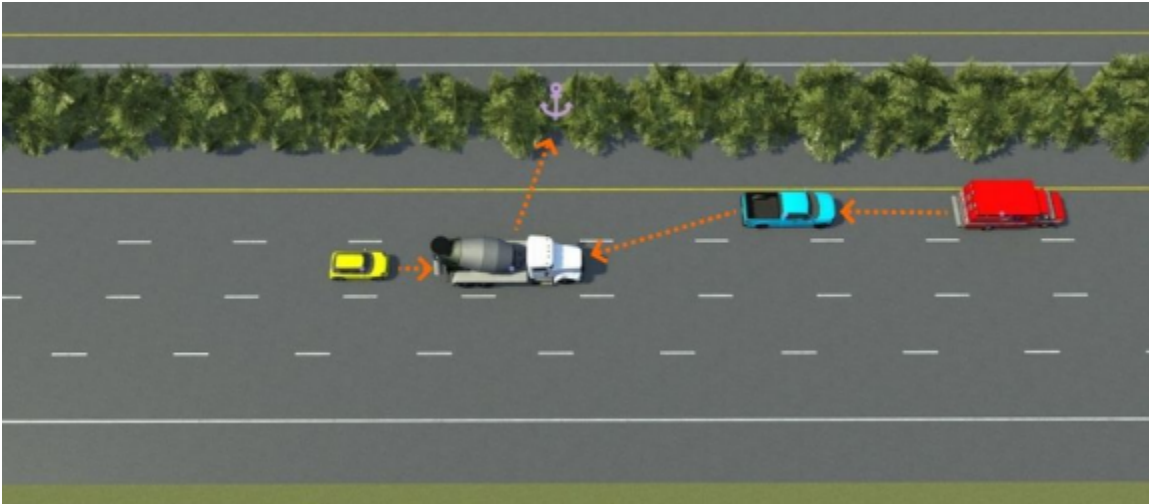
- Move objects, such as vehicles, relative to each other, either along lanes or into different lanes.
- Move an entire scenario to a new scene location by dragging a single anchor point.
- Quickly change the travel directions of all vehicles in a scenario.
- Align vehicles with each other or with an anchor point.
- Change the shape of a road while having vehicles maintain their relative positions.
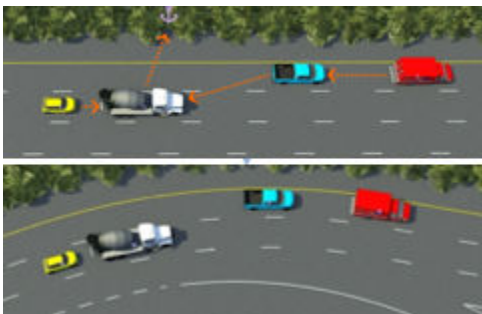
If multiple scenes have a road anchor with the same name, you can also relocate an scenario into an entirely new scene. In each scene, the scenario is positioned relative to the road anchor.
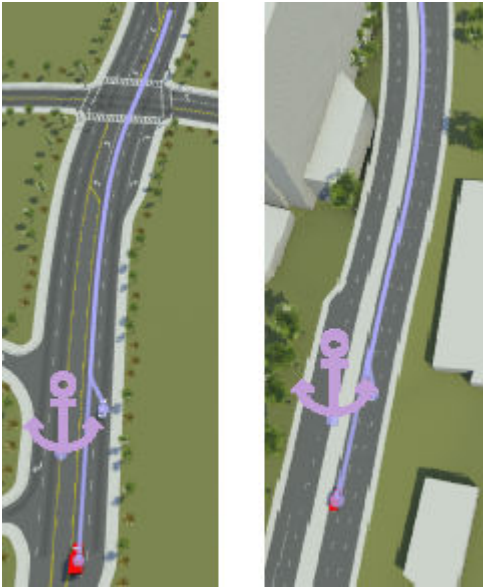
RoadRunner provides both interactive and programmatic ways to relocate scenes within scenarios. Using the programmatic option, you can quickly generate a scenario from multiple scenes.

Simulate Actors with MATLAB and Simulink

You can author RoadRunner actors using MATLAB System object™ and Simulink, associate actor behavior in RoadRunner scenario, and simulate the scenario.

These are the steps of the workflow for simulating RoadRunner Scenarios with MATLAB and Simulink:

- Author MATLAB System objects or Simulink models to define actor behavior.
- Associate actor behavior in RoadRunner.
- Optionally, publish an actor behavior.
- Tune the parameters defined in MATLAB or Simulink for RoadRunner simulations.
- Simulate a scenario using the RoadRunner user interface or control simulations programmatically from MATLAB.
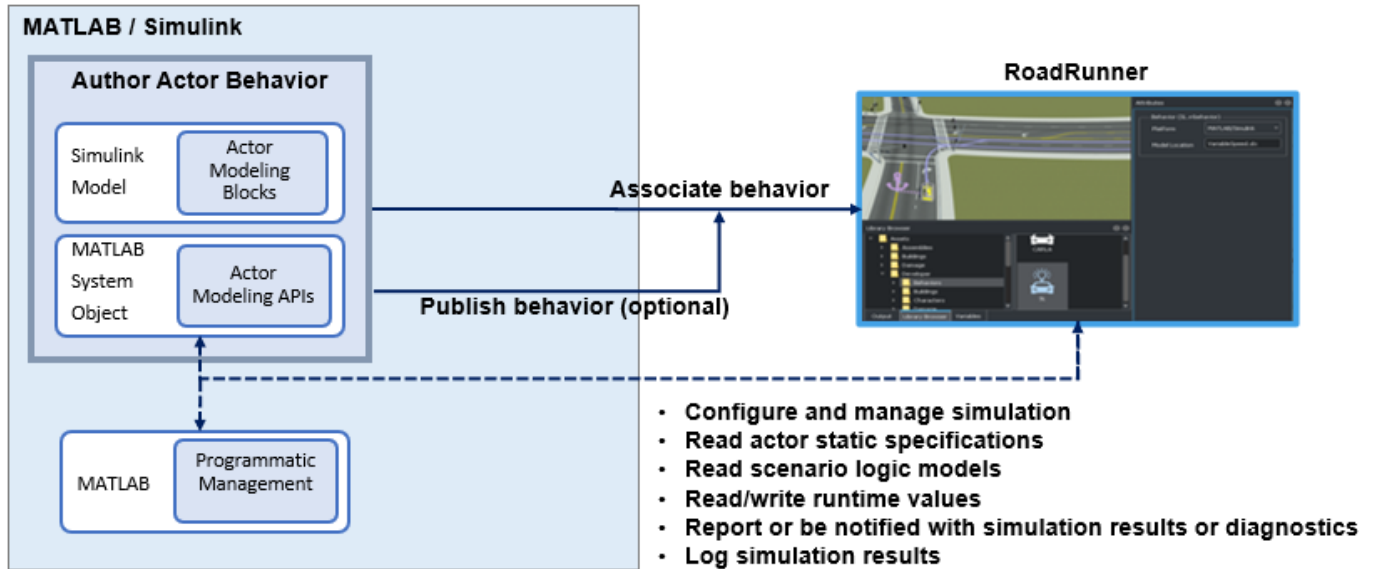
Cosimulate Actors with CARLA

You can cosimulate a scenario in RoadRunner Scenario and in an external simulator, such as CARLA. Using an external simulator, you can control vehicle actors to test your automated driving algorithms.
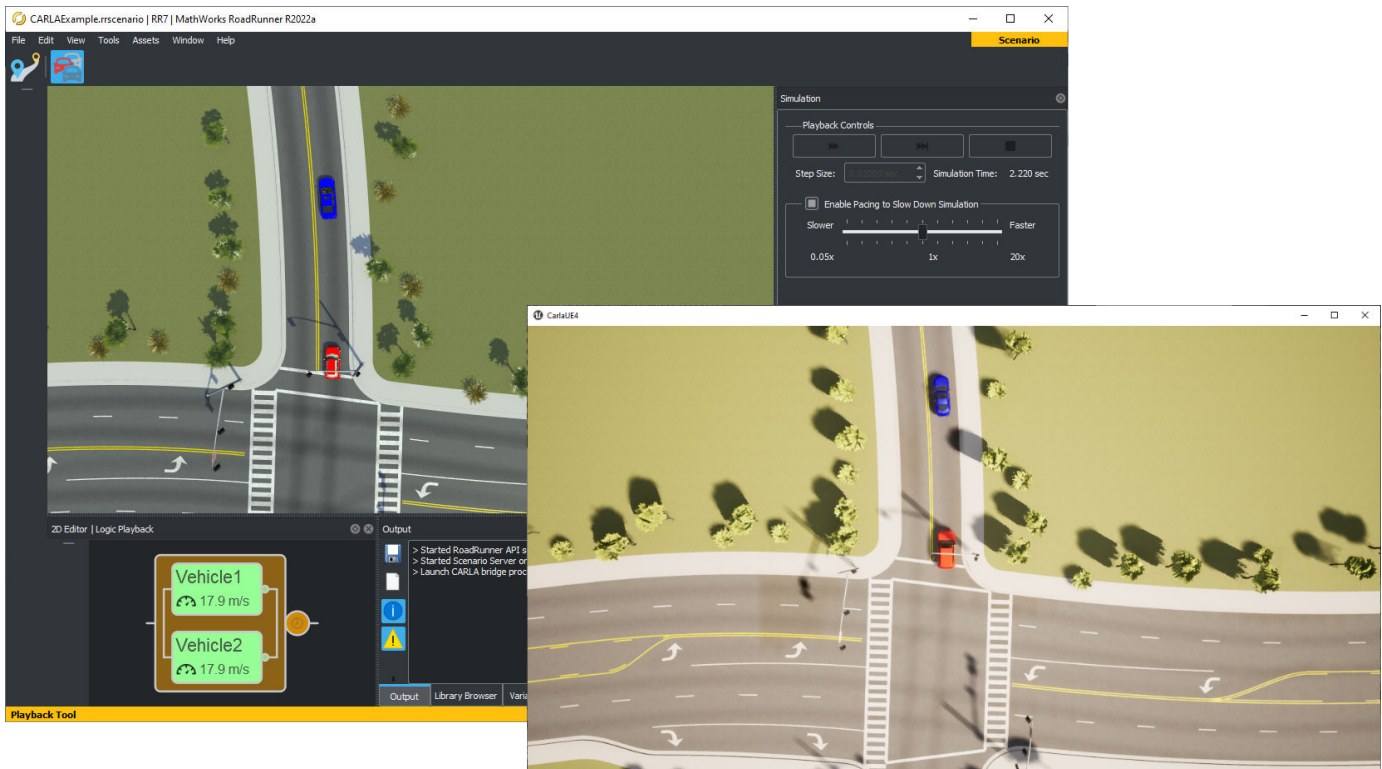
See Also

Scenario Edit Tool | **Simulation Tool** | **Vehicle Assets**

Related Examples

- "Explore and Simulate a Simple Scenario" on page 1-13

More About

- "Switch Between Scene and Scenario Editing" on page 3-58
- "Path Editing" on page 3-62
- "Built-In Behavior for Vehicles" on page 3-128
- "Define Scenario Logic" on page 3-70
- "Validate Scenarios" on page 3-123

Explore and Simulate a Simple Scenario

In this example, you explore a sample scenario in the RoadRunner Scenario interactive editor to learn the basics of scenario design and simulation. This example assumes that you have prior knowledge of working with RoadRunner, and that you have already created a project. For more details, see "Get Started with RoadRunner" and "RoadRunner Project and Scene System". If this is your first time working with RoadRunner Scenario, consider reading "RoadRunner Scenario Fundamentals" on page 1-3 first.

Open Scenario

Open the sample scene and scenario used in this example.

1. Open the RoadRunner application and, from the start page, select **Open Scene**.

2. Navigate to the `Scenes` folder of the current project and select the `ScenarioBasic.rrscene` scene. RoadRunner Scenario opens the scene in the editing canvas.

3. Switch to scenario editing mode. From the top-right corner of the RoadRunner application, select **Scene Editing**, then **Scenario Editing**.
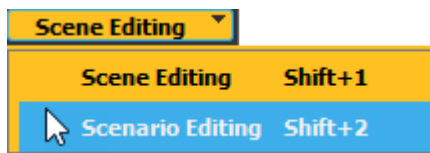
4. From the **File** menu, select **Open Scenario into Current Scene**.

5. Navigate to the `Scenarios` folder of the current project and select the `TrajectoryCutIn.rrscenario` scenario. RoadRunner Scenario opens the scenario in the editing canvas.
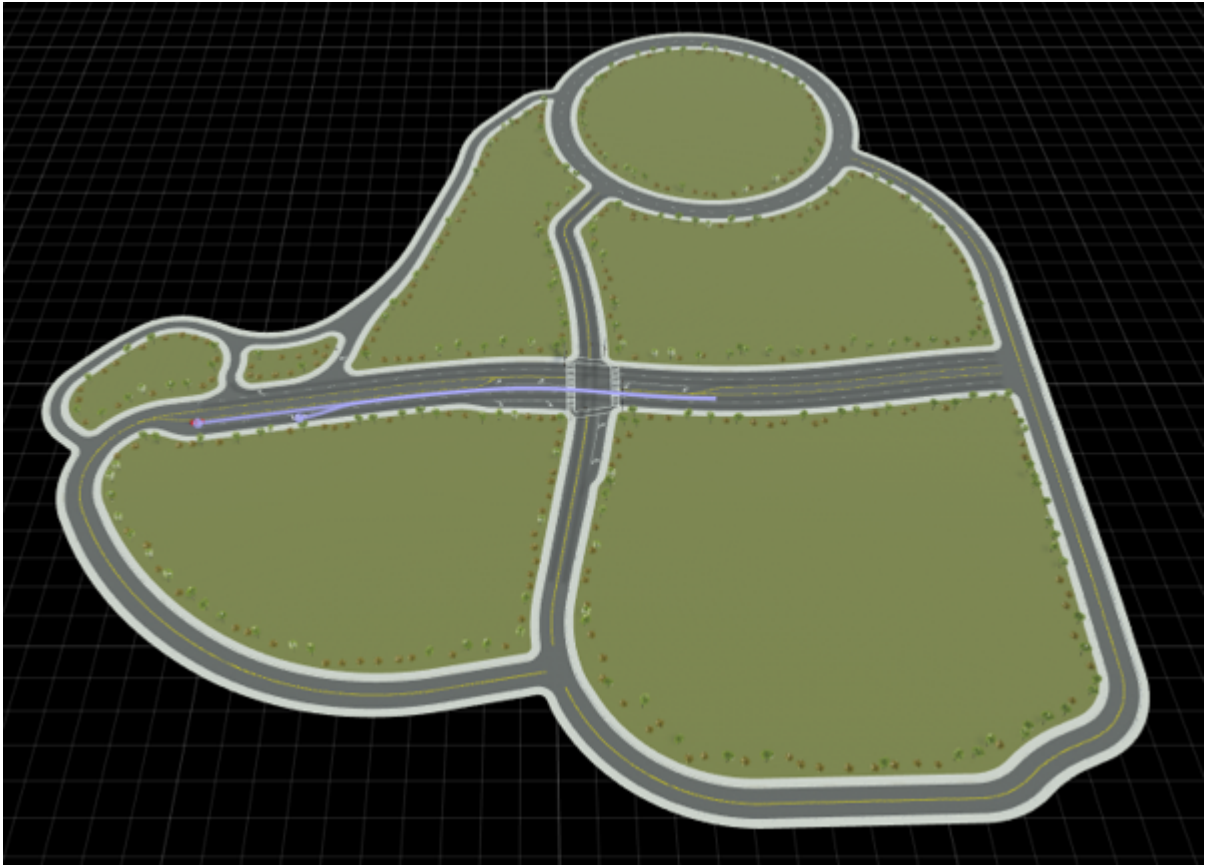
> **Tip** To avoid overwriting the original scenario, create and work with a copy of the scenario. From the **File** menu, select **Save Scenario As**. Name the scenario `MyScenario.rrscenario`.

In **Scenario Editing** mode, the dynamic scenario elements, such as the vehicles and their paths, are active for editing. The static scene elements, such as the roads and trees, are locked. To switch between editing the scenario and the scene, use the yellow **Scene Editing** or **Scenario Editing** toggle in the upper-right corner of the RoadRunner application.

Simulate Scenario

This scenario contains a white sedan named `Ego` and a red sedan named `Lead`. The vehicles follow predefined driving paths during simulation. Simulate the scenario and observe how the vehicles interact.

1
From the RoadRunner Scenario menu, click the **Simulation Tool** on the RoadRunner Scenario toolbar.

2 In the **Simulation** pane on the right, click **Play**.

> **Note** If the simulation runs too quickly, use the slider in the **Simulation** pane to slow down the pace of the simulation.
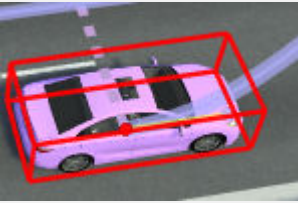
In this scenario, the red sedan begins driving immediately. The white sedan begins driving after some delay and then cuts into the lane of the red sedan. Shortly after the cut-in, the red sedan speeds up and, upon reaching a target speed, slows to a stop. The white sedan, after the cut-in, drives behind the red sedan and matches the speed of the red sedan until it comes to a stop.
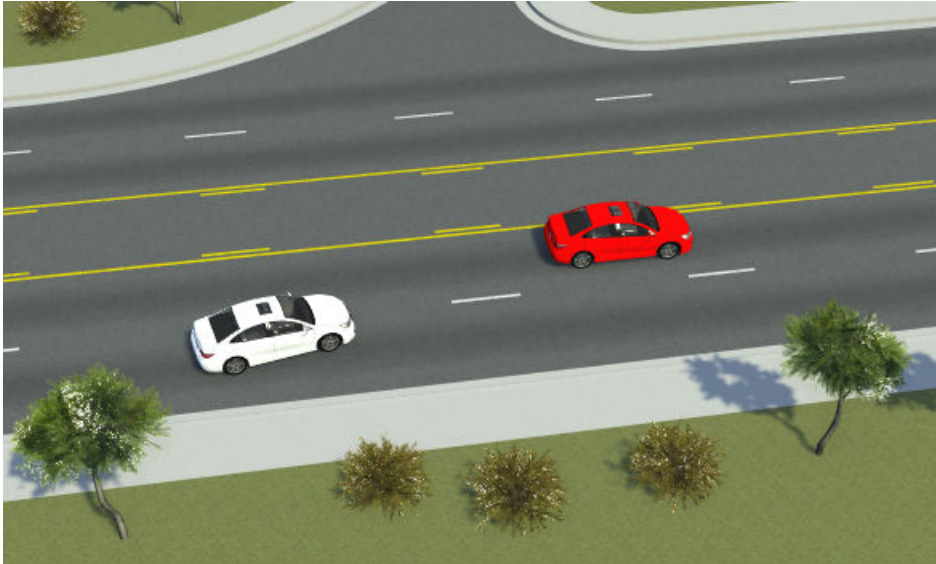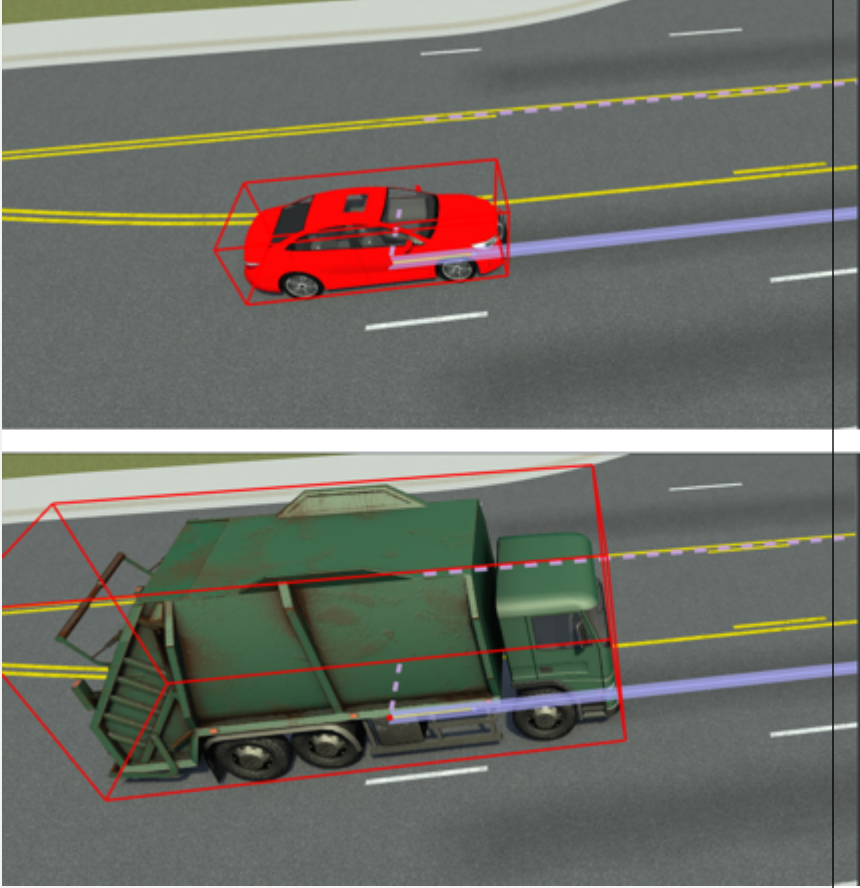
3   During simulation, the scenario elements become locked for editing. When you are done

simulating, switch back to scenario editing mode by clicking the **Scenario Edit Tool** on the RoadRunner Scenario toolbar.

Modify Vehicles

Vehicles in RoadRunner Scenario are available from the `Vehicles` folder of the **Library Browser**. Try adding or modifying vehicles, and then switching to the **Simulation Tool** to observe the effects on simulation. The table lists some sample vehicle modifications.
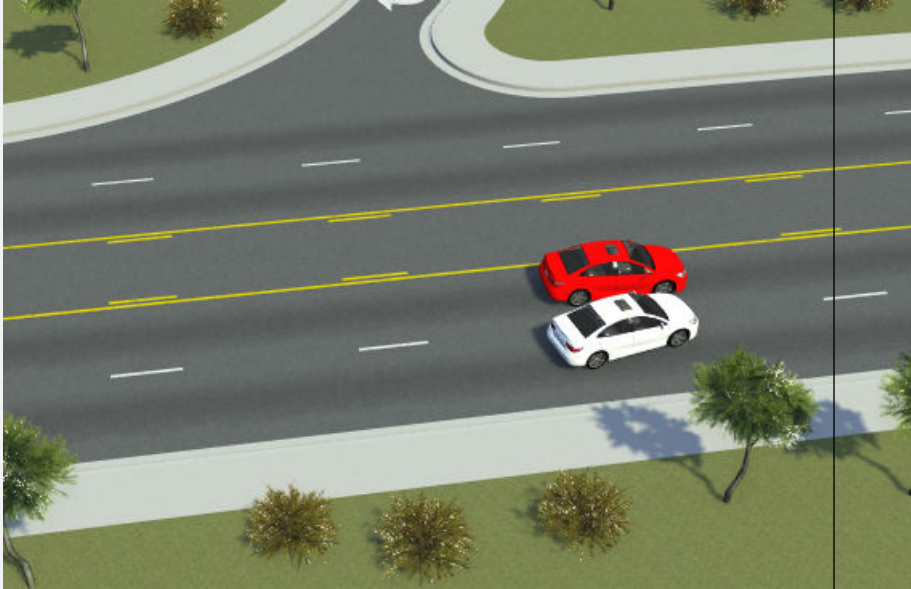
Description	Example
Change the color of a vehicle by selecting a vehicle and then selecting a new color from the **Attributes** pane. **Note** Not all vehicles have their colors visualized in the scenario editor, but their color values are included on export.	This image shows a sedan that has had its color changed from white to pink.
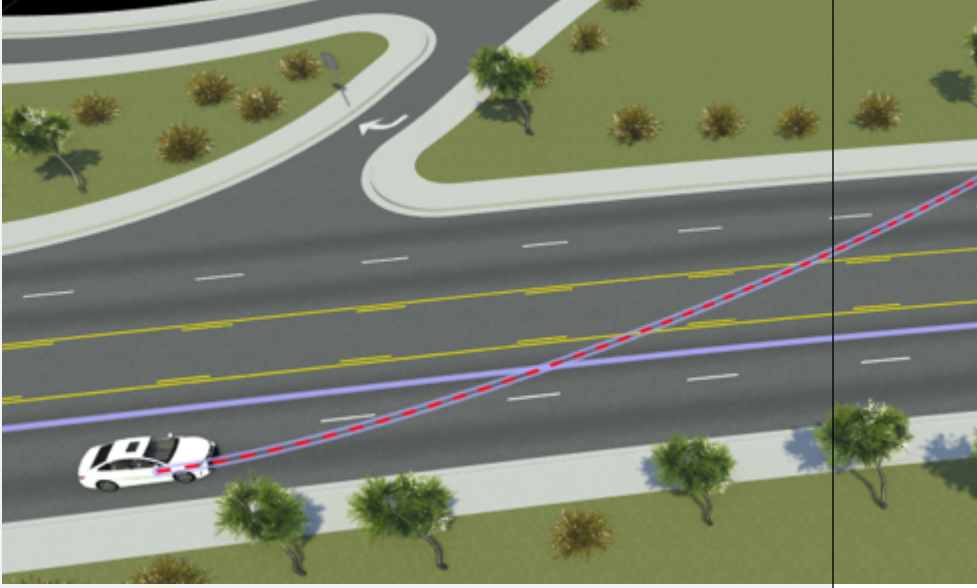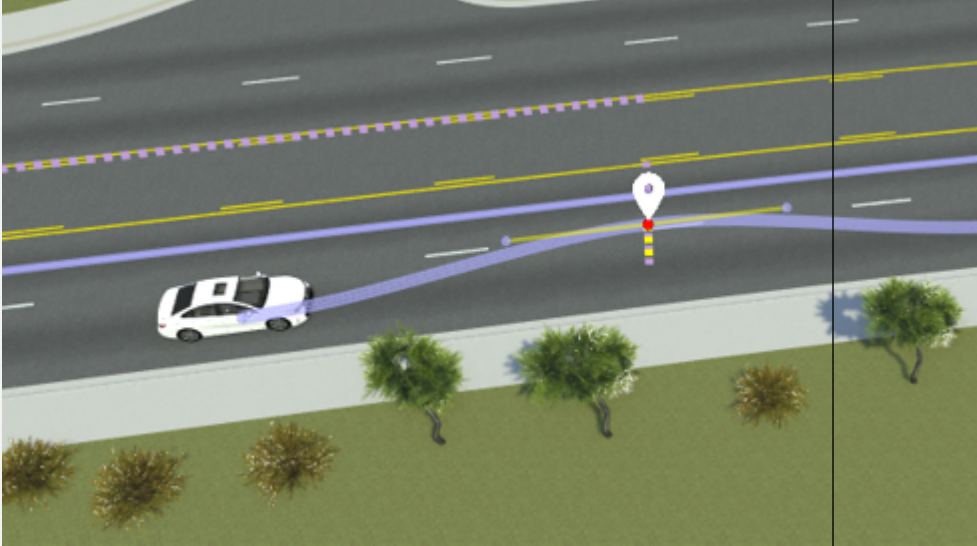
Description	Example
Change the type of a vehicle by selecting a vehicle and then dragging a new vehicle asset from the **Library Browser** onto the **Vehicle Type** image thumbnail in the **Attributes** pane. Additional vehicle types are available with **Vehicle Assets**.	This image shows the red sedan changing to a garbage truck.
Add a new vehicle by dragging a vehicle asset from the **Library Browser** into the scenario editing canvas. During simulation, vehicles without driving paths follow their lanes by default.	This school bus drives on its own and turns right at the exit during simulation.

Modify Driving Paths

When you select a driving path, the path turns red. The final waypoint appears as a yellow pinpoint icon, and intermediate waypoints appear as white pinpoint icons. Try modifying the driving paths in

the scenario, and then switch to the **Simulation Tool** to observe the effects on simulation. The table lists some sample driving path modifications.

Description	Example
Click and drag vehicles to change their starting position in the simulation. The driving paths automatically update to reflect the road and lane shapes at the new starting position.	Dragging the white sedan changes the point at which it starts its cut-in, which can cause the simulation to end in collision.
Change driving paths by clicking and dragging the last waypoint in the path. By default, driving paths follow the center lanes and follow the traffic laws of the road network, so moving the path to certain lanes can dramatically change its length. To extend a path with new waypoints, select the path and then right-click in the scenario. You can also split a path by right-clicking within a path segment to form new waypoints.	When you move the last waypoint of the red sedan into an opposing lane, the path makes a loop so that the sedan can turn around and drive in the opposite direction.

Description	Example
Create free-form paths by selecting path segments and then selecting **Freeform** in the **Attributes** pane.	If you specify a path segment as free-form, then you can specify a path using a series of waypoints. You can have the path move into an opposing lane without obeying traffic laws or have the path go off the road.
Change the position of a vehicle within its lane by selecting a path waypoint and updating the **Lateral Offset** value in the **Attributes** pane.	The white sedan veers to the left within its lane. It has a path waypoint with a **Lateral Offset** value of ‑2 meters (2 meters to the left of the lane center).

Modify Scenario Anchors

RoadRunner Scenario uses an anchoring system to specify the relative positions of objects. By moving objects designated as anchors, you can move an entire scenario to a new location in a scene.

Hide and show anchors for all actors in the scenario by clicking **Show All Anchors** ⚓ on the RoadRunner Scenario toolbar. Anchors are always displayed when you select an entity that references them.

When you add objects to a road, they become attached to road anchors. In this scenario, the vehicles are anchored to the `ScenarioStart` anchor on the left side of the intersection. Try dragging this anchor. The vehicles and their paths move along with it.
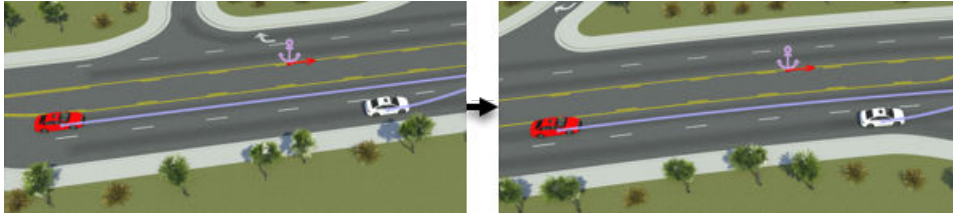
An anchor can also be a path waypoint or another vehicle. For example, select the white sedan and observe that, in the **Attributes** pane, its **Anchor** is set to the `Lead` actor. Now, try dragging both vehicles.

- When you drag the red sedan, the white sedan moves with it, because the red sedan is the parent anchor of the white sedan.

- When you drag the white sedan, the red sedan does not move with it.

Modify Scenario Logic

In the **2D Editor** pane, the **Logic** editor defines the interaction between vehicles in the scenario. The graphical interface consists of various action phases and the conditions that trigger those action phases. During simulation, the **Logic** editor displays the status of the actions and conditions. Green indicates completed actions and conditions, orange indicates active ones, and gray indicates ones that have not run.
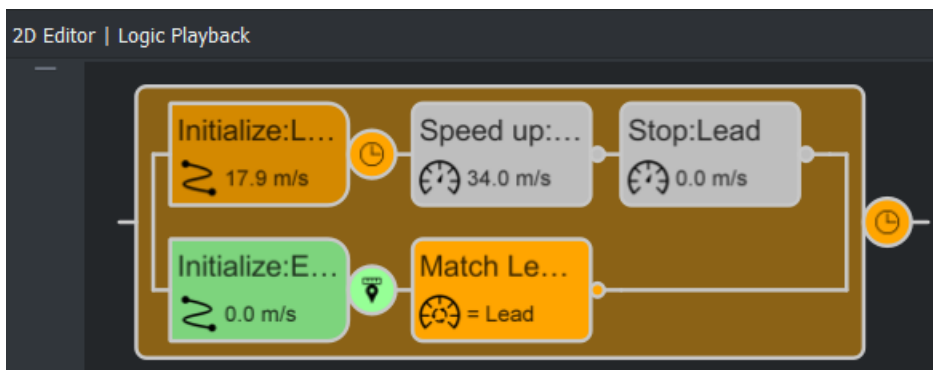
Explore the aspects of the scenario logic.

Initialization Phases

The first phases in the **Logic** editor are initial phases. They define the actions that the vehicles take at the start of simulation.
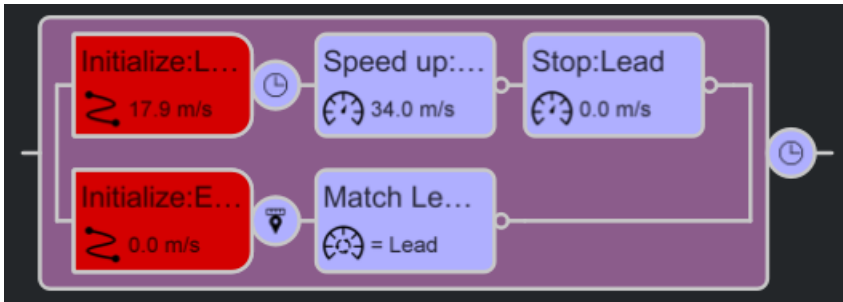
Click the two initial phases and observe the attributes of the phases in the **Attributes** pane. Both vehicles include an `Initialize Speed` action. Try changing the initial **Speed** values of the vehicles and observe the effects on simulation.

Condition Phases

The circular nodes specify conditions that must be met before the vehicles can start their next actions. You can add a condition after any phase.
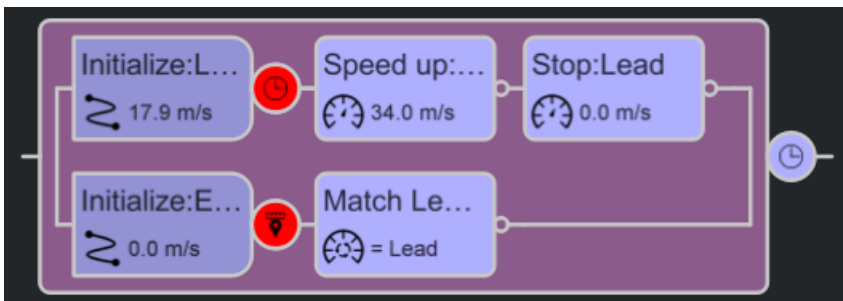
Select the condition node for the red sedan. The red sedan has a `Simulation Time` condition with the **Time** attribute set to 2 seconds. This condition means that at the start of simulation, the vehicle waits until two seconds have elapsed before performing its next action. Try increasing this value and observing the effects on simulation. For example, try setting **Time** to 10 seconds. The white sedan now completes its cut-in maneuver and matches the speed of the red sedan before the red sedan speeds up, as reflected by this scenario logic.
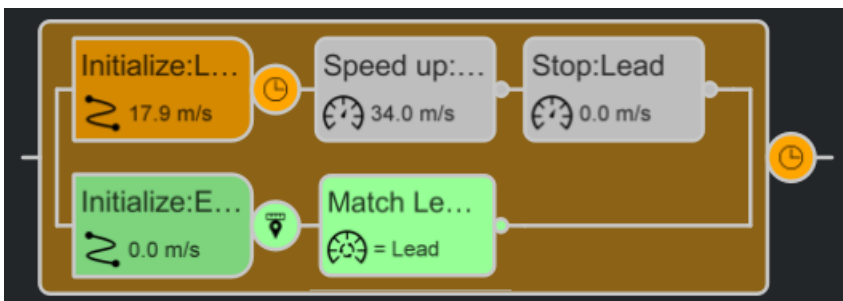
Now select the condition node for the white sedan. The white sedan has a `Distance to Point` condition. The scenario canvas displays this point, which is a point along the path of the red sedan. When the red sedan reaches any point within this radius, the white sedan begins its next action.
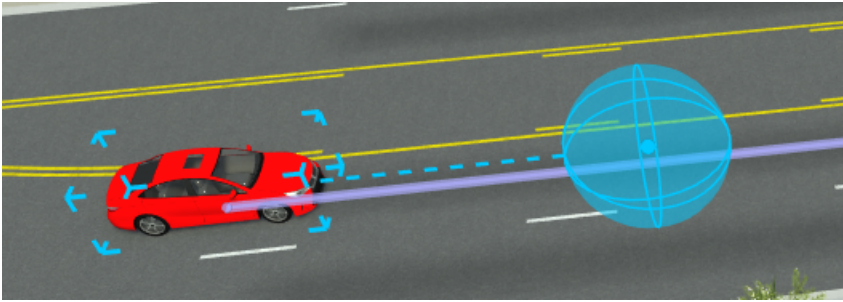
Try modifying the attributes for this condition to see how they affect the simulation. Moving the path waypoint that triggers the condition can also change the condition.

You can also try adding additional conditions by right-clicking any node after an action phase and selecting a condition from the **Attributes** pane. For example, try adding a `Duration` condition after the speed-up phase of the red sedan, and observe how the simulation changes.

You can add a condition, including a fail condition, after the entire simulation.
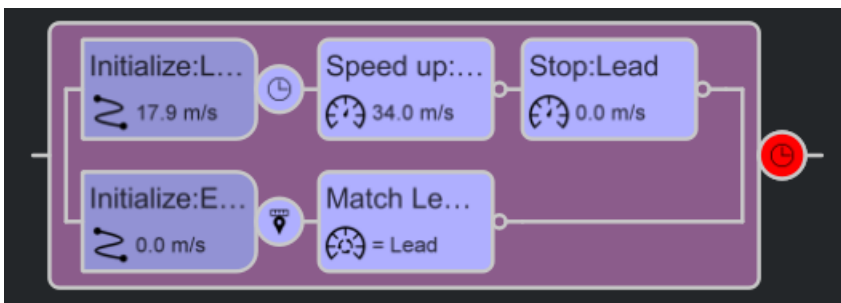
Action Phases

The subsequent phases after the conditions are action phases. This scenario contains only speed change actions.
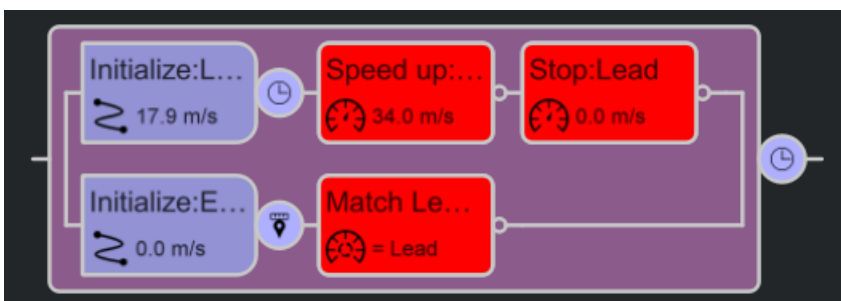
Select the action phase for the white sedan. The **Attributes** pane displays the attributes for the `Change Speed` action. Try modifying these attributes and observing the effects on simulation. For example:

- Set **Direction** so that the white sedan drives at a constant rate faster or slower than the red sedan.
- Under **Dynamics**, set **Time** to `10` seconds so that the white sedan takes longer to change its speed to match the red sedan.

You can change the units of speed and acceleration used by the vehicle actors in RoadRunner Scenario. You can select from these units of speed:

- **Feet per second (ft/s)**
- **Kilometers per hour (km/h)**
- **Meters per second (m/s)**
- **Miles per hour (mph)**

You can select from these units of acceleration:

- **Feet per second squared (ft/s^2)**
- **g-force (g)**
- **Kilometers per hour per second(km/h-s)**
- **Meters per second squared (m/s^2)**
- **Miles per hour per second (mph/s)**

To access these unit options, in **Scenario Editing** mode, from the RoadRunner Scenario toolbar, select **Edit**, then **Preferences** . The changes in speed and acceleration units are reflected in the **Attributes** pane for the vehicle assets and action phases.

You can also try selecting different actions, or adding new actions by right-clicking an action phase and adding it after, above, or below the selected phase. Actions above and below a phase occur in parallel with the selected phase.

See Also
Vehicle Assets

Related Examples

More About

Import Scenario Data

Import Trajectories from ASAM OpenSCENARIO Files

ASAM OpenSCENARIO is an open file format that describes the dynamic content for an automated driving simulation. Using RoadRunner Scenario, you can import scenarios from an ASAM OpenSCENARIO version 1.0 file.

RoadRunner Scenario supports importing initial target positions, initial speeds, and polyline trajectories of vehicles specified in the `Init` section of an ASAM OpenSCENARIO 1.0 file. If the input ASAM OpenSCENARIO 1.0 file specifies absolute time domain values of trajectory waypoints using the `TimeReference` element of the `FollowTrajectoryAction` action, then RoadRunner Scenario imports trajectory waypoints with the timing data specified in the `Vertex` elements of that action. Otherwise, RoadRunner Scenario ignores the specified time domain values while importing trajectory waypoints.

Note RoadRunner Scenario does not import any elements defined in the `Story` section of an ASAM OpenSCENARIO 1.0 file.

Import ASAM OpenSCENARIO File Interactively

To import scenarios from ASAM OpenSCENARIO file using the RoadRunner Scenario user interface, follow these steps:

1   From the **Tools** menu, select **Scenario Editing** to switch to scenario editing mode.

2   From the **File** menu, select **Import**, then **ASAM OpenSCENARIO 1.0 Trajectories**.

3   In the Import ASAM OpenSCENARIO dialog box, specify the **File path** by navigating to a directory containing a valid ASAM OpenSCENARIO 1.0 file of type `.xosc`. The directory must also contain all the associated files, such as an ASAM OpenDRIVE® file and catalog files.

4   Click **Import**.

Tip To avoid any conflicts with an existing scene or scenario, best practice is to clear any existing scene or scenario before importing an ASAM OpenSCENARIO file.

Import ASAM OpenSCENARIO File Programmatically

To import the ASAM OpenSCENARIO file programmatically, use the `Import` remote procedure call (RPC) method. For background information on how the RoadRunner API works, see "Control RoadRunner Programmatically Using gRPC API".

This example primarily shows Windows® commands and file paths to call the RoadRunner API to import an ASAM OpenSCENARIO file, but this API also works on Linux®.

To use the command-line API to control RoadRunner remotely, you must first start the API server on an IP network port. Start the API server by opening a RoadRunner project, which you can do programmatically by using the `AppRoadRunner` executable file.

From the command line, navigate to the folder that contains the `AppRoadRunner` executable file. For example, this code shows the default installation location of the executable file on Windows:

```
cd "C:\Program Files\RoadRunner R2022b\bin\win64"
```

This code shows the default installation location of the executable file on Linux:

```
cd /usr/local/RoadRunner_R2022b/bin/glnxa64
```

Open RoadRunner to an existing project and set the port over which to run the server. Make these updates to the code:

- Replace the `projectPath` option value, `C:\RR\MyProject`, with a path to a valid RoadRunner project on your system. If you do not have an existing project, open RoadRunner and create one interactively. See "RoadRunner Project and Scene System".

- (Optional) Replace the `apiPort` option value, `54321`, with an IP network port number of your choice, between `1024` and `65535`, inclusive. If you omit the `apiPort` option, then RoadRunner opens to its default port of `35707`.

```
AppRoadRunner --projectPath="C:\RR\MyProject" --apiPort=54321
```

RoadRunner opens to a new scene in the specified project.

The **Output** pane displays the port on which the RoadRunner API server is running.

> Started RoadRunner API server on port 54321.

Switch to scenario editing mode. In the top-right corner of RoadRunner, select **Scene Editing**, then **Scenario Editing**. The **Output** pane displays an additional message indicating that the Scenario API server is running on its default port. This server is for cosimulating scenarios with MATLAB and Simulink, or with external simulators such as CARLA. The commands used in this example do not communicate with this server.

> Started RoadRunner API server on port 54321.
> Started Scenario API server on port 35706.

Import an ASAM OpenSCENARIO file of the format `.xosc` into the current scene. To use your own file, update the `file_path` value to the path to your `.xosc` file and update the `serverAddress` value to use the `apiPort` value you specified when opening RoadRunner. If you used the default port, omit the `serverAddress` option.

```
CmdRoadRunnerApi "Import(file_path='C:\OSC\MyOpenSCENARIOFile.xosc' format_name='OpenScenario')" --serverAddress localhos
```

Limitations

- RoadRunner Scenario supports importing only these actions from the `Init` section of an ASAM OpenSCENARIO 1.0 file.

  - `TeleportAction`
  - `SpeedAction`
  - `FollowTrajectoryAction`

  Position type for the `TeleportAction` element must be either `WorldPosition` or `LanePosition`.

  RoadRunner Scenario does not import elements specified in the `Story` section of an ASAM OpenSCENARIO 1.0 file.

- If the `Init` section of your specified file contains any unsupported actions for an actor, then that actor is not imported.
- Importing entities other than vehicles is not supported.

See Also

Related Examples

- Export to ASAM OpenSCENARIO on page 5-2
- "Importing ASAM OpenDRIVE Files"
- "Export to ASAM OpenDRIVE"
- "Control RoadRunner Programmatically Using gRPC API"
- "Generate Scenario Variations Using gRPC API" on page 4-2

External Websites

- ASAM OpenSCENARIO

Import Trajectories from CSV Files

Using RoadRunner Scenario, you can import trajectory data from CSV files both interactively and programmatically.

CSV files store trajectory data in a column format, with each header indicating the type of data within the underlying column.

This table lists the supported header names for the data columns of each component in the CSV file, and the unit of measure for that component. Header names are case sensitive.

Supported Header Names and Units

Component	Units	Supported Names
Position X	meters	x, xpos, posx, positionx, xposition
Position Y	meters	y, ypos, posy, positiony, yposition
Position Z	meters	z, zpos, posz, positionz, zposition
Time	seconds	t, time, timestamp, pointtime, pttime
Yaw	radians	h, heading, yaw
Pitch	radians	p, pitch
Roll	radians	r, roll

Note CSV files must contain X position and Y position components.

Import CSV Files Interactively

To import a trajectory from a CSV file using the RoadRunner Scenario user interface, follow these steps:

1   Select the toggle in the upper-right corner of the application and select **Scenario Editing** to switch to scenario editing mode. Alternatively press **Shift+2**.

2   On the toolbar, select **File**, then **Import**, then **CSV Trajectory**.

3   Specify the path for the CSV file you wish to import by clicking the **...** button and browse to the location of the file. When you select a file, RoadRunner Scenario automatically detects the column names within the file and displays the associated components in the Import CSV Trajectory dialog box.

4   Select **Import**, and once the import is complete, RoadRunner Scenario displays the Import CSV Trajectory Results dialog box. The **Output** pane displays any errors encountered during import. If the import process does not encounter any errors, the **Output** pane is blank and the dialog box displays the "Import CSV Trajectory succeeded" message.

5   Locate the newly imported trajectory. If the trajectory is not visible in the workspace, it might be located outside the view range of the workspace camera. Zoom out until the trajectory is visible, and then select either the trajectory or accompanying vehicle actor and press **F** to frame the selection in the workspace.

6
To simulate the vehicle traveling along the trajectory, select the **Simulation Tool** ![icon] from the toolbar. Then, in the **Simulation** pane, under **Simulation Controls**, select **Play**.

Import CSV Files Programmatically

You can also import CSV files programmatically from the command line by following these steps:

Note Before importing CSV files programmatically, you must already be running RoadRunner Scenario with the workspace open.

---

1   From the command line, navigate to the directory that contains the program files for your RoadRunner installation. For example, this is the default path on Microsoft® Windows 10 machines, where *versionNumber* is the version of your RoadRunner Scenario installation, such as R2022b:

```
C:\ProgramFiles\RoadRunner versionNumber\bin\win64
```

2   Verify that the folder contains the file `CmdRoadRunnerApi.exe`.

3   Run `CmdRoadRunnerApi.exe`, specifying the full path to the CSV file you want to import, as well as the file format name.

```
CmdRoadRunnerApi.exe -c "Import(file_path='Your File Path.csv'
format_name='CSV Trajectory')"
```

- *Your File Path*.csv is the full path of the file.
- The format name for a CSV file is `CSV Trajectory`.

This code shows an example of a full path. The location and name of your file may differ.

```
CmdRoadRunnerApi.exe -c "Import(file_path='C:\User\Documents\RoadRunnerProject\MyCSVs\Tra
```

Once you execute the command, RoadRunner Scenario imports the specified CSV file. To verify the import and test the trajectory, follow steps 5 and 6 of the Import CSV Files Interactively section.

Limitations

CSV importing in RoadRunner Scenario supports only the header names listed in the **Supported Header Names and Units** table. If a CSV file contains headers that CSV importing does not recognize, RoadRunner Scenario ignores the data in the corresponding columns. To prevent RoadRunner Scenario from ignoring column data, modify the header name of each column to reflect one of the supported header names.

If a CSV file is missing either the X position or Y position component, RoadRunner Scenario fails to import the file.

Each row of data within an imported CSV file must contain the same number of commas as in the header row.

You cannot add imported trajectories to existing vehicles within a scenario. Importing a trajectory creates a new vehicle and assigns the trajectory to the newly created vehicle.

See Also

Related Examples
- "Importing ASAM OpenDRIVE Files"
- "Import Custom Vehicle Meshes" on page 2-8
- "Control RoadRunner Programmatically Using gRPC API"
- "Generate Scenario Variations Using gRPC API" on page 4-2

More About
- "Create, Import, and Modify Assets"
- "Import Trajectories from ASAM OpenSCENARIO Files" on page 2-2

Import Custom Vehicle Meshes

This example shows you how to create and import a vehicle mesh that is compatible with RoadRunner Scenario. To create a compatible custom vehicle mesh, follow these workflow steps.

Step	Description
"Set Up Bone Hierarchy" on page 2-8	In a 3D creation environment, set up the vehicle mesh bone hierarchy and specify part names.
"Assign Materials" on page 2-9 (Optional)	Optionally, assign materials to the vehicle parts.
"Export Mesh and Armature" on page 2-10	Export the vehicle mesh and armature in the .fbx file format.
"Import Mesh to RoadRunner Scenario" on page 2-12	Import the vehicle mesh into the RoadRunner Scenario.

Note To create the mesh, this example uses the 3D creation software Blender® Version 2.80.

Set Up Bone Hierarchy

1 Import a vehicle mesh into a 3D modeling tool, such as Blender.
2 To ensure that this mesh is compatible with RoadRunner Scenario, the minimal bone hierarchy in the mesh must include these vehicle parts using this naming convention..

Vehicle Part	Name
Vehicle	*vehicle_name*
Vehicle body	body
Front left wheel	wheel01
Front right wheel	wheel02
Back left wheel	wheel03
Back right wheel	wheel04

3 Set the name of the top-level vehicle object to the expected vehicle type, such as Sedan. The top-level vehicle object must be the parent of the other vehicle objects.
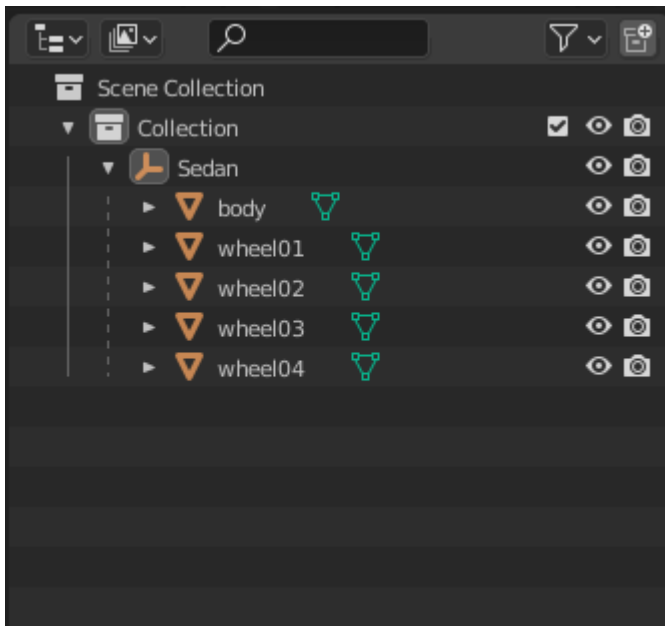
4 For two wheels on the same axle, such as `wheel01` and `wheel02`, their transforms require that any difference in Y-axis offsets be less than 0.01 m. Otherwise, the wheels are categorized into different axles.

Assign Materials

You can optionally assign a material slot to the vehicle `body`. Applying a material containing the word `paint` to a mesh under the `body` node enables you to modify the color of the mesh in the RoadRunner Scenario **Attributes** pane. Because RoadRunner Scenario multiplies the color, you can leave the material the default color of white.
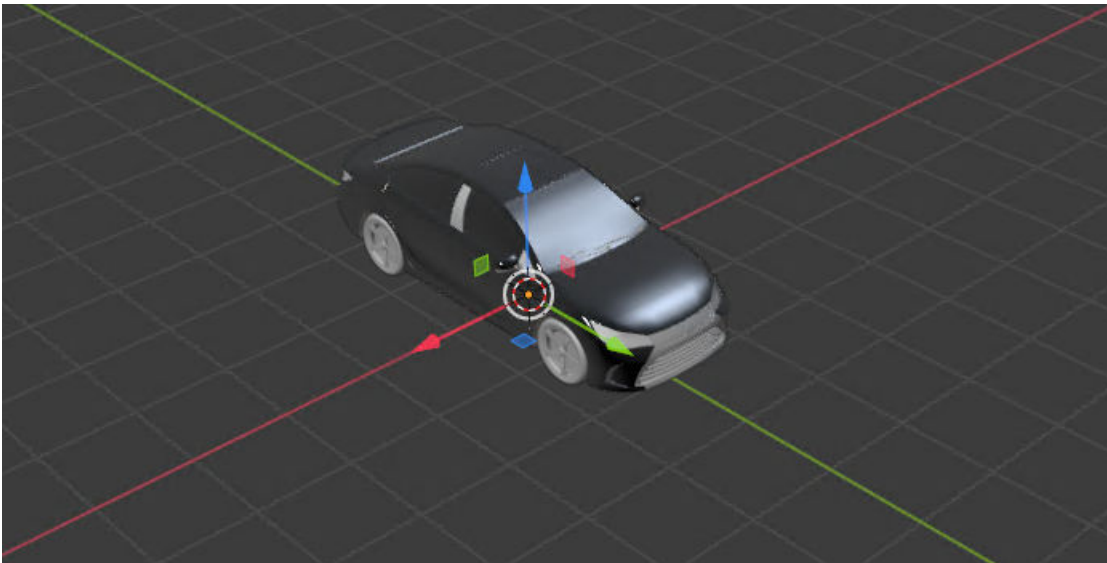
Create and assign material slots to the vehicle body. Confirm that the first material slot corresponds to the `body` object. For example, this image shows the hierarchy in Blender.
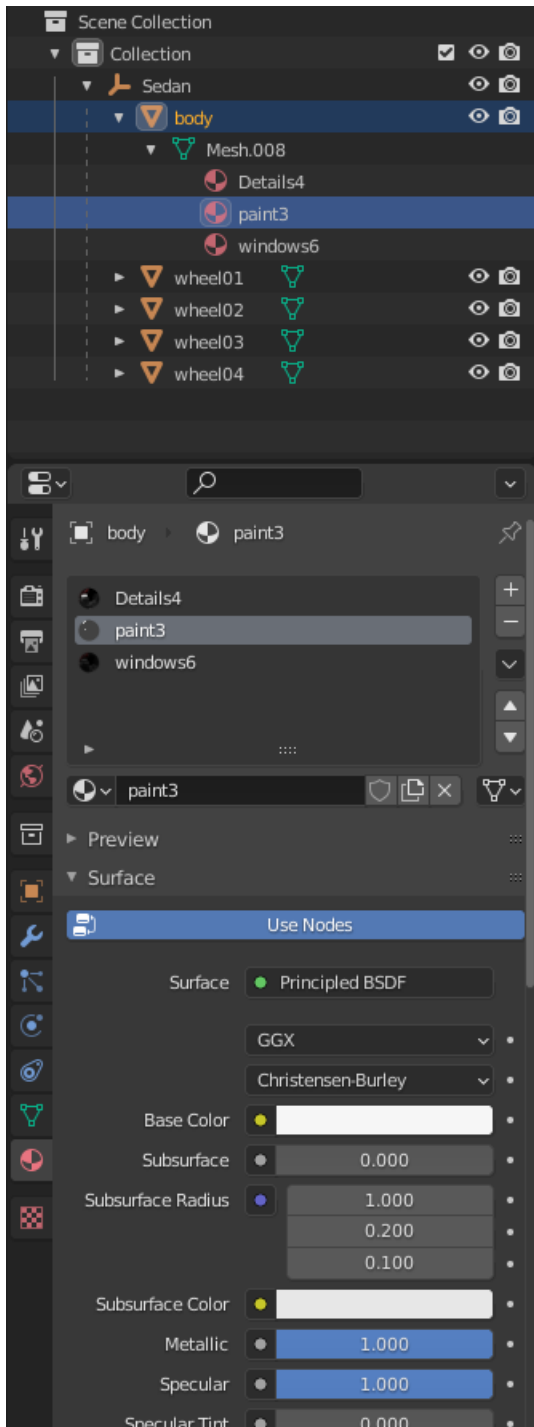
Export Mesh and Armature

Export the mesh and armature to the `.fbx` file format. For example, in Blender:

1   In the **Include** section, for **Object Types**, select **Armature** and **Mesh**.
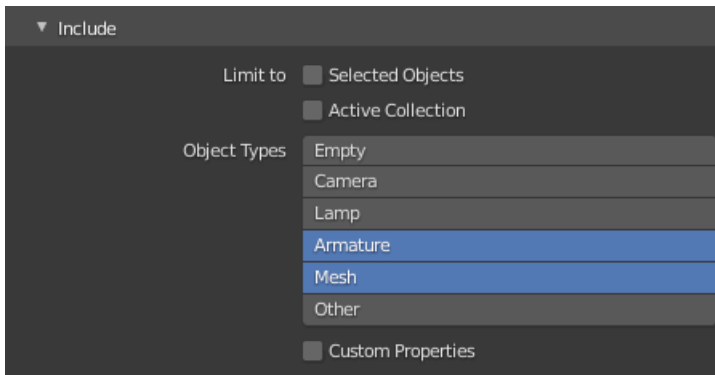
2   On the **Transform** section, set:

- **Scale** to `1.00`
- **Apply Scalings** to `All Local`
- **Forward** to `Y Forward`
- **Up** to `Z Up`

Select **Apply Unit** and **Use Space Transform**.
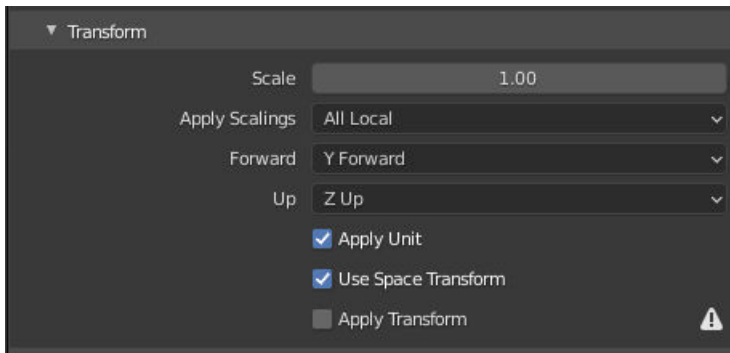
3   On the **Geometry** section:

- Set **Smoothing** to `Face`
- Select **Apply Modifiers**
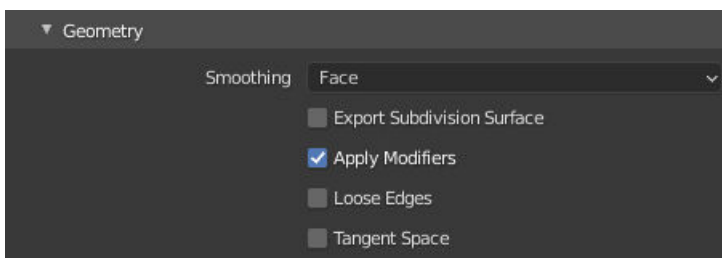
4   On the **Armature** section, set:

- **Primary Bone Axis** to `Y Axis`
- **Secondary Bone Axis** to `X Axis`
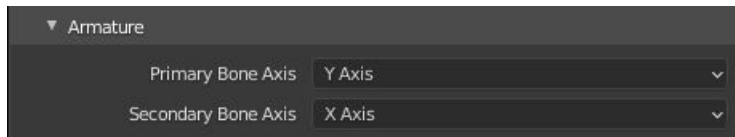
5   Select **Export FBX**.

Import Mesh to RoadRunner Scenario

To import the new FBX model into RoadRunner Scenario, follow the instructions provided in "Create, Import, and Modify Assets". After the new vehicle has been imported, you must change the mesh to a `Vehicle` type. In the asset browser, select the vehicle asset and right-click, select **Default Type > Vehicle**.
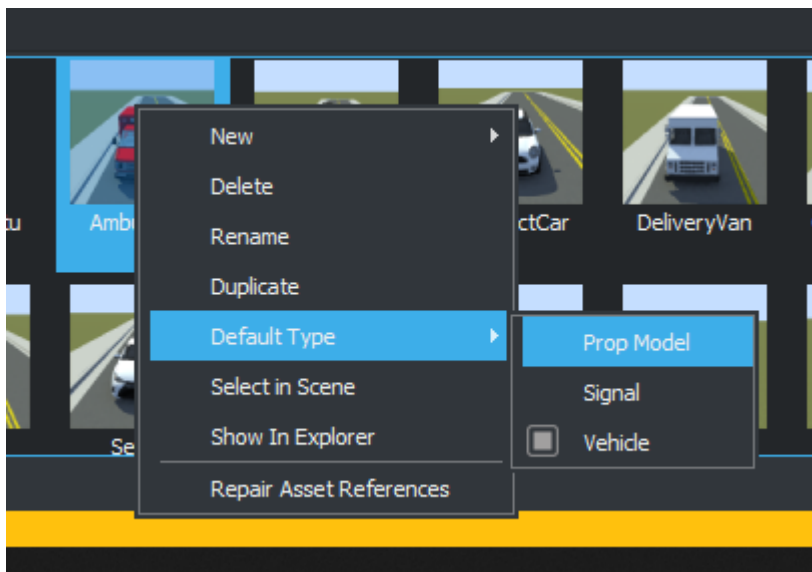
Now you can use the new vehicle asset it in any RoadRunner Scenario simulations.

See Also

"Create, Import, and Modify Assets" | "Import Trajectories from ASAM OpenSCENARIO Files" on page 2-2

Import Custom Character Meshes

This section shows the workflow to create and import a character mesh that is compatible with RoadRunner Scenario. To create a compatible custom character mesh, follow these workflow steps.

Step	Description
"Create Character Mesh" on page 2-13	In a 3D creation environment, set up the character mesh bone hierarchy and specify part names.
"Set Up Bone Hierarchy" on page 2-14	Create a compatible bone hierarchy and rig to the character mesh.
"Create Idle, Walk, and Run Animations" on page 2-16 (Optional)	Create idle, walk, and run animations of your character.
"Import Character into RoadRunner Scenario" on page 2-16	Import the character mesh, rig, and optional animations into RoadRunner Scenario.

Create Character Mesh

Create a character mesh in the 3D creation software. To ensure that the character mesh can be imported into RoadRunner Scenario, the character mesh must:

- Be in a T-pose, by default.
- Include a diffuse, normal, and specular texture map.
- Use proper edge flow with no broken edge seams.

This figure shows a sample of the default character mesh in 3D creation software.

Set Up Bone Hierarchy

Using the character mesh developed in the previous section, create a character rig. To be usable in RoadRunner Scenario, the bones in the character rig must follow the hierarchies in the tables shown. The `Hips_Male` bone must be the root element of the hierarchy.

Supported values of *side* are L for left and R for right. For example, `Hips_Male` has children `Spine_Male`, `R_UpperLeg_Male`, and `L_UpperLeg_Male`.

Supported values of *finger* are `Thumb`, `Index`, `Middle`, `Ring`, and `Pinky`. For example, `R_Hand_Male` has five children: `R_Thumb1_Male`, `R_Index1_Male`, `R_Middle1_Male`, `R_Ring1_Male`, and `R_Pinky1_Male`. Similarly, `R_Thumb1_Male` has child `R_Thumb2_Male`.

Body Hierarchy

Name	Parent	Children
Hips_Male	None	Spine_Male *side*_UpperLeg_Male
Spine_Male	Hips_Male	Spine1_Male
Spine1_Male	Spine_Male	Spine2_Male
Spine2_Male	Spine1_Male	Neck_Male *side*_Shoulder_Male
Neck_Male	Spine2_Male	Neck_Male
Head_Male	Head_Male	None

Leg Hierarchy

Name	Parent	Children
*side*_UpperLeg_Male	Hips_Male	*side*_LowerLeg_Male
*side*_LowerLeg_Male	*side*_UpperLeg_Male	*side*_Foot_Male
*side*_Foot_Male	*side*_LowerLeg_Male	*side*_ToeBase_Male
*side*_ToeBase_Male	*side*_Foot_Male	None

Arm Hierarchy

Name	Parent	Children
*side*_Shoulder_Male	Spine2_Male	*side*_Arm_Male
*side*_Arm_Male	*side*_Shoulder_Male	*side*_Forearm_Male
*side*_Forearm_Male	*side*_Arm_Male	*side*_Hand_Male
*side*_Hand_Male	*side*_Forearm_Male	*side_finger*1_Male
*side_finger*1_Male	*side*_Hand_Male	*side_finger*2_Male
*side_finger*2_Male	*side_finger*1_Male	*side_finger*3_Male
*side_finger*3_Male	*side_finger*2_Male	*side_finger*4_Male
*side_finger*4_Male	*side_finger*3_Male	None

This image shows a sample of the rig for the Male_Citizen character compatible with RoadRunner Scenario.
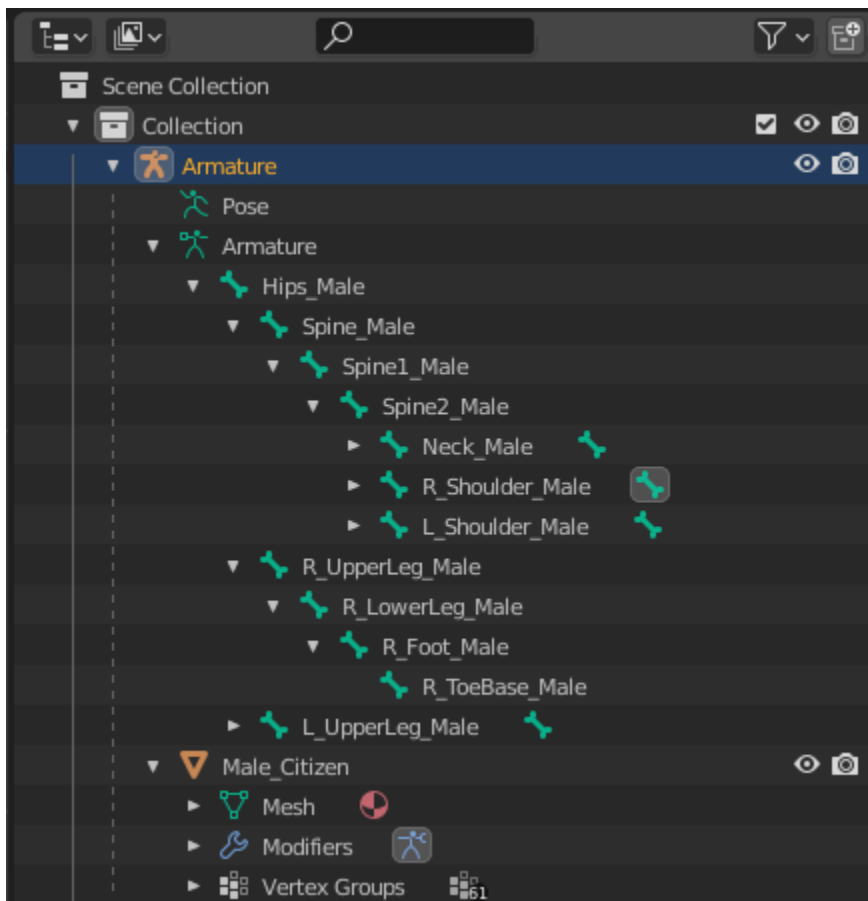
Create Idle, Walk, and Run Animations

Using the character rig, you can generate animations in 3D creation software. A typical strategy for creating an animation from a character rig is:

1   Create controllers for the rig for easier manipulation of the limbs without going through each joint.
2   With the controllers, generate keyframes for the rig.
3   Using the keyframes, create the idle, walk, and run animations.

Note The animations can be made in different tools than the mesh and rig, but the rig hierarchy must be the same to support import into RoadRunner Scenario.

Optionally, if you do not want to create animations, then you can use existing animations of the `Male_Citizen` character asset because your mesh will share the same skeletal rig hierarchy. For more information on rigging and animation in Blender, see https://docs.blender.org/manual/en/latest/animation/introduction.html.

Import Character into RoadRunner Scenario

To import the new character model into RoadRunner Scenario, follow these steps.

1   In RoadRunner Scenario, open the **Asset Browser**, open the **Assets > Characters** folder. Create a new folder with the name of your new character.

2   In 3D creation software, export these objects to your new character folder:

- Character mesh and rig as an FBX file.
- Idle animation as an FBX file.
- Walking animation as an FBX file.
- Running animation as an FBX file.

3   In the **Assets > Characters** folder, create a new character by right-clicking in the **Asset Browser** and selecting **New > Character**. Give the new character the same name as the new character folder created earlier.

4   Assign the character mesh and rig FBX file to the `Skin` and `Skeleton` attributes.

5   Assign the idle, walking, and run FBX files to the `Idle Animation`, `Walk Animation`, and `Run Animation` properties, respectively.

6   Click **Apply Changes** button. Your character display in the **Asset Browser** updates to mesh provided.

Now you can use your new character asset in any RoadRunner Scenario simulation.

See Also
Character Assets

External Websites

- https://www.blender.org/
- https://docs.blender.org/manual/en/latest/animation/introduction.html
- https://www.autodesk.com/products/maya/overview

3

Design and Simulate Scenarios

Design Lane Following Scenario

This example shows how to design a scenario in which a vehicle follows its lane and performs a speed change action. In this example, you learn about the behavior of built-in vehicle actors and how to define actions.

About the Scenario

This scenario contains two vehicles:

- A white sedan drives at a constant speed in one lane throughout the scenario.
- A red sedan drives in the adjacent lane and speeds up after 10 seconds.

Create New Scenario

Create a new scenario in one of the default RoadRunner scenes.

1   Open RoadRunner and, from the start page, select **Open Scene**. If you have RoadRunner open already, then select **Open Scene** from the **File** menu instead.
2   Select the `ScenarioBasic.rrscene` scene from the current project. This scene is included with RoadRunner projects by default.

    This scene contains a four-way intersection with traffic signals, a roundabout, and several roads of varying lengths and curve types.

3   Switch to scenario editing mode. From the top-right corner of the RoadRunner application, select **Scene Editing**, then **Scenario Editing**.

The scene is now locked, and you can begin populating the scenario.

Add Vehicles

Add the two vehicles to the scenario.

1   From the **Library Browser**, drag a `Sedan` asset into the scenario. Place the sedan in the left lane, at the start of the left side of the divided highway and near the road anchor.

The **Logic** editor now contains an initial action phase labeled **Sedan**. This phase initializes the speed of the sedan to approximately 17.9 m/s.
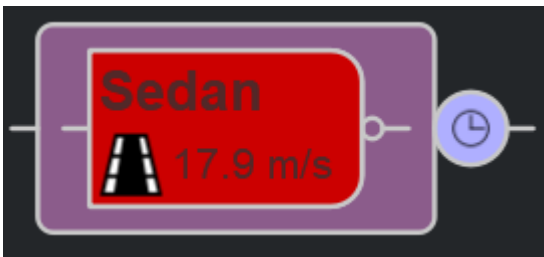
2.   Drag a second `Sedan` asset into the scenario. Place this sedan in the right lane, adjacent to the original sedan.

The **Logic** editor now contains a parallel initial action phase labeled **Sedan2**. This sedan has the same initial speed as the original sedan.
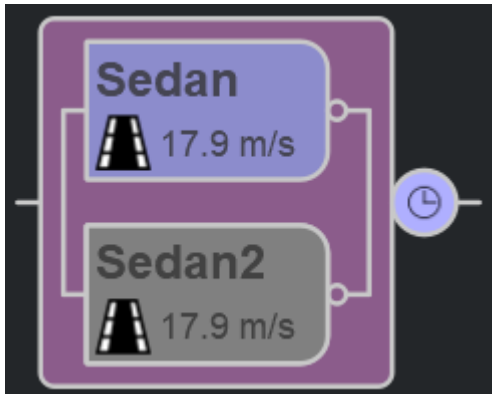
3 Change the color of the new sedan to red to visually differentiate it from the original one. With the new sedan still selected, in the **Attributes** pane, click the **Color** box and select the red color patch. The scenario editing canvas displays the updated color.

4 (Optional) Align the two vehicles by their front bumpers so that they have the same starting point. To set precise alignments, you can align the vehicles with the road anchor that they are attached to.

    a In the scenario editing canvas, select the white sedan.

    b In the **Attributes** pane, select **Manual Reference Line**, and then set **Reference Line** to `Front`.

    c Set **Offset** to `0` meters.

    The front bumper of the white sedan is now aligned with the road anchor.
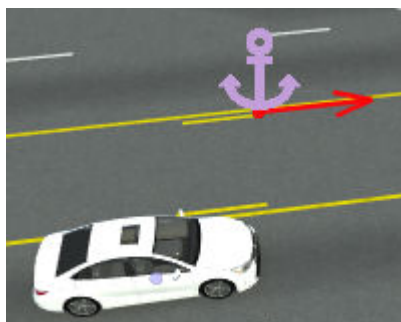
d Repeat these steps with the red sedan so that both sedan front bumpers are aligned with the road anchor.
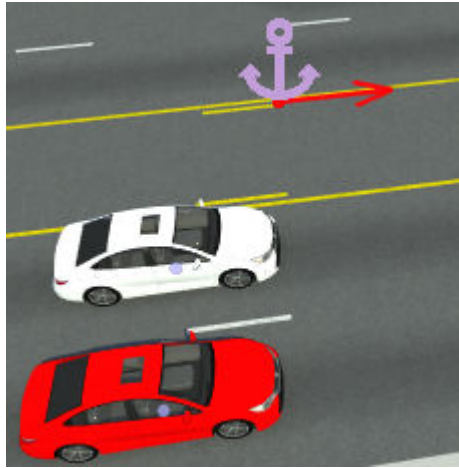
5
Simulate the scenario so far by using the **Simulation Tool** . The two vehicles follow their lanes and drive at the same speed.

6 Stop the simulation early by pressing **Stop**, and then switch back to the **Scenario Edit Tool** . By default, the simulation ends either upon collision or after 60 seconds of simulation time.

Add Speed Change Action

Add an action phase in which the red sedan speeds up to 30 m/s.

1 In the **Logic** editor, right-click the **Sedan2** initial phase and select **Add Action Phase After**.

The **Logic** editor adds a new a new `Change Speed` action phase. By default, this action phase is set to the same speed as the initial action phase: 17.9 m/s.
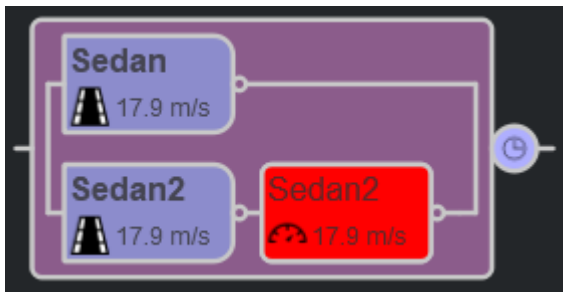
2 With the new action still selected, from the **Change Speed** section of the **Attributes** pane, set **Speed** to 30 m/s.

3
Simulate the scenario so far by using the **Simulation Tool** . The red sedan now immediately changes speeds and drives faster than the white sedan.

Add Speed Change Condition

Currently, the red sedan immediately changes speed after initialization. Set a condition that causes the red sedan to change speeds only after 10 seconds have elapsed.

1 Select the condition node that is connected to the initial phase of the red sedan, **Sedan2**.

2 From the **Attributes** pane, select the `Duration` condition. The **Logic** editor displays the selected condition.
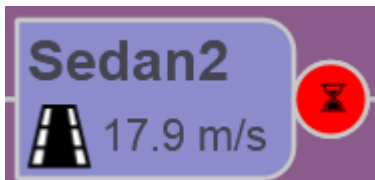
3 In the **Attributes** pane, set **Duration** to 10 seconds.

4 Simulate the scenario. The sedans now drive at the same speed initially. Then, after 10 seconds, the red sedan speeds up.

The **Logic** editor displays the active actions and conditions during simulation. Green actions and conditions have been completed, orange ones are currently active, and gray ones (not shown in this figure) are not active.

Alternatively, instead of using a `Duration` condition, you can use the `Simulation Time` condition.

- The `Duration` condition becomes active as soon as the associated action phase starts.
- The `Simulation Time` condition becomes active as soon as the associated action phase starts, and is satisfied once the specified simulation time is met.

In this scenario, since the associated action phase (the `Initial Speed` action) starts at the beginning of simulation, the `Duration` and `Simulation Time` conditions are effectively the same.

Other Things to Try

To customize the scenario further, try modifying the speeds of the vehicles and observe the effects on the simulation. Also, try specifying other action phases for the red sedan. For example, try changing the `Speed Change` action phase to a `Lane Change` action phase, and observe how the simulation changes. For an example of a lane change scenario, see "Design Lane Change Scenario" on page 3-10.

Once you are satisfied with the scenario, you can export it to ASAM OpenSCENARIO. For more details on exporting to ASAM OpenSCENARIO, see "Export Scenarios".

You can also try generating variations of the scenario. For example, create a variable for the speed change. Select the speed change action phase from the **Logic** editor and, in the **Attributes** pane, right-click the **Speed** attribute and select **Create Variable**.

The **Variables** table now displays a `ChangeSpeed_TargetSpeed` variable.

You can then programmatically change this variable and export the varied scenarios to ASAM OpenSCENARIO. For more details on generating scenarios, see "Generate Scenario Variations Using gRPC API" on page 4-2.

See Also

Related Examples

More About

Design Lane Change Scenario

This example shows how to design a scenario in which one vehicle changes into the same lane as another vehicle. In this example, you learn how to set conditions and define parallel actions, where a vehicle changes lanes and speeds simultaneously.

About the Scenario

This scenario contains two vehicles:

- The ego vehicle, a white sedan, drives at a constant speed in one lane throughout the scenario.
- The lead vehicle, a red sedan, cuts into the lane of the ego vehicle when the ego vehicle is within 30 meters of it.

By the time the lead vehicle completes its lane change, it is driving faster than the ego vehicle, so no collision occurs.

This scenario is based on the ASAM OpenSCENARIO automated lane keeping systems (ALKS) test scenario 4.4_1. For more details on ALKS scenarios, see the ALKS Scenario page on GitHub®.

Create New Scenario

Create a new scenario in one of the default RoadRunner scenes.

1 Open RoadRunner, and from the start page, select **Open Scene**. If you have RoadRunner open already, then select **Open Scene** from the **File** menu instead.

2 Select the `ScenarioBasic.rrscene` scene from the current project. This scene is included with RoadRunner projects by default.

   This scene contains a four-way intersection with traffic signals, a roundabout, and several roads of varying lengths and curve types.

3 Switch to scenario editing mode. From the top-right corner of the RoadRunner application, select **Scene Editing**, then **Scenario Editing**.

The scene is now locked, and you can begin populating the scenario.

Add Ego Vehicle

Add the ego vehicle to the scenario. This vehicle drives at a constant speed and stays in the same lane throughout the scenario.

1 From the **Library Browser**, drag a Sedan asset into the scenario. Place the sedan in the left lane, at the start of the left side of the divided highway.

2 Rename the vehicle. With the sedan still selected, in the **Attributes** pane, set **Name** to Ego.

The **Logic** editor now contains an initial action phase labeled **Ego**. This phase initializes the speed of the ego vehicle to 17.9 m/s. Because the sedan has no driving path defined, the sedan drives in its current lane at the set speed by default.
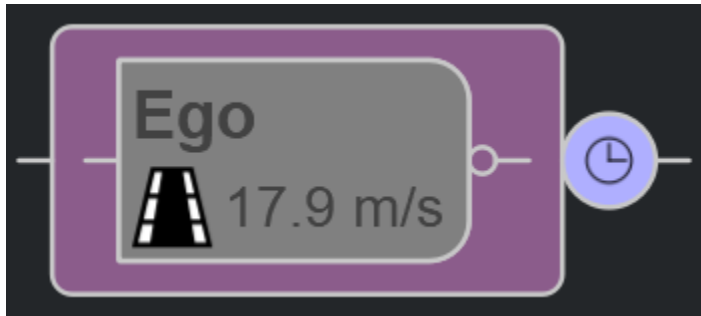
The scenario is set to end after 60 seconds of simulation time or upon the first collision. Leave the default end condition.

Add Lead Vehicle

Add the lead vehicle to the scenario.

1 From the **Library Browser**, drag another Sedan asset into the scenario. Place the new sedan in the lane adjacent to the original sedan and several car lengths in front of it.

2 Rename the lead vehicle. With the vehicle still selected, in the **Attributes** pane, set **Name** to Lead Vehicle.

3 Change the color of the lead vehicle to visually differentiate it from the ego vehicle. Click the **Color** attribute box and select the red color patch. In the scenario editing canvas, the lead vehicle is now red.
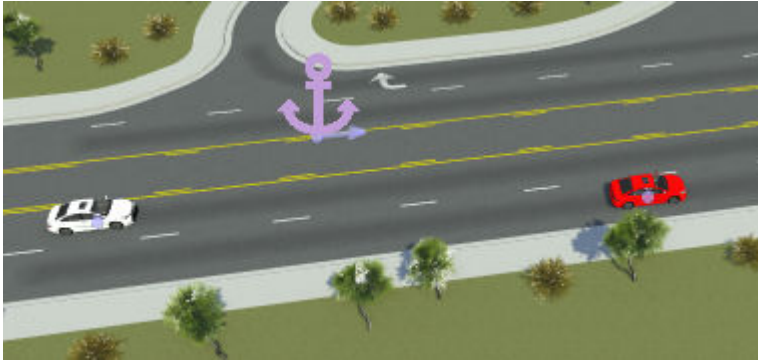
4 Set the initial speed of the lead vehicle to be slower than the ego vehicle. In the **Logic** editor, select the initial action phase for the lead vehicle. Then, in the **Attributes** pane, in the `Initialize Speed` action, set **Speed** to 10 m/s.
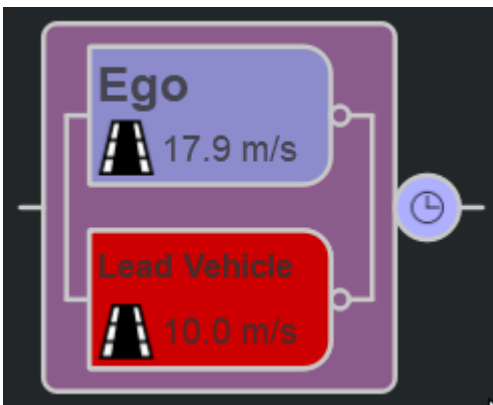
5

Simulate the scenario. From the RoadRunner Scenario toolbar, select the **Simulation Tool** .
Then, in the **Simulation** pane, click **Play**.

The ego vehicle and lead vehicle follow their lanes, and the ego vehicle eventually overtakes the lead vehicle.
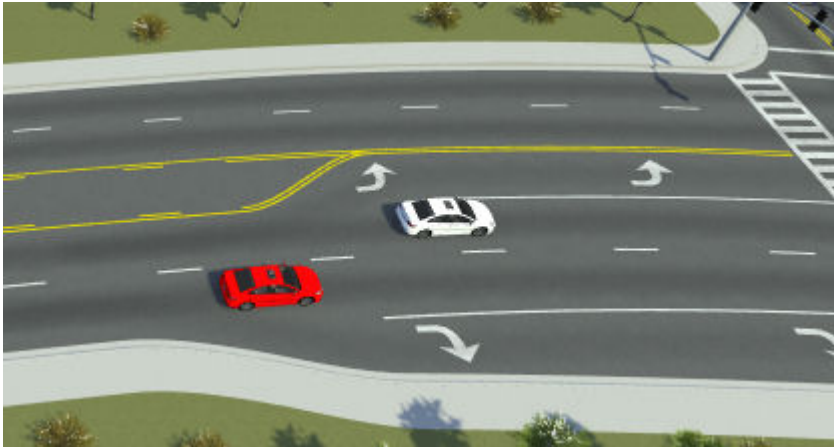
6 Stop the simulation early by pressing **Stop**, and then switch back to the **Scenario Edit Tool**
.

Add Lane Change Action

Specify a lane change action in which the lead vehicle cuts into the lane of the ego vehicle over a distance of 20 meters.

1 Right-click the **Lead Vehicle** initial phase and select **Add Action Phase After**. By default, RoadRunner Scenario sets action phases added after `Initial Speed` actions to `Change Speed` actions.

2 In the **Attributes** pane, set **Action Type** to `Change Lane`. The **Logic** editor displays the updates to the action phase.
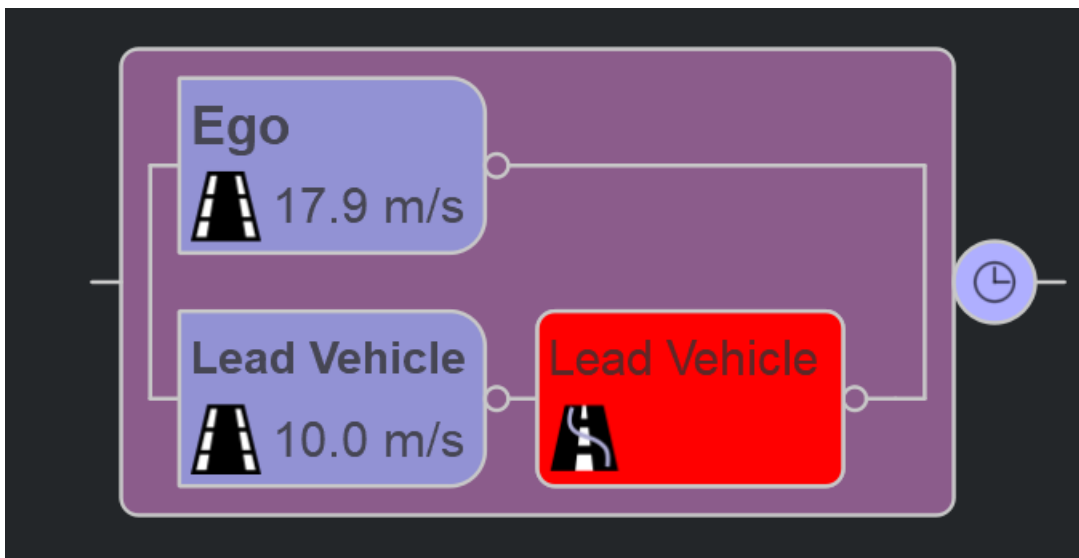
3 In the **Change Lane** section of the **Attributes** pane, specify these lane change attributes, in this order:

   a Set **Relative to** to `Actor`.

b Click the **Reference Actor** box and select the ego vehicle from either the scenario editing canvas or the **Logic** editor. These areas of the RoadRunner layout are outlined by blue lines.

c Set **Direction** to Same Lane.

d Set **Distance** to 20 meters.

4

Simulate the scenario so far by using the **Simulation Tool** . The lead vehicle now immediately changes lanes, but the ego vehicle then collides with the slower lead vehicle.

Add Parallel Speed Change Action

To prevent a collision, specify a speed change action that occurs in parallel with the lane change action. In this example, the lead vehicle speeds up to 20 m/s.

1   In the **Logic** editor, right-click the lane change action phase and select **Add Action Phase Below**. The **Logic** editor adds an unset parallel action phase below the lane change action phase.
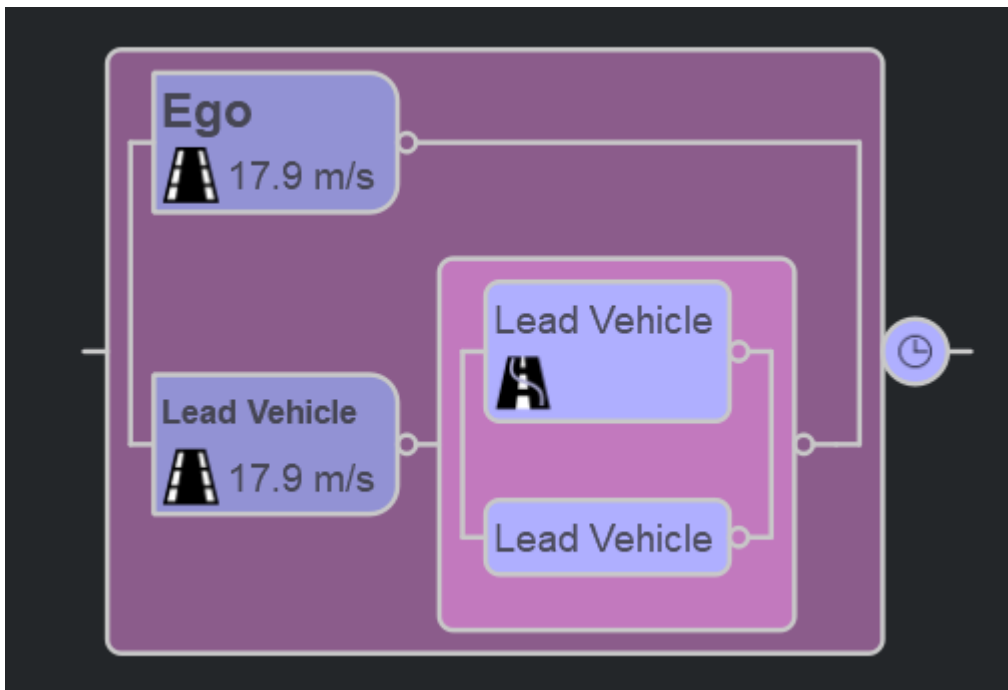
2   In the **Attributes** pane, set **Action Type** to Change Speed.

3   In the **Change Speed** section, set **Speed** to 20 m/s. The **Logic** editor displays the speed change update in the action phase.
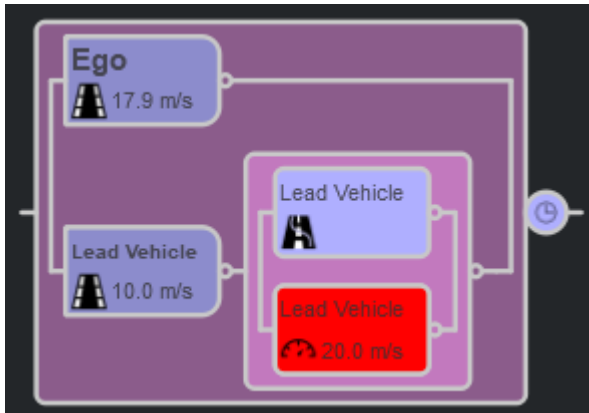
4 Simulate the scenario so far. Because the lead vehicle speeds up while performing the lane change, the scenario no longer ends in collision.
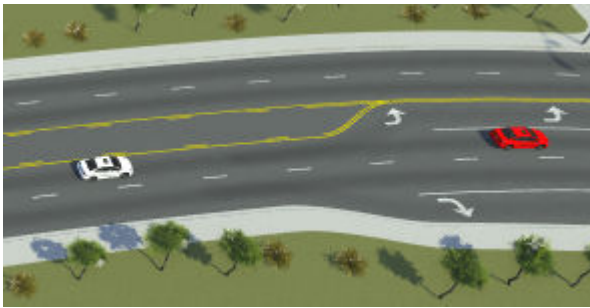
Set Lane Change Condition

Instead of having the lead vehicle change lanes as soon as the simulation starts, set a condition that triggers the lane change. In this example, the lead vehicle begins its maneuver when it is less than 20 meters from the ego vehicle.

1 Select the condition node connected to the initial phase of the lead vehicle.
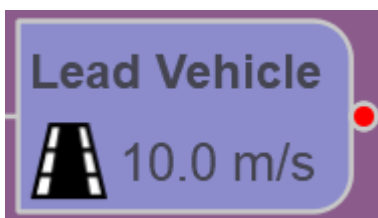
2 From the **Attributes** pane, select the `Distance to Actor` condition. The **Logic** editor displays the selected condition.
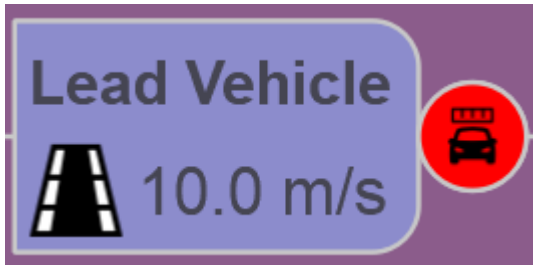
3 In the **Attributes** pane, click the **Reference Actor** attribute box. Then, select the ego vehicle either from the scenario editing canvas or the **Logic** editor.

4 Set **Threshold** to 20 meters. The scenario editing canvas displays a dashed line between the ego vehicle and lead vehicle. The blue arrow within this line is 20 meters from the lead vehicle. When the ego vehicle reaches this arrow, the lead vehicle meets its **Distance to Actor** condition.
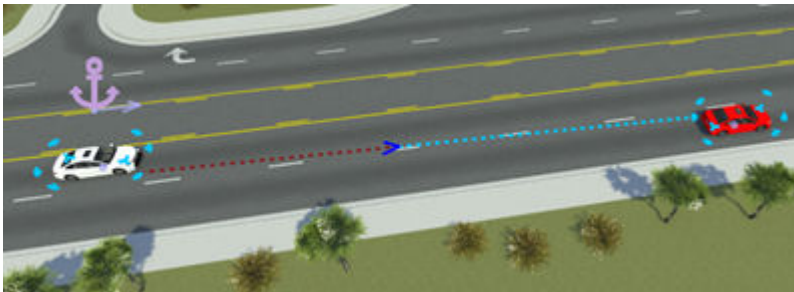
5 Simulate the scenario. The lead vehicle now does not start its lane change until the ego vehicle is closer. Depending on your initial placement of the vehicles, this scenario can end in collision.

Other Things to Try

To customize the scenario further, try modifying the location and speeds of the vehicles and observe the effects on the simulation.

Once you are satisfied with the scenario, you can export it to ASAM OpenSCENARIO. For more details on exporting to ASAM OpenSCENARIO, see "Export Scenarios".

You can also try generating variations of the scenario. For example, create a variable for the distance at which the lead vehicle begins its cut-in. Select the condition node for the lead vehicle and, in the **Attributes** pane, right-click the **Threshold** attribute and select **Create Variable**.
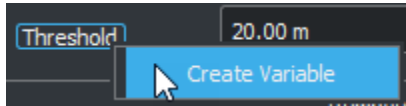
The **Variables** table now displays a `DistanceToActor_Threshold` variable.

You can then programmatically change this variable and export the generated variations of scenarios to ASAM OpenSCENARIO. For more details on varying scenarios, see "Generate Scenario Variations Using gRPC API" on page 4-2.

See Also

Related Examples

- "Design Lane Following Scenario" on page 3-2
- "Design Lane Swerve Scenario" on page 3-20
- "Design Path Following Scenario" on page 3-28

More About

- "Define Scenario Logic" on page 3-70

Design Lane Swerve Scenario

This example shows how to design a scenario in which one vehicle swerves from side-to-side within its lane. In this example, you learn how to use lateral offset actions, and how to define scenarios in which vehicles do not follow a predefined path.

About the Scenario

This scenario contains two vehicles:

- The ego vehicle, a white sedan, drives at a constant speed in one lane throughout the scenario.
- The lead vehicle, a red sedan, drives at a constant speed in front of the ego vehicle, but swerves to the left and right within its lane during simulation.
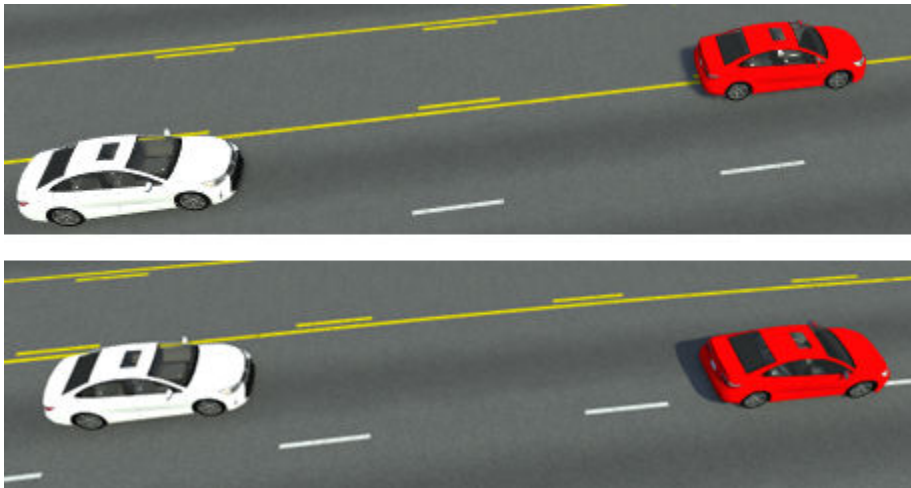
This scenario is based on the ASAM OpenSCENARIO automated lane keeping systems (ALKS) test scenario 4.1_2. For more details on ALKS scenarios, see the ALKS Scenario page on GitHub.

Create New Scenario

Create a new scenario in one of the default RoadRunner scenes.

1   Open RoadRunner and, from the start page, select **Open Scene**. If you have RoadRunner open already, then select **Open Scene** from the **File** menu instead.

2   Select the `ScenarioBasic.rrscene` scene from the current project. This scene is included with RoadRunner projects by default.

    This scene contains a four-way intersection with traffic signals, a roundabout, and several roads of varying lengths and curve types.

3 Switch to scenario editing mode. From the top-right corner of the RoadRunner application, select **Scene Editing**, then **Scenario Editing**.

The scene is now locked, and you can begin creating the scenario.

Add Ego Vehicle

Add the ego vehicle to the scenario. This vehicle drives at a constant speed and stays in the same lane throughout the scenario.

1 From the **Library Browser**, drag a Sedan asset into the scenario. Place the sedan in the left lane, at the start of the left side of the divided highway.

2 Rename the vehicle. With the sedan still selected, in the **Attributes** pane, set **Name** to `Ego`.

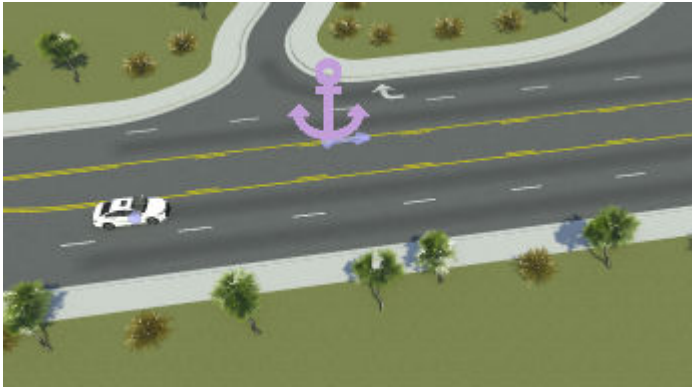The **Logic** editor now contains an initial action phase labeled **Ego**. This phase initializes the speed of the ego vehicle to approximately 17.9 m/s. Because the sedan has no driving path defined by default, the sedan drives in its current lane at the set speed.

The scenario, by default, ends after 60 seconds of simulation time or upon the first collision. Leave the default end condition.

Add Lead Vehicle

Add the lead vehicle to the scenario.

1 From the **Library Browser**, drag a second `Sedan` asset into the scenario. Place the new sedan in the same lane as the original sedan and slightly in front of it.
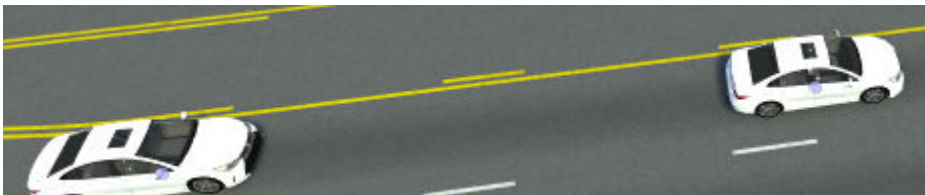
2 Rename the lead vehicle. With the vehicle still selected, in the **Attributes** pane, set **Name** to `Lead Vehicle`.

3 Change the color of the lead vehicle to visually differentiate it from the ego vehicle. Click the **Color** attribute box and select the red color patch. In the scenario editing canvas, the lead vehicle is now red.
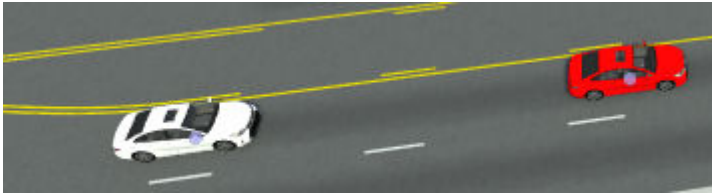
4   Simulate the scenario so far. From the RoadRunner Scenario toolbar, select the **Simulation Tool** . Then, in the **Simulation** pane, click **Play**.

The ego vehicle and lead vehicle follow their lane and drive at their default speed of 17.9 m/s.
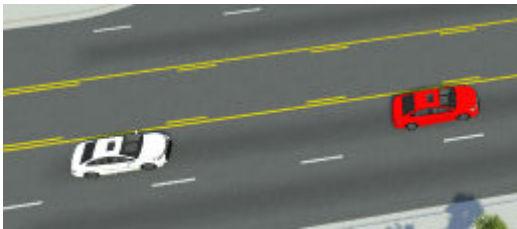
5   Stop the simulation early by pressing **Stop**, and then switch back to the **Scenario Edit Tool** . By default, the simulation ends either upon collision or after 60 seconds of simulation time.

Add Lane Swerve Actions

Specify the actions that cause the vehicle to swerve left, straighten back to center, swerve right, and straighten back to center again.

To cause the lead vehicle to swerve left and right, specify the lateral distance, in meters, to offset the vehicle from its lane center. In this example, specify offsets of 1.5 meters to the left and right of the vehicle. At these distances, the vehicle center roughly aligns with the left and right edges of the lane. Lateral coordinates are positive to the right of the vehicle. Therefore, to swerve left, specify -1.5 meters. To swerve right, specify 1.5 meters of lateral offset.

1   Right-click the **Lead Vehicle** initial phase and select **Add Action Phase After**. By default, RoadRunner Scenario sets action phases added after `Initial Speed` actions to `Change Speed` actions.

2 Specify the action in which the lead vehicle swerves left. In the **Attributes** pane, set **Action Type** to `Change Lateral Offset`. The **Logic** editor displays the updates to the action phase.
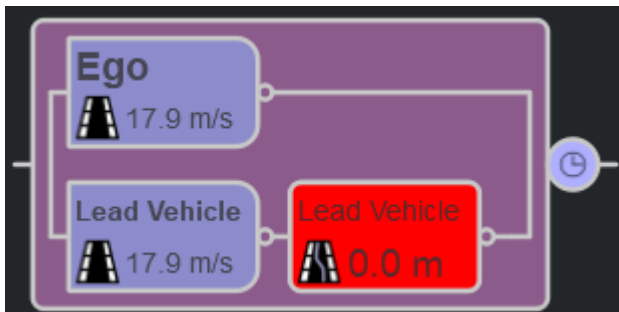
3 In the **Attributes** pane, set the **Lateral Offset** attribute to `-1.5` m. The Logic editor displays the updated lateral offset value.
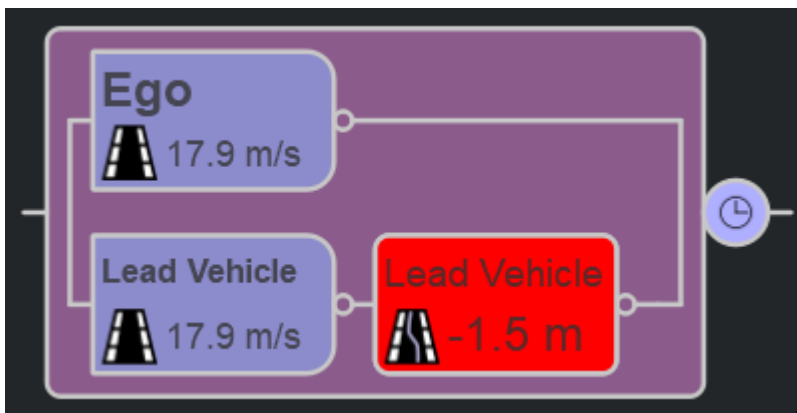
Set the **Time** attribute to 2 so that the swerve to the left occurs over the course of 2 seconds.

4 Add three additional lateral offset actions after the first lateral offset action by right-clicking the action and selecting **Add Action Phase After**. The added action phases inherit their values from the first action phase.

5 Select each lateral offset action phase and set the **Lateral Offset** attributes to the values shown in the table.

Action Phase	Lateral Offset Attribute Value	Vehicle Action
First lateral offset action	`-1.5` (already set)	Swerve left
Second lateral offset action	`0`	Straighten back to center
Third lateral offset action	`1.5`	Swerve right

Action Phase	Lateral Offset Attribute Value	Vehicle Action
Fourth lateral offset action	0	Straighten back to center

The **Logic** editor displays the updated values.
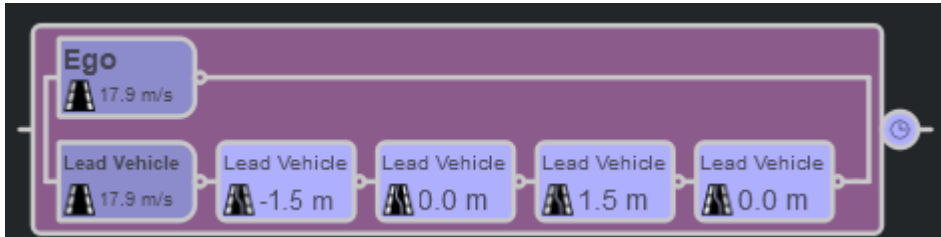
Each action occurs over the course of 2 seconds, because the added actions inherit the **Time** value of 2 from the first lateral offset action.

6

Simulate the scenario so far by using the **Simulation Tool** . The lead vehicle swerves to the left, but because the logic has no conditions to trigger the subsequent actions, the vehicle does not perform the subsequent lateral offset actions
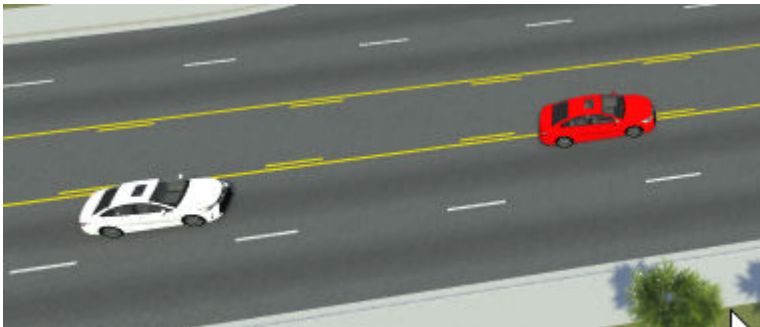
Add Lane Swerve Conditions

Set the conditions that trigger the subsequent lateral offset actions that the lead vehicle performs. In this example, have the lead vehicle proceed to each action after 5 seconds.

1	Click the condition node that is connected to the first lateral offset action (swerve left action) to select it.
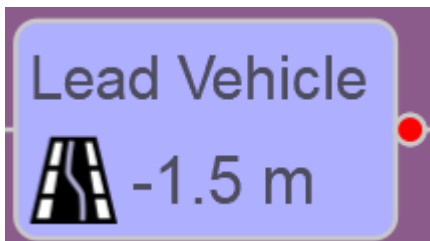
2	From the **Attributes** pane, select the `Duration` condition. The **Logic** editor displays the selected condition.
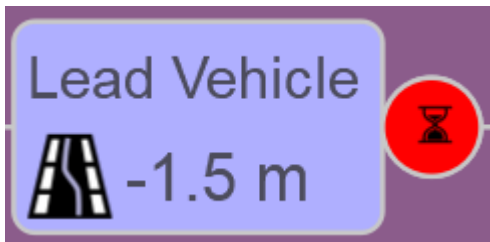
Leave the **Duration** attribute value set to the default of 5 seconds.

3 Add identical **Duration** conditions of 5 seconds to each of the subsequent lateral offset actions. The Logic editor displays the added conditions.
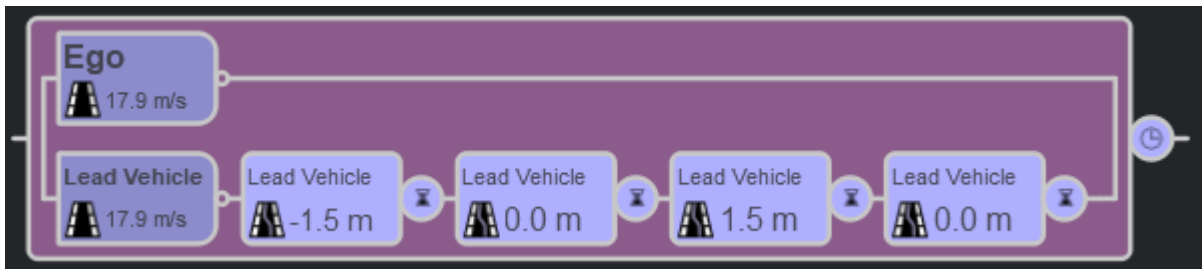
4 Simulate the scenario. The lead vehicle now switches between actions in five-second intervals.
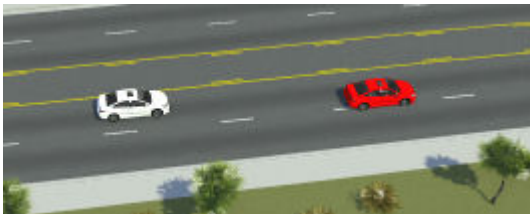
When the lead vehicle reaches the final lateral offset action (straighten back to center), it maintains this action, even after the duration condition is met.
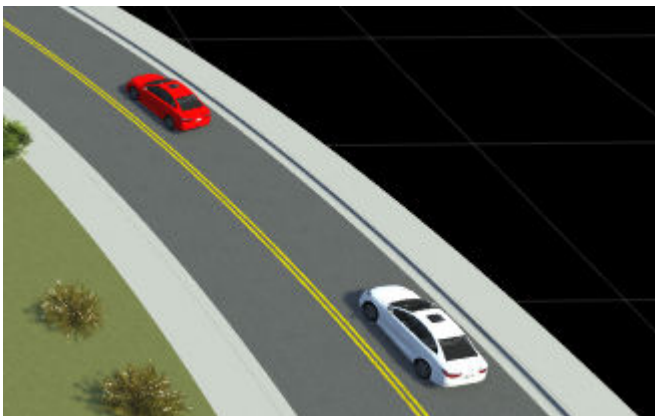
Other Things to Try

To customize the scenario further, try modifying various attributes of the scenario logic, such as:

- The **Lateral Offset** values.
- The **Time** values over which the lateral offset actions occur.
- The **Duration** values between each lateral offset action.

Once you are satisfied with the scenario, you can export it to ASAM OpenSCENARIO and visualize the scenario in a simulation viewer such as esmini. For more details on exporting to ASAM OpenSCENARIO, see "Export Scenarios".

You can also try generating variations of the scenario. For example, create a variable for the first lateral offset value. First, in the **Logic Editor**, select the first lateral offset action for the lead vehicle. Then, in the **Attributes** pane, right-click the **Lateral Offset** attribute and select **Create Variable**.
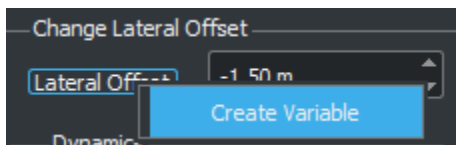
The **Variables** table now displays a `ChangeLateralOffset_LateralOffsetValue` variable.

You can then programmatically change this variable and export the varied scenarios to ASAM OpenSCENARIO. For more details on generating various scenarios, see "Generate Scenario Variations Using gRPC API" on page 4-2.

See Also

Related Examples
- "Design Lane Following Scenario" on page 3-2
- "Design Lane Change Scenario" on page 3-10
- "Design Path Following Scenario" on page 3-28

More About
- "Define Scenario Logic" on page 3-70

Design Path Following Scenario

This example shows how to design a scenario in which a vehicle follows a predefined driving path that goes on- and off-road. By specifying explicit driving paths, you can create more complex and unique scenarios.

About the Scenario

This scenario contains one vehicle, a white sedan. In the scenario, the vehicle starts on the road, slows down as it drives off the road, and then speeds up as it drives back onto the road.

Create New Scenario

Create a new scenario in one of the default RoadRunner scenes.

1   Open RoadRunner and, from the start page, select **Open Scene**. If you have RoadRunner open already, then select **Open Scene** from the **File** menu instead.

2   Select the `ScenarioBasic.rrscene` scene from the current project. This scene is included with RoadRunner projects by default.

    This scene contains a four-way intersection with traffic signals, a roundabout, and several roads of varying lengths and curve types.

3 Switch to scenario editing mode. From the top-right corner of the RoadRunner application, select **Scene Editing**, then **Scenario Editing**.

The scene is now locked, and you can begin populating the scenario.

Add Vehicle

Add the vehicle to the scenario.

1 From the **Library Browser**, drag a Sedan asset into the scenario. Place the sedan in the left lane, at the start of the right side of the divided highway.

The **Logic** editor now contains an initial action phase labeled **Sedan**. This phase initializes the speed of the sedan to approximately 17.9 m/s.
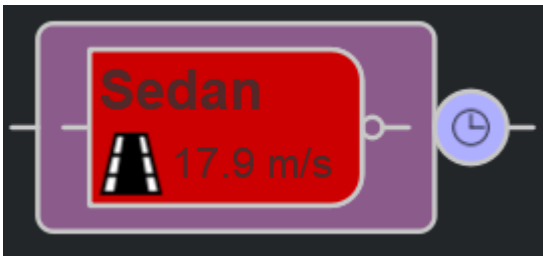
2   Simulate the scenario so far by using the **Simulation Tool** . With no path specified, the sedan drives in its lane at its initial speed.

3   Stop the simulation early by pressing **Stop**, and then switch back to the **Scenario Edit Tool**
. By default, the simulation ends either upon collision or after 60 seconds of simulation time.

Add On-Road Path Segment

Add an explicit driving path for the vehicle to follow. For this first path segment, the vehicle changes lanes and drives straight in its new lane. In future path segments, the vehicle drives off the road.

1 In the scenario editing canvas, click the sedan to select it.
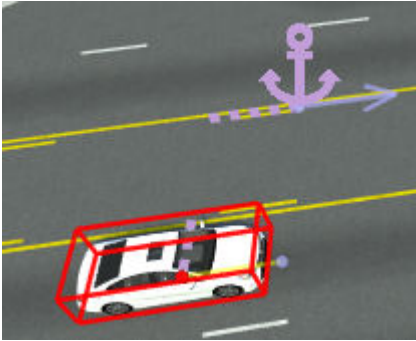
2 Add a path segment by right-clicking the adjacent lane slightly in front of the ego vehicle. RoadRunner adds a new path waypoint, selected in yellow, at the point where you clicked. The purple path segment snaps to the center of the lane and includes a lane change maneuver.
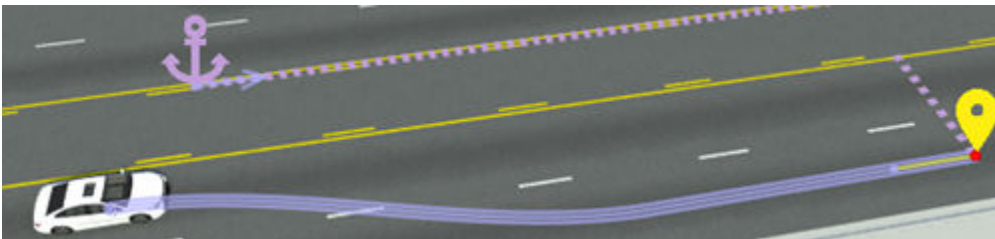
The **Logic** editor now shows that the vehicle drives along a predefined path instead of along its lane.
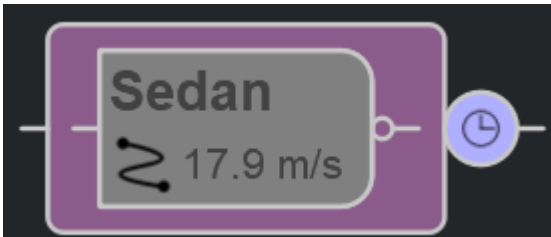
3 Drag the selected waypoint until just before the turning lanes form. This is the point at which you want the vehicle to go off the road.

4 Simulate the scenario by using the **Simulation Tool** button. The sedan changes lanes and drives until it reaches the specified waypoint. The simulation continues until 60 seconds of simulation time elapse, or until you stop the scenario early.
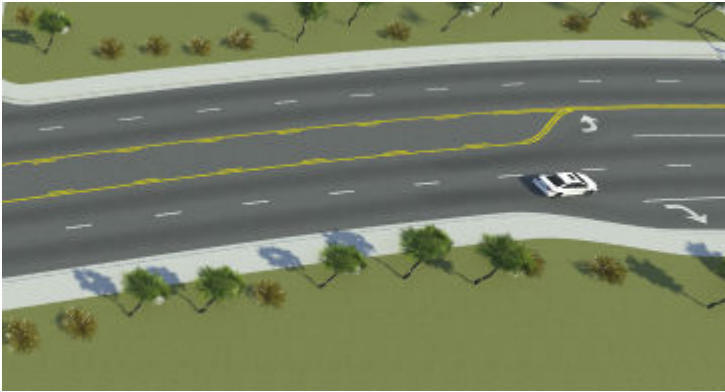
Add Off-Road Path Segment

Add free-form, off-road path segments to the end of the driving path. In this scenario, the vehicles cuts through the ground terrain to avoid the intersection.

1   In the scenario editing canvas, click the driving path to select it.

2   Add a new waypoint to the path by right-clicking the southbound lane, below the intersection. By default, this path continues to follow the road network. The vehicle now merges into the turning lane, turns right, and then follows the road south.
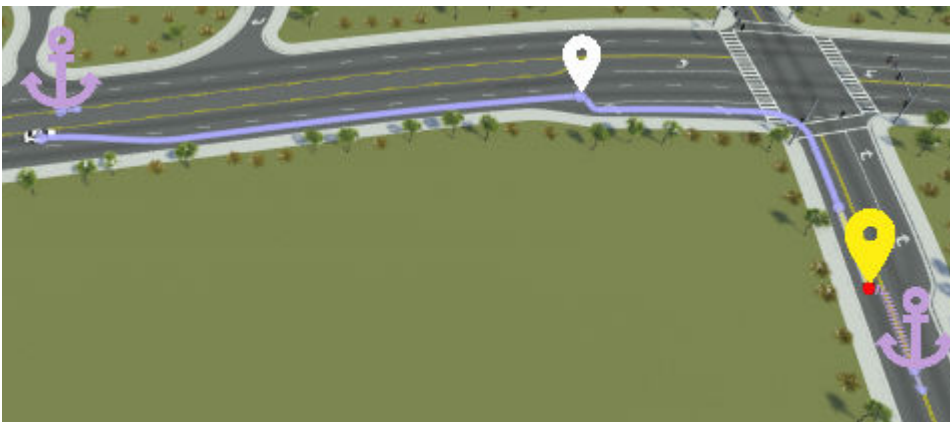
3   Convert the added path segment to an off-road path. Select the path segment and, in the **Attributes** pane, select **Route Segment Parameters > Freeform**.

The path segment no longer follows the road network and does not turn at the intersection. Instead, the path follows a curve based on the specified **Curve Type**.
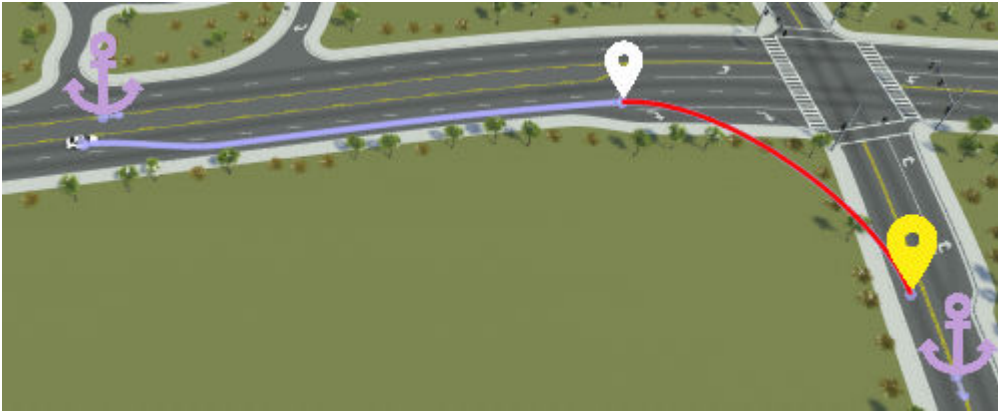
4 Simulate the scenario to view the sedan traveling off the road along the curved path.

Refine Off-Road Path Segment

Modify the off-road path segment to introduce more complex maneuvers.

1 Split the off-road path segment into two segments. First, click the path, and then click the off-road path segment to select it. Then, right-click the center of the path segment to add a waypoint and split the segment. The split segments inherit the attributes of the original segment and ignore the road network.
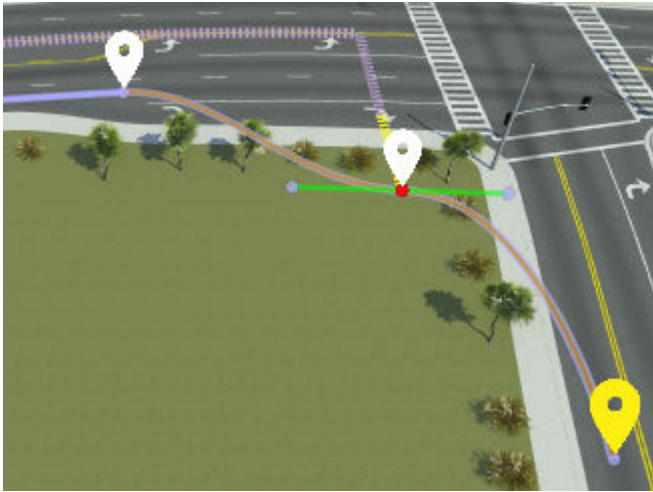
2 Click and drag the off-road path waypoint to form a more complex off-road path.

3 Optionally refine the path further by modifying the tangent of the waypoint. For example, try rotating or extending the tangent lines.
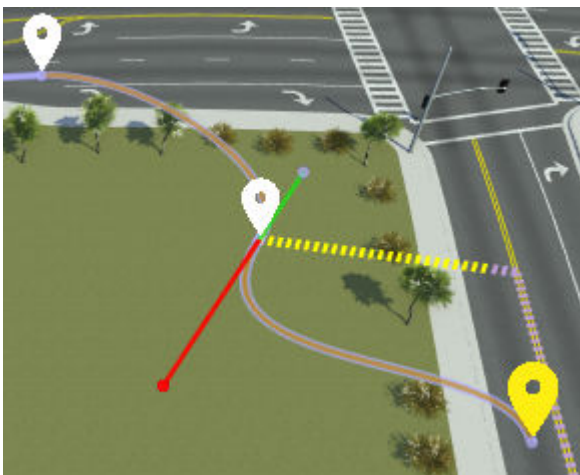
4 Simulate the scenario to view the sedan traveling the more complex off-road path.

Add Speed Change Along Path

Currently, the vehicle drives at a constant speed along the entire path. To make the scenario more realistic, slow the vehicle down while it drives off the road.

Add Speed Change Action

Add a speed change action and decrease the speed of the sedan to 10 m/s.

1 In the **Logic** editor, right-click the **Sedan** initial phase and select **Add Action Phase After**.

The **Logic** editor adds a new speed change action phase. By default, this action phase is set to the same speed as the initial action phase: 17.9 m/s.
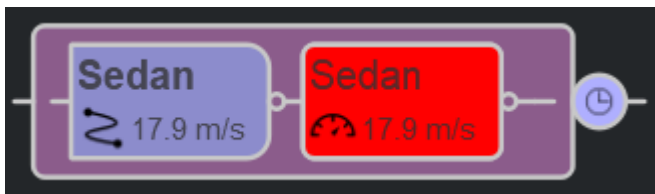
2 With the new action still selected, from the **Change Speed** section of the **Attributes** pane, set **Speed** to 10 m/s. The **Logic** editor displays this change in speed.
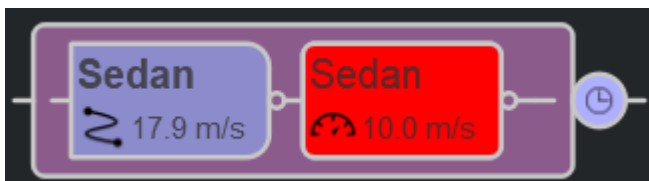
Set Speed Change Condition

Set the condition that triggers the decrease in speed. In this example, the speed change occurs when the vehicle begins the off-road segment of its path.

1 Select the condition node that is connected to the initial action phase of the sedan.
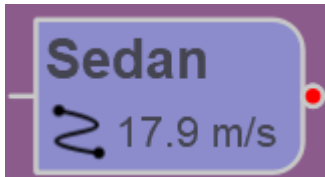
2   From the **Attributes** pane, select the `Distance to Point` condition. The **Logic** editor displays the selected condition.
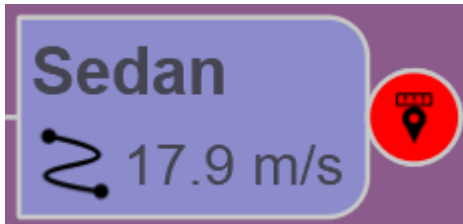
3   In the **Attributes** pane, click the **Point** attribute box. RoadRunner outlines the **Logic** editor and scenario editing canvas with blue lines, indicating that you can select an object from either section. Then, in the scenario editing canvas, select the path and then the waypoint immediately before the vehicle drives off the road.

The scenario editing canvas displays the added distance-to-point condition in blue. A dotted blue line extends from the sedan to this point.

4   Simulate the scenario. The sedan now slows down to 10 m/s as soon as it begins to drive off the road.

The **Logic** editor displays the active actions and conditions during simulation. Green actions and conditions have been completed, orange ones are currently active, and gray ones (not shown in this figure) are not active.

Other Things to Try

To customize the scenario further, try modifying the vehicle path for other path following scenarios.

Once you are satisfied with the scenario, you can export it to ASAM OpenSCENARIO. For more details on exporting to ASAM OpenSCENARIO, see "Export Scenarios".

You can also try generating variations of the scenario. For example, create a variable for the speed change. Select the speed change action phase from the **Logic** editor and, in the **Attributes** pane, right-click the **Speed** attribute and select **Create Variable**.

The **Variables** table now displays a `ChangeSpeed_TargetSpeed` variable.

You can then programmatically change this variable and export the varied scenarios to ASAM OpenSCENARIO. For more details on generating various scenarios, see "Generate Scenario Variations Using gRPC API" on page 4-2.

See Also

Related Examples

More About

Design Vehicle with Trailer Scenario

Add Vehicle with Trailer to Scene

To add a vehicle with a trailer to a scenario, follow these steps.

Note The vehicle images shown in this example are from the RoadRunner Asset Library. For more information on the RoadRunner Asset Library, see RoadRunner Asset Library product page.

1   From the **Library Browser**, drag a parent vehicle into the scene. This image shows the `SemiTruck` vehicle asset placed into the scene.

2   From the **Library Browser**, drag a trailer onto the lead vehicle in the scene. RoadRunner Scenario automatically assigns the trailer as the child vehicle to the lead vehicle, which becomes the parent vehicle. This image shows the `SemiTruck` vehicle asset with the `SemiTruck_Trailer1` vehicle asset attached as a child.

Note You can also assign the parent vehicle by using the **Parent Attachment** attribute in the Vehicle-Connection Attributes section of the **Attributes** pane of the trailer.
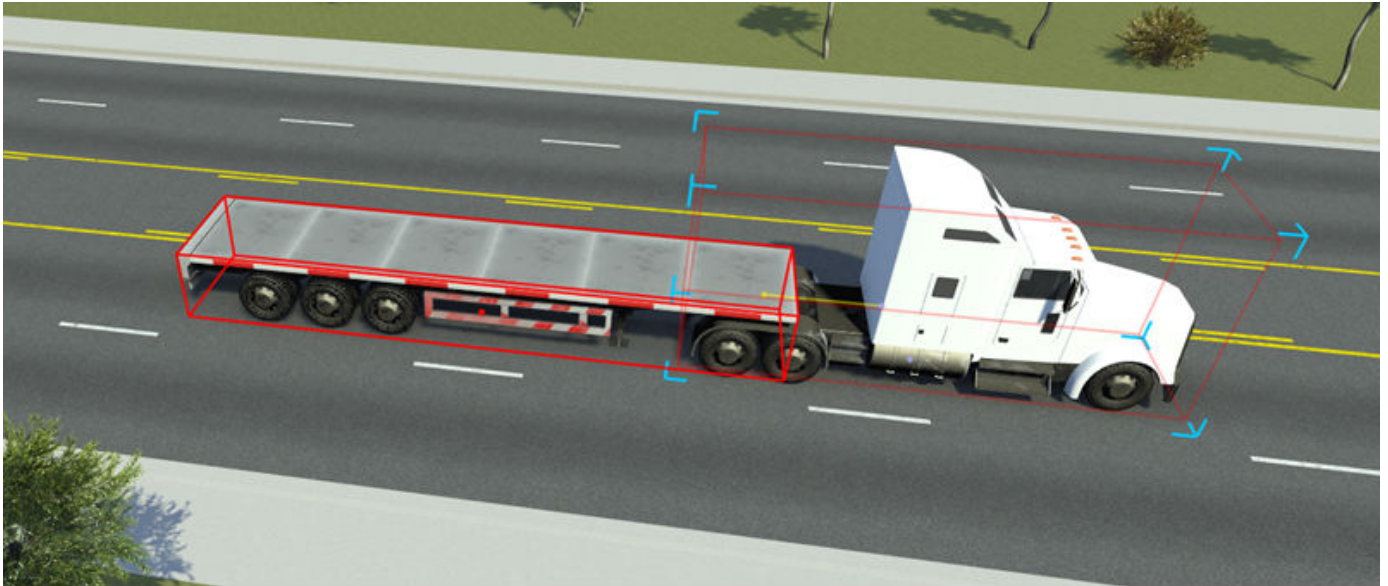
3 You can adjust the position of the trailer, relative to attachment point on the parent vehicle, by using the Parent Attachment Point attributes of the trailer. This image shows the **Forward** offset attribute of the trailer set to 1.00 m.
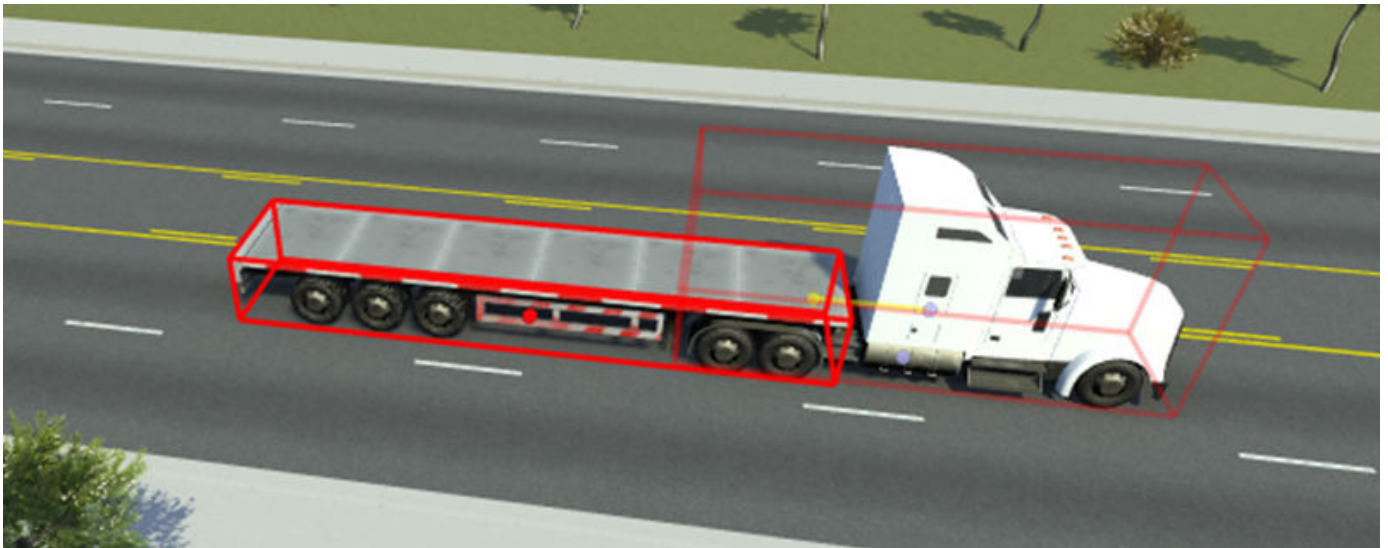
Trailers in Simulation

The trailer or child vehicle attached to a parent vehicle can behave in one of two ways: fixed to the parent vehicle, or dynamically attached to the parent vehicle through a hitch. In the trailer vehicle attributes, set **Connection Attributes > Type** to `trailer`, this animation shows a sample truck with a trailer turning a corner, with the trailer moving dynamically relative to the truck.

In the trailer vehicle attributes, set **Connection Attributes > Type** to `fixed`, this animation shows a sample truck with a trailer turning a corner with the trailer fixed to the truck.

Multi-Vehicle Trailers

A trailer can also act as a parent for another trailer. You can combine multiple vehicles as trailers to create more advanced vehicles in a scene, such as multi-trailer trucks or trailers with cars on the trailer deck. This image shows a sample of attached `Truck` and `SemiTruck_Trailer01` vehicle assets with the `Sedan` vehicle attached as a fixed child to the `SemiTruck_Trailer01` asset.

See Also

Vehicle Assets

Design Overtake Using Longitudinal Distance Condition Scenario

This example shows how to use a longitudinal distance condition to allow one vehicle to overtake another vehicle. You also learn how you can use the longitudinal distance action to build a scenario.

About the Scenario

This scenario contains two vehicles:

- The reference vehicle, a red sedan, which drives in one lane throughout the scenario.
- The ego vehicle, a white sedan, which overtakes the reference vehicle. After the overtake, the ego and reference vehicle maintain a constant longitudinal distance gap.

Create New Scenario

Create a new scenario in one of the default RoadRunner scenes.

1   Open RoadRunner and from the start page, select **Open Scene**. If you have RoadRunner open already, then select **Open Scene** from the **File** menu instead.

2   Select the `ScenarioBasic.rrscene` scene from the current project. This scene is included with RoadRunner projects by default.

   This scene contains a four-way intersection with traffic signals, a roundabout, and several roads of varying lengths and curve types.

3 Switch to scenario editing mode. From the top-right corner of the RoadRunner application, select **Scene Editing**, then **Scenario Editing**.

The scene is now locked, and you can begin creating the scenario.

Add Reference Vehicle

Add a red sedan to the scenario as the reference vehicle. This vehicle stays in the same lane throughout the scenario.

1 From the **Library Browser**, drag a `Sedan` asset into the scenario. Place the sedan on the multilane road, in any lane except the extreme left and right lanes.

2 Rename the vehicle. With the sedan still selected, set **Name** to `Reference` in the **Attributes** pane.

3 Change the color of the reference vehicle to visually differentiate it from the ego. Click the **Color** attribute box and select the red color patch. In the scenario editing canvas, the reference vehicle is now red.

The **Logic** editor now contains an initial action phase labeled **Reference**. This phase initializes the speed of the reference vehicle to approximately 17.9 m/s. Since the sedan has no driving path defined by default, it drives in the current lane at the set speed.

The scenario, by default, ends after 60 seconds of simulation time or upon the first collision.

Add Ego Vehicle

Add a white sedan as the ego vehicle to the scenario.

1   From the **Library Browser**, drag a second `Sedan` asset into the scenario. Place the new sedan in the same lane as the original sedan and slightly behind it.

2   Rename the ego vehicle. With the vehicle still selected, in the **Attributes** pane, set **Name** to `Ego`.

3   Simulate the scenario. From the RoadRunner Scenario toolbar, select the **Simulation Tool** . Then, in the **Simulation** pane, click **Play**.

    The ego vehicle and lead vehicle follow their lane and drive at their default speed of 17.9 m/s.

4   Stop the simulation early by pressing **Stop**, and then switch back to the **Scenario Edit Tool** .

Add Lane Change Action

To complete an overtake on the reference vehicle, the ego must first move to a different lane before speeding up and overtaking the reference vehicle.

Specify a lane change action that causes the ego vehicle to change lanes to prepare for the overtake on the reference vehicle.

1   Right-click the **Ego** initial phase and select **Add Action Phase After**.

2   Specify the action by which the ego vehicle moves to the lane right of the reference vehicle: in the **Attributes** pane, set **Action Type** to `Change Lane`. Enter a suitable **Name** for the action phase such as `Lane_Change_Before_Overtake`. The **Logic** editor displays the updates to the action phase.

3   Select `Current Lane` from the **Relative to** list to set up the lane change action of the ego vehicle relative to the current lane.

4   Set **Direction** to `To the right`, and **Lane Offset** to `1 lane(s)` to enable the ego vehicle to move to the right by one lane before initiating the overtake.

5   In the **Dynamics** section, set **Dynamics Type** to `Over time`. Set **Time** to `0.5 s` to indicate the time that the ego takes to complete the lane change action. Set **Dynamics Profile** to `Linear`.

6   The **Logic** editor displays the updated values. Simulate the scenario by using the **Simulation Tool** . The ego moves to its right by one lane over a time period of 0.5 seconds.
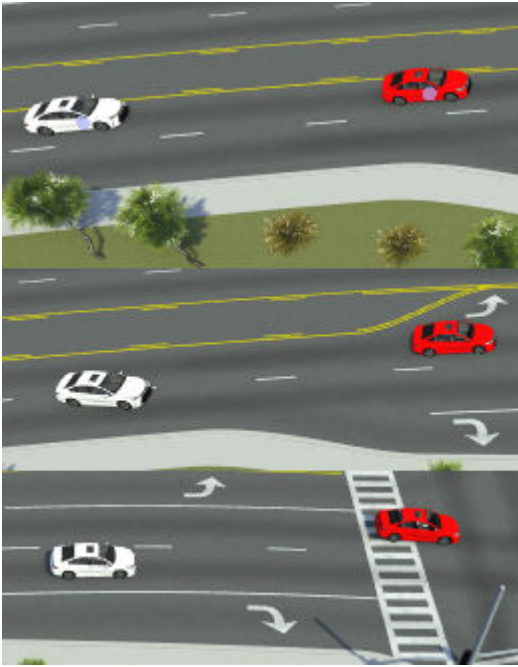
The simulation continues playing, with the ego vehicle moving forward in the lane right of the reference vehicle at the speed of 17.88 m/s.

Accelerate Ego to Complete Overtake on Reference Vehicle

Create a new action phase that accelerates the ego vehicle so that it can build speed to overtake the reference vehicle. The overtake occurs on the lane right of the reference vehicle.

1   Right-click the previous action phase `Lane_Change_Before_Overtake`, and select **Add Action Phase After**. Select an appropriate **Name** such as `Complete_Overtake` for the newly created action phase.

2   To accelerate the ego vehicle, set the **Action Type** to `Change Speed`. Set the **Actor** to `Ego`.

3   In the **Change Speed** section, set **Relative To** to `Absolute`. Set the new **Speed** to a greater value, such as `35 m/s` to enable the ego to speed up and overtake the reference vehicle.

4
    Simulate the scenario by using the **Simulation Tool** . After moving to the right, the speed of the ego vehicle increases. The ego passes the reference vehicle, thereby completing the overtake action.

Use Longitudinal Distance Condition to Determine Return of Ego to Original Lane

Once the ego vehicle completes the overtake, the longitudinal distance condition enables you to decide the distance threshold to the reference vehicle at which the ego moves back to its original lane in front of the red sedan.

1   Click the small circle at the end of the previous action phase `Complete_Overtake` to create a new condition. In the **Attributes** pane, under **Add Condition**, select `Longitudinal Distance to Actor`.

2   Set **Relative Position** to `Ahead of`, and select `Reference` as the actor from which the relative position is calculated. The **Rule** and **Threshold** values together determine the distance threshold at which the condition is considered satisfied. For example, if you set **Rule** and **Threshold** to `Greater than` and **3.00 m**, respectively, then `Ego` satisfies this condition when its longitudinal distance gap to the reference actor is greater than 3.00 m ahead of `Reference`.

Return Ego to Original Lane

Create a new action phase that moves the ego vehicle to the original lane after the overtake is complete.

1   Right-click the previous action phase `Complete_Overtake`, and select **Add Action Phase After**. Select an appropriate **Name** such as `Change_Lane_Back` for the newly created action phase. Note that the simulation reaches this action phase only after the longitudinal distance condition is satisfied following the overtake.

2   Set the **Action Type** to `Change Lane`. Select the **Actor** for which this action type is applicable as `Ego`.

3   In the **Change Lane** section, program `Ego` to move one lane to the left relative to the current lane in a linear manner, over a time period of one second:

   • Set **Relative To** to `Current Lane`.
   • Set **Direction** to `To the left`.
   • Set **Lane Offset** to `1 lane(s)`.
   • In the **Dynamics** section, set **Dynamics Type** to `Over time`.
   • Set **Time** to `1.00 s`.
   • Set **Dynamics Profile** to `Linear`.

4   Simulate the scenario by using the **Simulation Tool** . When the `Longitudinal Distance to Actor` condition is satisfied, the ego moves back to its original lane ahead of the red sedan using the specified dynamics.

Maintain Constant Longitudinal Distance Between Ego and Reference Vehicle

Create a new action phase that closes the gap between the ego and reference vehicle after the overtake, and maintains a constant longitudinal distance between both vehicles.

1   Right-click the previous action phase `Change_Lane_Back`, and select **Add Action Phase After**. Select an appropriate **Name** such as `Maintain_Long_Distance` for the newly created action phase.

2   Set the **Action Type** to `Change Longitudinal Distance`. Set the **Actor** to `Ego`.

3   In the **Change Longitudinal Distance** section, configure the ego vehicle to stay at a constant longitudinal distance of `8.00 m` in front of the reference vehicle:

- Set **Relative Position** to Ahead of.
- Set **Reference Actor** to Reference.
- Set the **Distance Type** to Space.
- Set the **Space Distance Offset** to 8.00 m.

4 Set **Measure From** to Bounding Boxes if you want the longitudinal distance to be measured between the edges of the bounding boxes of the ego and the reference vehicle. Select Origins instead if you want the longitudinal distance to be measured between the origins of the ego and the reference vehicle.

5 Set **Sampling Mode** to At start of action if you want the target longitudinal distance gap to be achieved in the current action phase itself. Select Continuous if the target longitudinal distance gap can also be achieved outside the current action phase.

6 In the **Dynamic Constraints** section, set **Constraint Type** to Asset Constraints. The ego vehicle must follow the constraints included in its own list of attributes.

7
Simulate the scenario by using the **Simulation Tool** . After the overtake is complete, a constant longitudinal distance is maintained between Ego and Reference.

See Also

Related Examples

- "Design Lane Following Scenario" on page 3-2
- "Design Lane Change Scenario" on page 3-10
- "Design Path Following Scenario" on page 3-28

More About

- "Define Scenario Logic" on page 3-70

Design Vehicle Following User-Defined Actions Scenario

You can use either MATLAB or Simulink to modify actor behavior by applying user-defined actions created in RoadRunner Scenario.

The example workflows on this page assume that:

- You have a RoadRunner license and the product is installed.
- You have a RoadRunner Scenario license and the product is installed.

Model Vehicle Behavior Using User-Defined Actions in MATLAB

You can design a scenario in which a vehicle's behavior is modified in MATLAB by a user-defined action created in RoadRunner Scenario.

The custom parameters of the user-defined action are dispatched from RoadRunner Scenario to actor behavior models in cosimulation clients like MATLAB. Here, you can use the custom parameters to modify base actor parameters like pose or velocity, thereby controlling the overall movement of a vehicle.

This example shows how custom parameters (such as **Steering Angle**) of a user-defined action are used to modify actor behavior.

You also learn how to create an actor behavior model in MATLAB that can receive and use this user-defined action to modify actor behavior in a meaningful way. For instance, the **Steering Angle** custom parameter can be programmed to control the angle of movement of a vehicle.

Set Up MATLAB-RoadRunner Scenario Cosimulation Environment
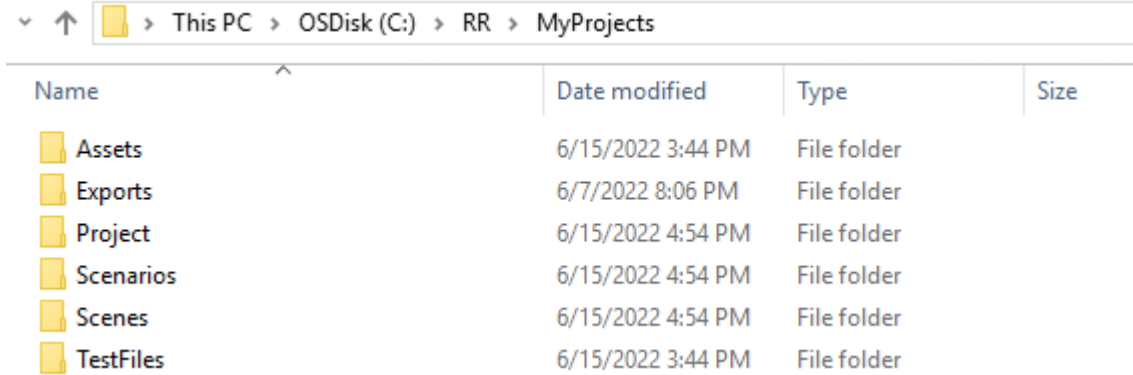
In the MATLAB command prompt, specify the path to your local RoadRunner installation folder. This code snippet uses the default installation path of the RoadRunner application on Windows.

```
RRInstallationFolder = "C:\Program Files\RoadRunner R2022b\bin\win64";
```

To update the path for the RoadRunner installation folder, get the root object within the settings hierarchical tree. Then, use the root object to set the installation path of the RoadRunner application. For more information, see `SettingsGroup` (MATLAB).

```
s = settings;
s.roadrunner.application.InstallationFolder.PersonalValue = RRInstallationFolder;
```

This code snippet shows the path to a sample project folder on Windows that contains the default RoadRunner directory structure.

If required, change the path to reflect the RoadRunner project path on your machine.

`rrProjectLocation = "C:\RR\MyProjects";`

Create and open the `roadrunner` object that represents the specified project.

`rrApp = roadrunner(rrProjectLocation, "InstallationFolder", RRInstallationFolder);`

Add these files to the appropriate folders within your RoadRunner project.

- `ScenarioBasic_UDA_ML.rrscene` — Scene file for the MATLAB-RoadRunner cosimulation example.
- `SimpleUDA_ML_v0.rrscenario` — Scenario file based on `ScenarioBasic_UDA_ML.rrscene`.
- `CustomDriveAction.rraction.rrmeta` — Action asset file that contains the list of custom parameters of a user-defined action with associated values. Here, the name of the user-defined action is **CustomDrive**. The initial values of custom parameters **Steering Angle** and **ThrottleLevel** are 0 and 30, respectively. This file is required for scenario simulation by both MATLAB and Simulink. For more information about creating an action asset file, see "User-Defined Actions" on page 3-92.
- `hUDA_ML.m` — MATLAB System object file that processes the user-defined actions.
- `hUDA_ML.rrbehavior.rrmeta` — Behavior asset file that links the MATLAB System object behavior to the vehicle in the scenario.

```
copyfile("ScenarioBasic_UDA_ML.rrscene",fullfile(rrProjectLocation,"Scenes"));
copyfile("SimpleUDA_ML_v0.rrscenario",fullfile(rrProjectLocation,"Scenarios"));
copyfile("CustomDriveAction.rraction.rrmeta",fullfile(rrProjectLocation,"Assets","Actions"));
copyfile("hUDA_ML.m",fullfile(rrProjectLocation,"Assets","Behaviors"));
copyfile("hUDA_ML.rrbehavior.rrmeta",fullfile(rrProjectLocation,"Assets","Behaviors"));
```

Explore Scenario

Open the scene `ScenarioBasic_UDA_ML.rrscene` for the MATLAB-RoadRunner Scenario cosimulation example.

`openScene(rrApp, "ScenarioBasic_UDA_ML");`

Open the scenario file `SimpleUDA_ML_v0.rrscenario`.

`openScenario(rrApp,"SimpleUDA_ML_v0");`

The scenario contains a red ambulance stationary in a lane.

Visualize the scenario logic in the **Logic** editor pane. For more information, see "Define Scenario Logic" on page 3-70.

The ambulance moves at an initial absolute speed of 17.88 m/s.

In the second action phase, the value of custom parameter **Throttle Level** is increased from 30 to 60.
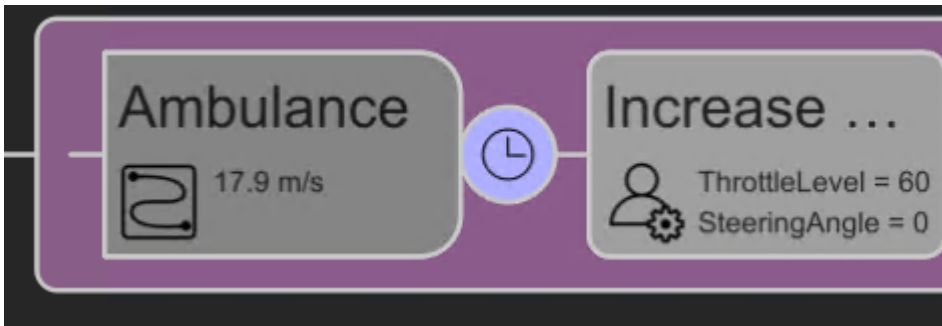
In the final action phase, the custom parameter **Steering Angle** of the red ambulance is increased from 0 to 20.
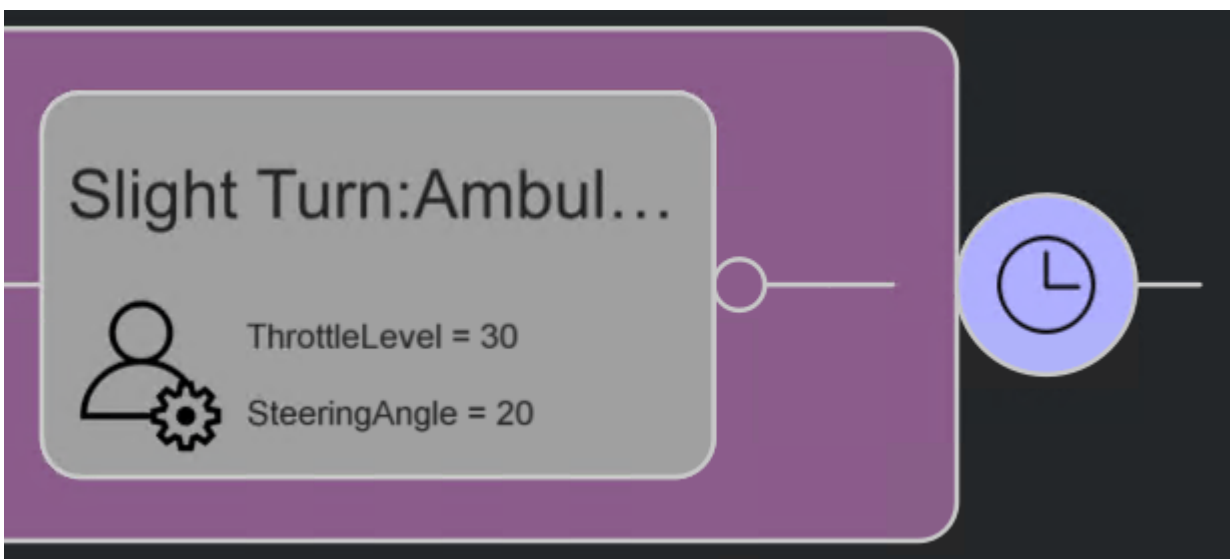
Now, navigate to **Assets > Behaviors** in the **Library Browser**.

Click the behavior file named `hUDA_ML.rrbehavior`. In the **Attributes** pane on the right, note that the **Platform** is set to **MATLAB/Simulink**. The MATLAB actor behavior file name `hUDA_ML.m` is present in the **File Name** text box. For more information about modeling actor behavior in MATLAB using user-defined actions, see "Actor Behavior Using User-Defined Actions" (Automated Driving Toolbox).
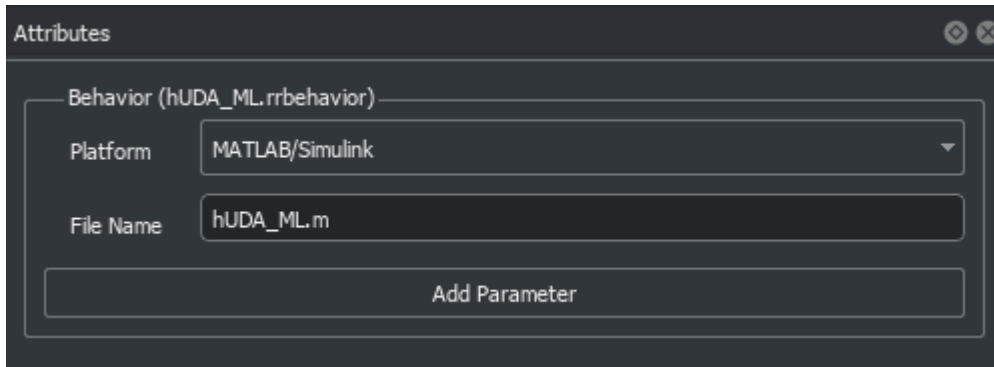
Click on the red ambulance in the Scenario Edit Tool. In the **Attributes** pane, note that the behavior asset file `hUDA_ML.rrbehavior.rrmeta` is linked to the red ambulance.
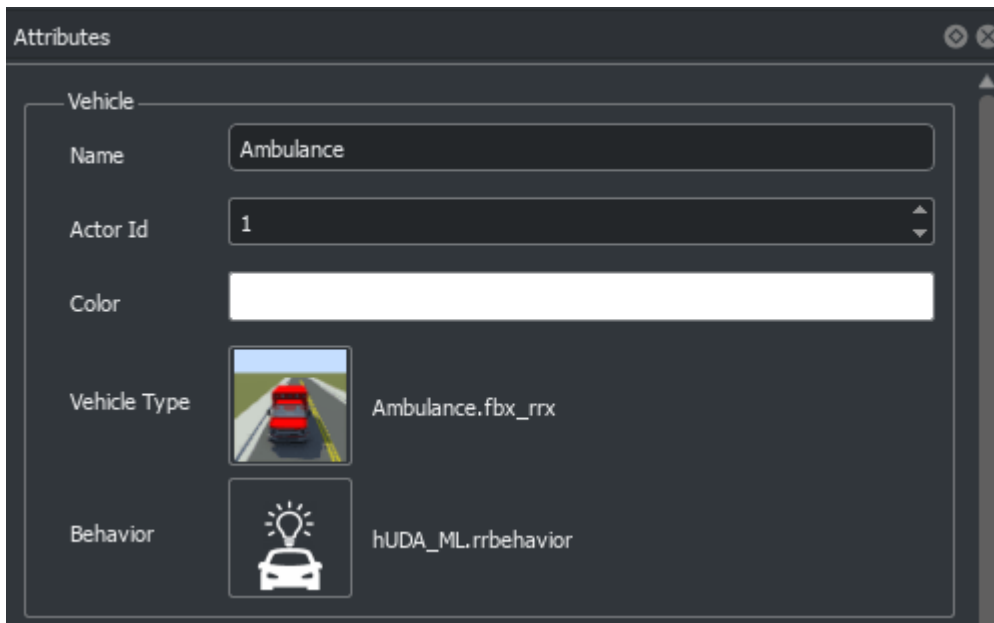
Connect to the RoadRunner Scenario server to enable cosimulation by using the createSimulation function.

```
ss = rrApp.createSimulation();

Warning: Undefined function 'attachhdlcconfig' for input arguments of type
'double'.
Warning: Unrecognized function or variable
'realtime.internal.TargetHardware.refresh'.
Connection status: 1
Connected to RoadRunner Scenario server on localhost:53777, with client id {dda1a2d0-0742-4368-8(
```

Simulate Scenario

Play the scenario.

```
ss.set("SimulationCommand", "Start");
```

When the scenario is played, the actor behavior programmed in hUDA_ML.m is implemented.

The ambulance veers to its right and crosses the road limits onto the grassy surface. Then, it moves left as the Steering Angle is increased to 20.

Model Vehicle Behavior Using User-Defined Actions in Simulink

This example shows how to set Simulink up to process user-defined actions, and modify the movement of an actor accordingly.

Set Up Simulink-RoadRunner Scenario Cosimulation Environment

This section shows how to set up a cosimulation environment between RoadRunner Scenario and Simulink.

Specify the path to your local RoadRunner installation folder.

```
RRInstallationFolder = "C:\Program Files\RoadRunner R2022b\bin\win64";
```

Get the root object within the MATLAB settings hierarchical tree, and use it to set the installation path of the RoadRunner application.

```
s = settings;
s.roadrunner.application.InstallationFolder.PersonalValue = RRInstallationFolder;
```

Set the path to your RoadRunner project folder.

```
rrProjectLocation = "C:\RRScenario\MyProjects";
```

Create and open the roadrunner object representing the specified project.

```
rrApp = roadrunner(rrProjectLocation, "InstallationFolder", RRInstallationFolder);
```

Run commands to add the following files to your RoadRunner project.

- ScenarioBasic_UDA_SL.rrscene — Scene file for the Simulink-RoadRunner Scenario cosimulation example.

- `buildingUDAScenarioFromScratch_v2.rrscenario` — Scenario file based on `ScenarioBasic_UDA_SL.rrscene`.
- `CustomDriveAction.rraction.rrmeta` — Action asset file that contains the list of custom parameters and values of a user-defined action.
- `UDA_SL.rrbehavior.rrmeta` — Behavior asset file that links a Simulink actor behavior model to a vehicle in the scenario.

```
copyfile("ScenarioBasic_UDA_SL.rrscene",fullfile(rrProjectLocation,"Scenes"));
copyfile("buildingUDAScenarioFromScratch_v2.rrscenario",fullfile(rrProjectLocation,"Scenarios"));
copyfile("CustomDriveAction.rraction.rrmeta",fullfile(rrProjectLocation,"Assets","Actions"));
copyfile("UDA_SL.rrbehavior.rrmeta",fullfile(rrProjectLocation,"Assets","Behaviors"));
```

The following required files are present in the example folder.

- `hUDA_SL.slx` — Simulink actor behavior model that processes user-defined actions.
- `BusCustomDriveAction.mat` — MAT file that maps parameters entered for a user-defined action in RoadRunner Scenario to the fields of a bus structure. Required as input by the Simulink actor model behavior.

To check how the above files are created, see "Simulate RoadRunner Scenarios with Actors Modeled in Simulink" (Automated Driving Toolbox).

Explore Scenario

Open the RoadRunner Scenario scene `ScenarioBasic_UDA_SL.rrscene` to run the Simulink-RoadRunner Scenario cosimulation.

```
openScene(rrApp, "ScenarioBasic_UDA_SL");
```

Open the scenario file `buildingUDAScenarioFromScratch_v2`.

```
openScenario(rrApp,"buildingUDAScenarioFromScratch_v2");
```

The scenario contains a red ambulance stationary in a lane.

View the scenario logic in the **Logic** editor pane.

The ambulance moves at an initial absolute speed of 17.88 ms. In the second action phase, the custom parameter **Throttle Level** is increased to 10, while the **SteeringAngle** is 0. In the final action phase, the **Throttle Level** of the ambulance is increased to 40, and the **SteeringAngle** of the ambulance is increased to 35.

Navigate to **Assets > Behaviors** in the **Library Browser**, and click on the behavior named UDA_SL. In the **Attributes** pane on the right, note that the **Platform** is selected as **MATLAB/Simulink**. The MATLAB actor behavior file name, hUDA_SL.slx is present in the File Name text box.
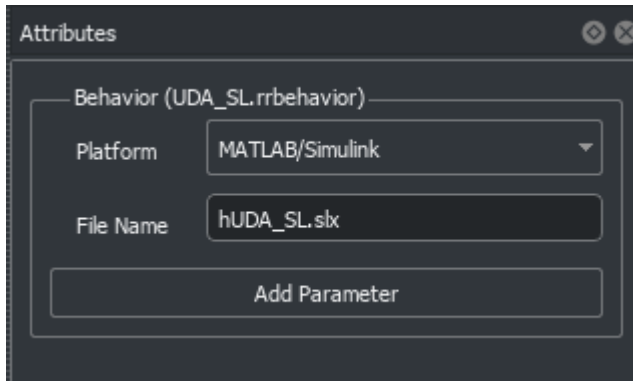
Click on the red ambulance in the Scenario Edit Tool. In the Attributes pane, this behavior asset file UDA_SL.rrbehavior is linked to the red ambulance.
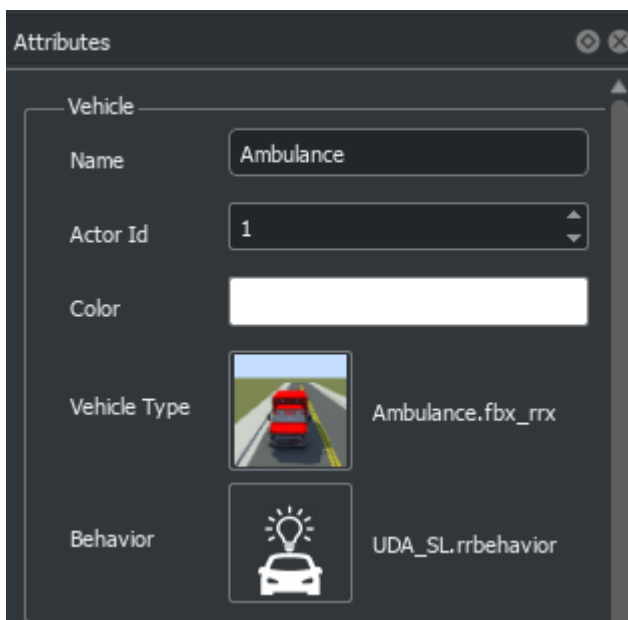
Connect to the RoadRunner Scenario server to enable cosimulation by using the createSimulation function.

```
ss = rrApp.createSimulation();
```

Simulate Scenario

Play the scenario, buildingUDAScenarioFromScratch_v2.

```
ss.set("SimulationCommand", "Start");
```

When the scenario is played, the ambulance moves sharply to left as the Steering Angle is increased to from 0 to 35.
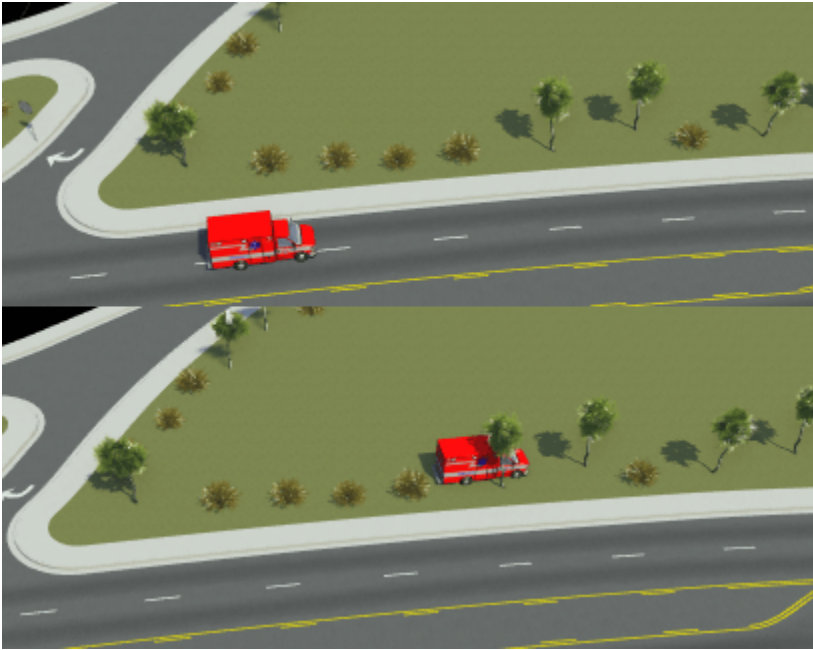
See Also

Related Examples

- "Design Lane Change Scenario" on page 3-10
- "Design Lane Swerve Scenario" on page 3-20
- "Design Path Following Scenario" on page 3-28

More About

- "Define Scenario Logic" on page 3-70

Switch Between Scene and Scenario Editing

When you have a RoadRunner Scenario license, you can switch between scene and scenario editing.
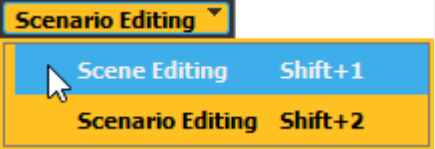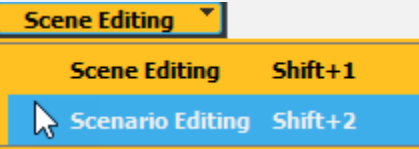
- Scenes are composed of static elements, such as roads, lanes, props, and terrain. Scene editing requires only a RoadRunner license, and is the default editing mode in RoadRunner.
- Scenarios are composed of dynamic elements, such as moving vehicles. Scenarios are built on top of scenes, and scenario editing requires a RoadRunner Scenario license. If the RoadRunner Scenario license is not present, then attempting to switch to scenario editing mode results in an error.

In scene editing mode, you can create a scene, add and delete static assets, and edit the lanes and maneuver roads. In scenario editing mode, you can add or delete vehicles on the road, add anchors, and specify the paths of vehicles.

Scene Editing	Scenario Editing
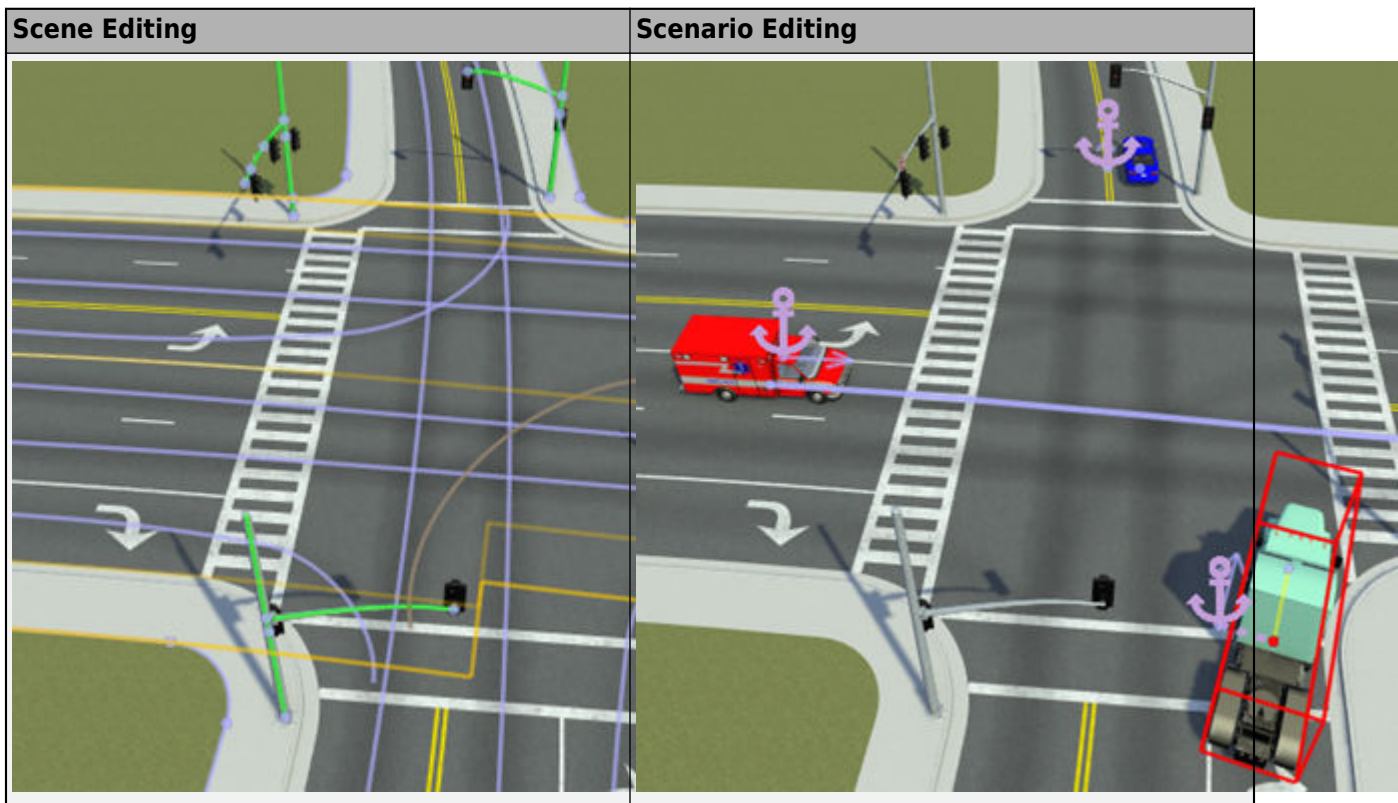
Switch Between Editing Modes

To switch between editing modes, use the yellow toggle button in the upper-right corner of the RoadRunner application. The menu has **Scene Editing** and **Scenario Editing** options, and you can toggle between the two modes.

Switch to Scene Editing	Switch to Scenario Editing
To switch to scene editing, when the yellow toggle button says **Scenario Editing**, click it and select **Scene Editing**.	To switch to scenario editing, when the yellow toggle button says, **Scene Editing**, click it and select **Scenario Editing**.

Alternatively, you can switch between editing modes by using the keyboard shortcuts. Press **Shift+1** to switch to scene editing mode and **Shift+2** to switch to scenario editing mode.

Note Switching between editing modes resets the undo/redo stack.

How Scene Editing Affects Scenarios

Scenarios are built on top of scenes. Therefore, editing a scene can significantly affect scenarios. For example, modifying the road network can change which lanes the vehicles drive on or change to. Open the `TrajectoryCutIn` scenario, which is included in the `Scenarios` folder of new RoadRunner projects. In this scenario, the `Ego` vehicle cuts into the lane of the `Lead` vehicle.

Scenarios open with the scene they were previously saved with, which in this case is the `ScenarioBasic` scene. This scene is included in the `Scenes` folder of new RoadRunner projects. Switch to scene editing mode and delete one of the driving lanes by using the **Lane Tool**.
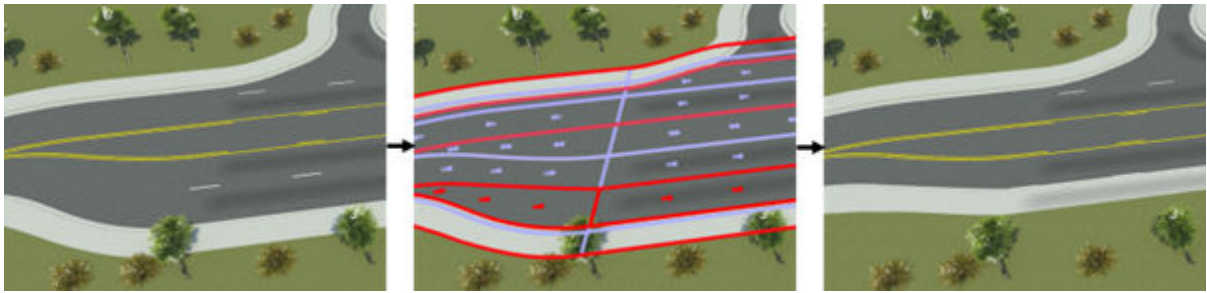
Switch back to scenario editing mode and open the `TrajectoryCutIn` scenario into this scene by opening the **File** menu and selecting **Open Scenario Into Current Scene**. The white vehicle no longer cuts-in, because it has no second lane to cut into.

How Scenario Editing Affects Scenes

In most cases, the edits that you make to a scenario do not affect the active scene. For example, actors in a scenario are not part of a scene, so modifying an actor path or position does not prompt you to save changes to the scene.

Road anchors, however, are considered part of the scene, and are active during scenario editing. Road anchors are points along roads that actors on the road attach to. Changes to road anchors, such as moving or renaming them, are saved to the scene, and can affect scenarios that open into these scenes.

Moving Road Anchors

When you move an anchor, the actors anchored to it move with it, as shown in these figures. In

Scenario Editing mode, click the **Show All Anchors** button to view all anchors in the scenario. In these figures, the `Ego` and the `Lead` vehicles are anchored to the road anchor. Moving the anchor along the road moves the two vehicles along the road, maintaining their relative positions to the anchor.
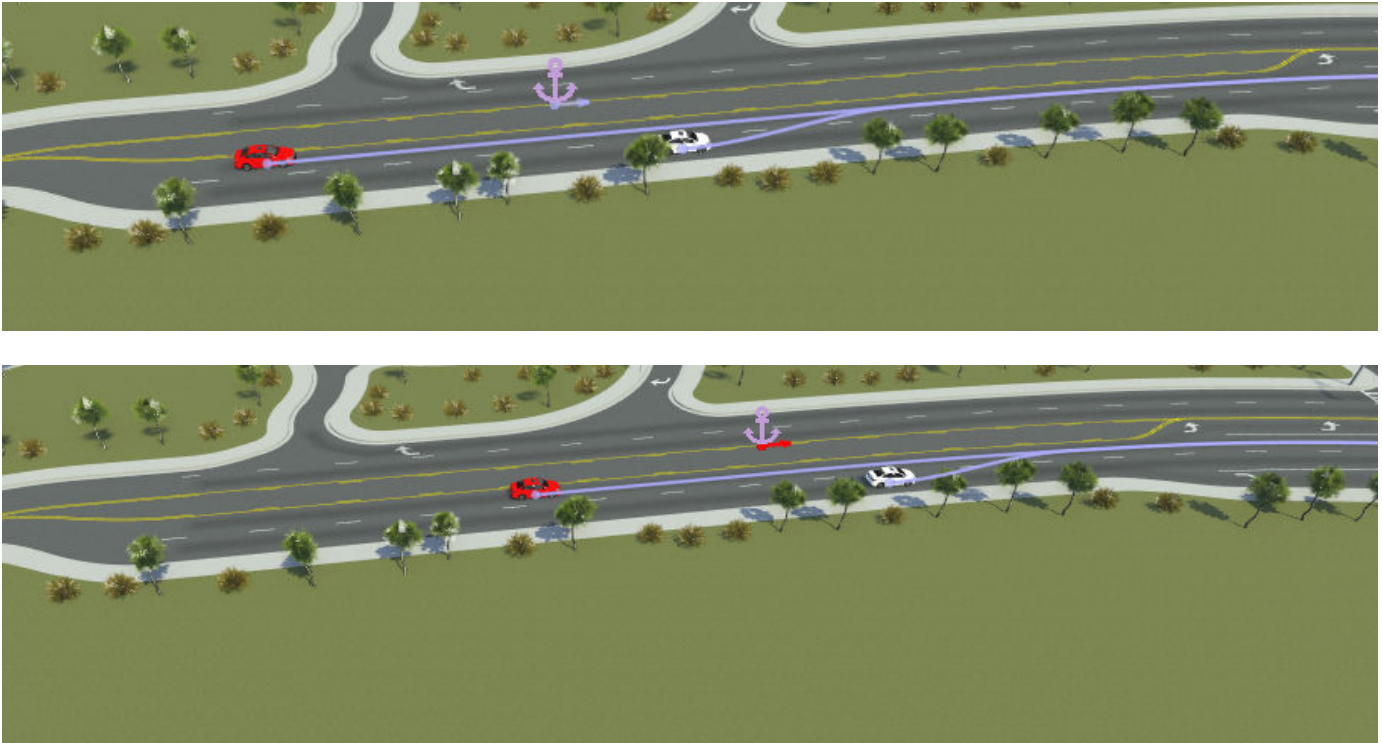
If you move a road anchor within a scenario, and then switch to scene editing mode and select the **Road Anchor Tool**, you can see the new position of the anchor reflected in the scene.

Moving an anchor in a scene can affect the starting positions of any actors that attach to it. When moving road anchors, keep in mind how scenarios that use the scene might be affected by the move.

Renaming Road Anchors

Changes to road anchor names are also saved to the scene. If you rename a road anchor in a scene, any scenario actors attached to that anchor become unanchored in that scene. To resolve this issue, you must update the anchor parent of each actor that was attached to the anchor. For more details, see "Change Anchor Parent" on page 3-114.

See Also
Road Anchor Tool

More About
- "Relocate Scenarios" on page 3-117
- "Scenario Anchoring System" on page 3-108

Path Editing

During simulation, actors in a scenario move using either their built-in behavior or by moving along a path that you specify for them. This scenario shows a sample path for a vehicle that includes segments following the road and one segment that goes off-road.

Add Path Along Driving Lane

To add a path for an actor along its driving lane, follow these steps:

1   In the scenario editing canvas, click an actor to select it.

2   Right-click along the driving lane of the actor.

The added path segment extends from the actor to the point where you right-clicked and ends with a path waypoint.

By default, paths added onto a road network snap to the center of the lane. You can then extend this path by clicking and dragging the waypoint, and the path remains snapped to the lane center.

Create Lane Changes
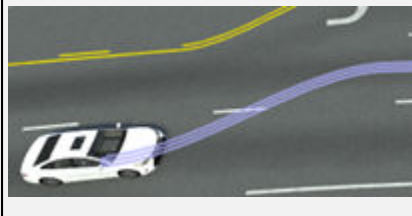
To add a lane change to a path, drag a waypoint into another lane.

You can drag waypoints across multiple lanes.
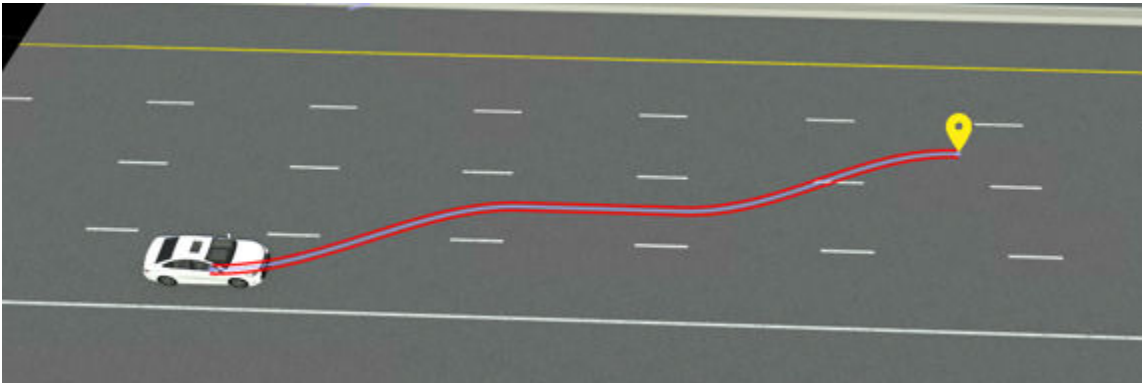
The generated path remain snapped to the lane centers. By default, lane changes take place over a maximum distance of 20 meters. To modify this distance, select the path or any segment or waypoint along it, and in the **Attributes** pane, modify the **Lane Change Distance** attribute. Lane change distances apply to the entire path. This table shows sample lane change distance values.

Lane Change Distance = 10 meters	Lane Change Distance = 20 meters	Lane Change Distance = 30 meters

Extend Path with Additional Segments

To extend paths by adding new path segments, follow these steps:

1 Click the path to you want to extend to select it. Alternatively, click a segment or waypoint along that path.

2 Right-click the location that you want to extend the path to.

A new path segment extends from the previous waypoint and ends in a new waypoint at the point where you right-clicked.
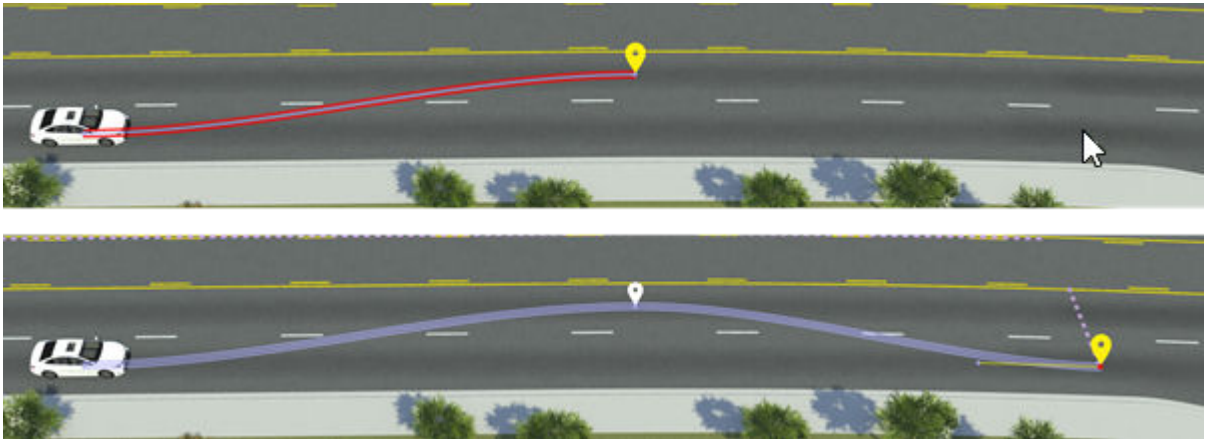
Split Path into Separate Segments

To split a path or path segment into separate segments, right-click within the path to add a new waypoint,
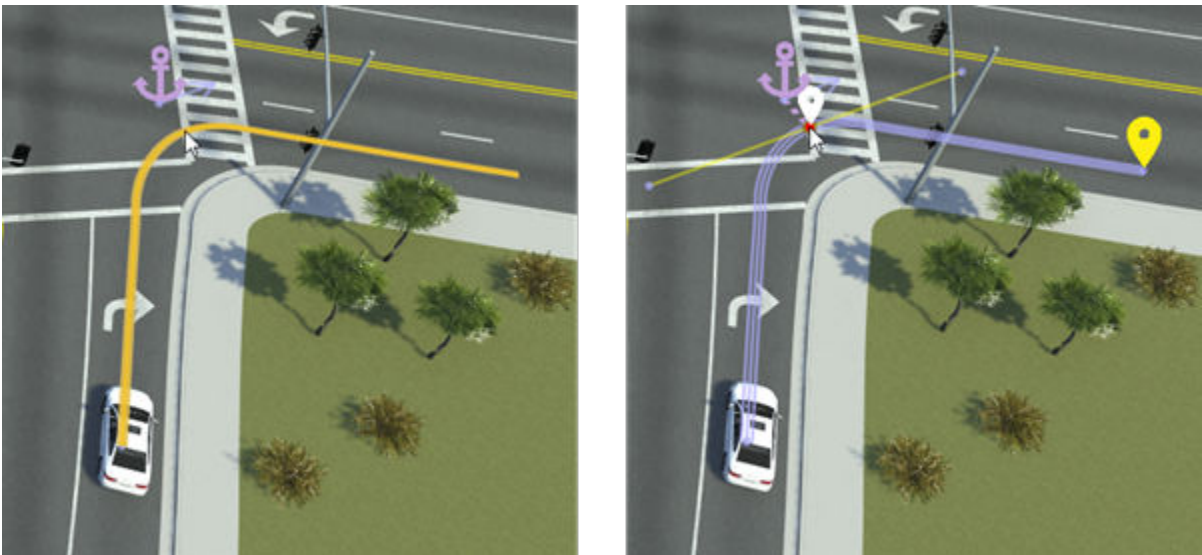
Modify Path Tangents

To change the shapes of paths, you can modify the tangents of path waypoints. To modify path tangent, click a waypoint to select it, and then click and drag the tangent lines to set the direction and scale of the tangent.
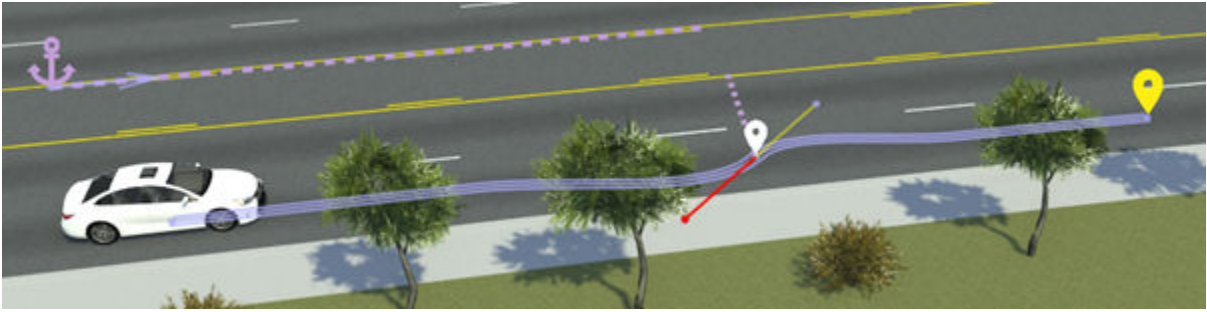
For more details on working with tangents, see "Tangent Editing".

Set Specific Path Lengths

To set specific lengths of path segments, select a path waypoint and in the **Attributes** pane, under **Forward Offset**, modify the **Offset** value. This value sets how many meters the waypoint is offset from its parent anchor.

By default, path segments are anchored to the road. If you want to set path lengths relative to the vehicle, then you must change the anchor parent of the waypoints.

Consider this vehicle path containing two path waypoints. The waypoints are 25 meters and 50 meters in front of the road anchor, respectively.

Suppose you want to set the intermediate waypoint 10 meters in front of the vehicle. Changing only the **Offset** value of this waypoint offsets it from the road anchor, not the vehicle.

To offset the waypoint 10 meters from the vehicle, you must first change its anchor parent from the road anchor to the vehicle, and then set the offset value, by following these steps:

1 Select the waypoint.

2 From the **Attributes** pane, select the name of the parent anchor from the **Anchor** attribute box.

3 Select the vehicle from either the scenario editing canvas or the **Logic** editor. RoadRunner outlines these areas in blue lines.

4 Set **Offset** to 10 meters.

This image shows the updated offset values. If you drag the vehicle, the first waypoint remains fixed 10 meters in front of the vehicle. The second waypoint remains fixed in place, because it is still anchored to the road. If you wanted the second waypoint to remain fixed 50 meters in front of the vehicle, you must change its anchor parent to the vehicle as well.
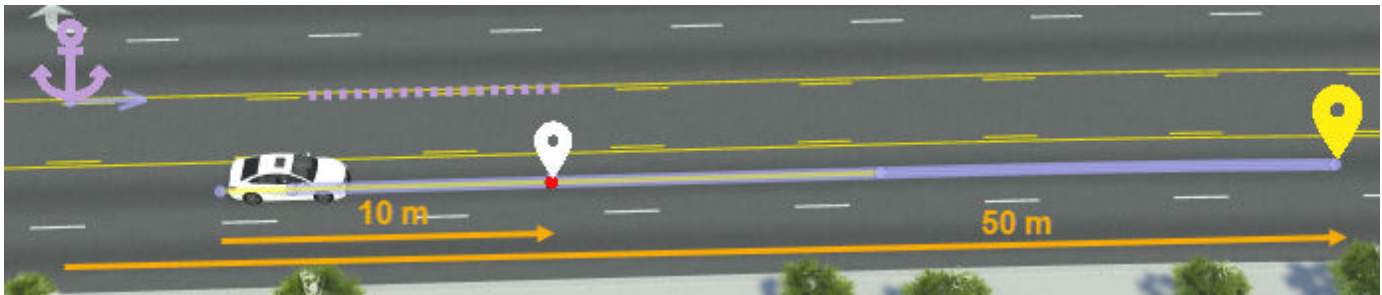
For more details on working with anchors, see "Scenario Anchoring System" on page 3-108.

Set Precise Waypoint Locations

By default, path waypoints lock to an anchor. If you disable anchoring, you can set more precise (x,y,z) locations of path waypoints. Follow these steps:
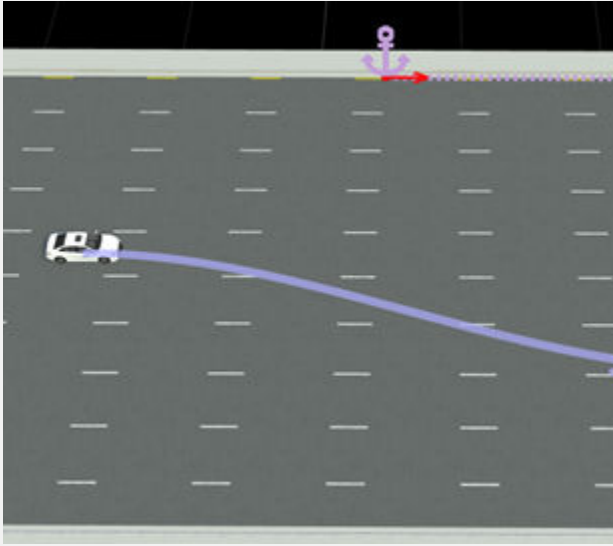
1 In the scenario editing canvas, select the path waypoint you want to set the precise location for.

2 In the **Attributes** pane, under **Point Offsets**, clear the **Enable Anchoring** attribute.

3 Under **Position**, modify the **X**, **Y**, and **Z** coordinate values. These values are relative to the scene origin.

Shift Paths Within Lanes

To shift a path laterally within a lane, hold the **Ctrl** key and then drag the path waypoint you want to shift. This image shows a path segment shifted to the curb to simulate a parking maneuver.
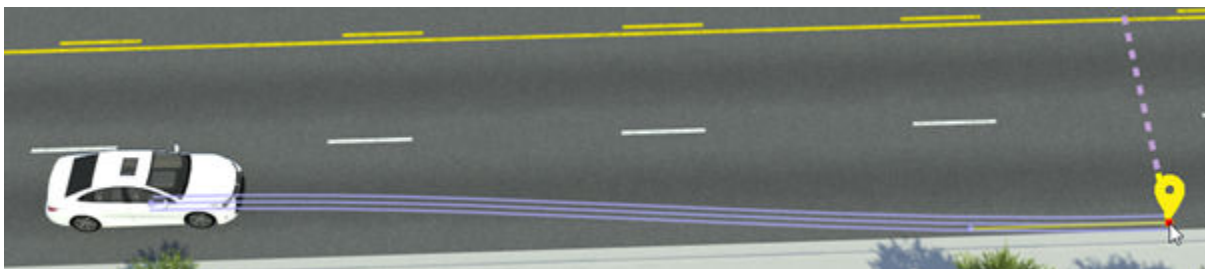
To set more precise lane-relative locations of waypoints, modify the waypoint attributes in the **Lane Offset** section of the **Attributes** pane. This table shows sample lane offset attributes for a waypoint.

Lane Offset Attributes	Description
Relative To — `Road Edge` **Offset From** — `Right Lane` **Direction** — `2 lane(s)` **Travel Direction** — `With Road Anchor` **Lateral Offset** — `1.5` meters	Set waypoint two lanes from the road edge and in the same travel direction of the road anchor. Offset the lane 1.5 meters to the right of the lane center.

Create Free-Form Paths

All path segments so far have followed the road network. To create more free-form paths that go on and off the roads, you can have path segments ignore the road network.

1   Click a path segment to select it.

2   In the **Attributes** pane, select **Freeform** under **Route Segment Parameters**.

When you drag the waypoint at the end of the segment that ignores the road network, the segment no longer snaps to the center of the lane. You can now more easily create U-turn or off-road paths. For example, in this image, the second and fourth path segments ignore the road network.
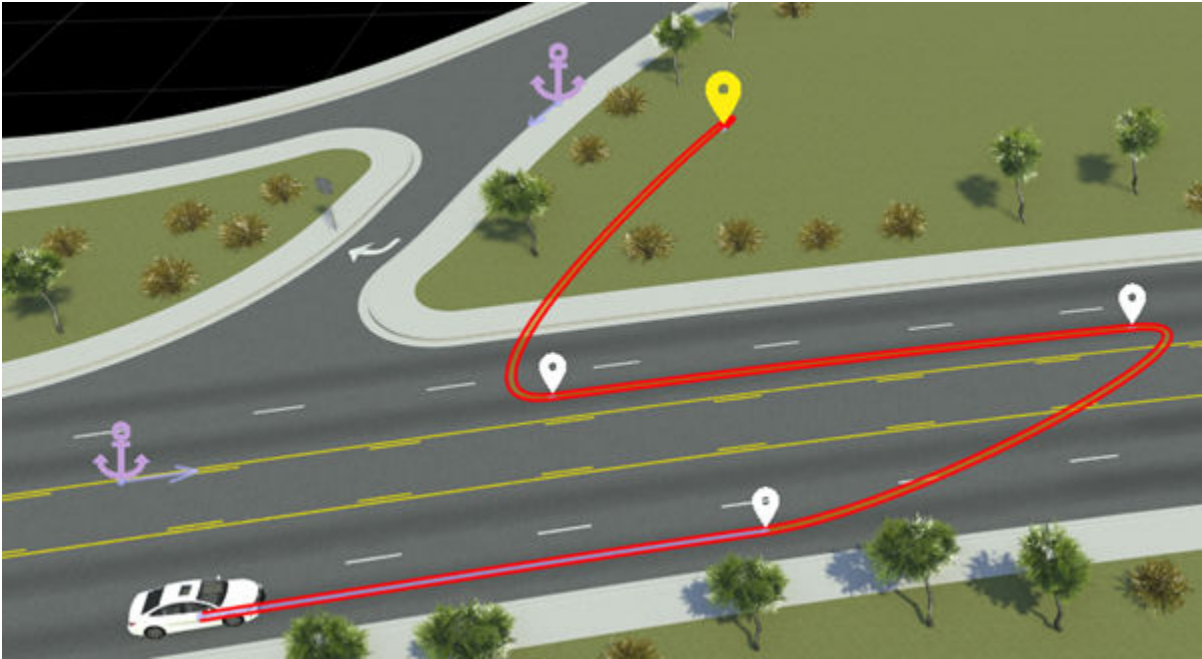
If you modify these segments to follow the road network, then they follow the road to reach the desired waypoints, leading to long and winding paths. For example, this image shows the second path segment following the road network. To reach the opposite side of the highway, the segment follows the long loop around the intersection.
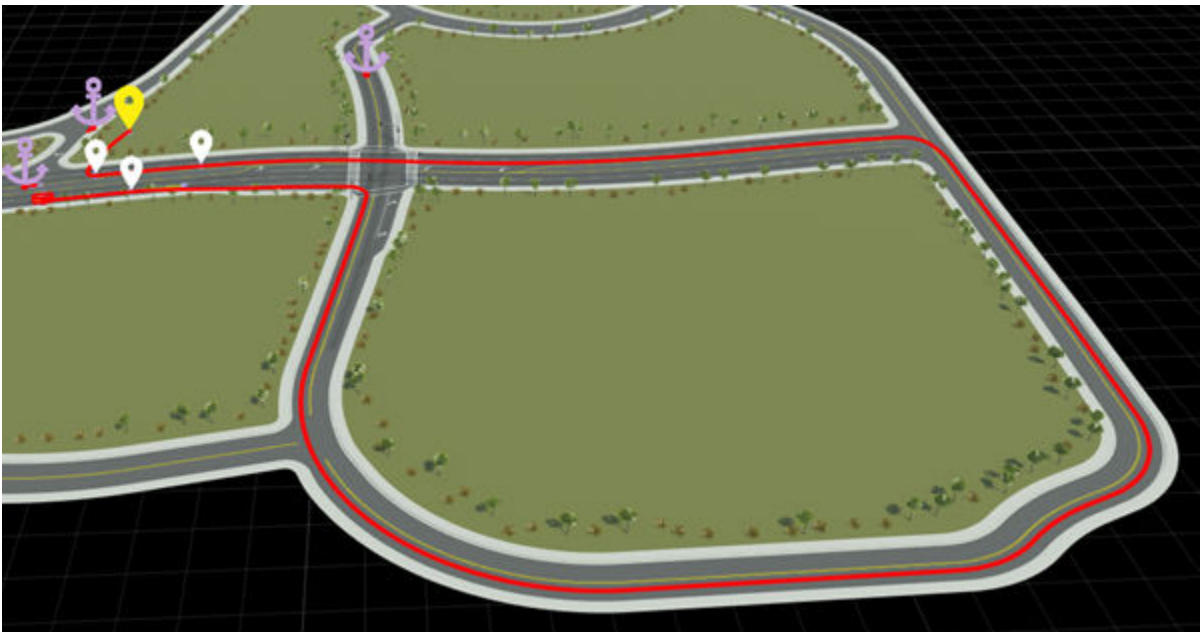
For additional control over segments that ignore the road network, you modify their curve types. In the **Attributes** pane, change the **Curve Type** attribute of the selected segments. This table describes the curve types.

Curve Type	Description
Cubic (default)	Path segment follows a cubic polynomial curve.
Clothoid	Path segment follows a clothoid curve, where the curve changes linearly with distance.

See Also

Related Examples

- "Design Path Following Scenario" on page 3-28

More About

- "Scenario Anchoring System" on page 3-108

Define Scenario Logic

RoadRunner Scenario provides a graphical interface for defining the logic of a scenario. This graphical **Logic** editor is available from the **2D Editor** pane. The scenario logic defined in this editor consists of a series of actions, with optional conditions that trigger those actions. Actions and conditions can also occur in parallel, enabling you to build complex scenarios containing multiple actors that have different goals. Consider the logic for a scenario containing a car and a truck that perform lane and speed change actions.
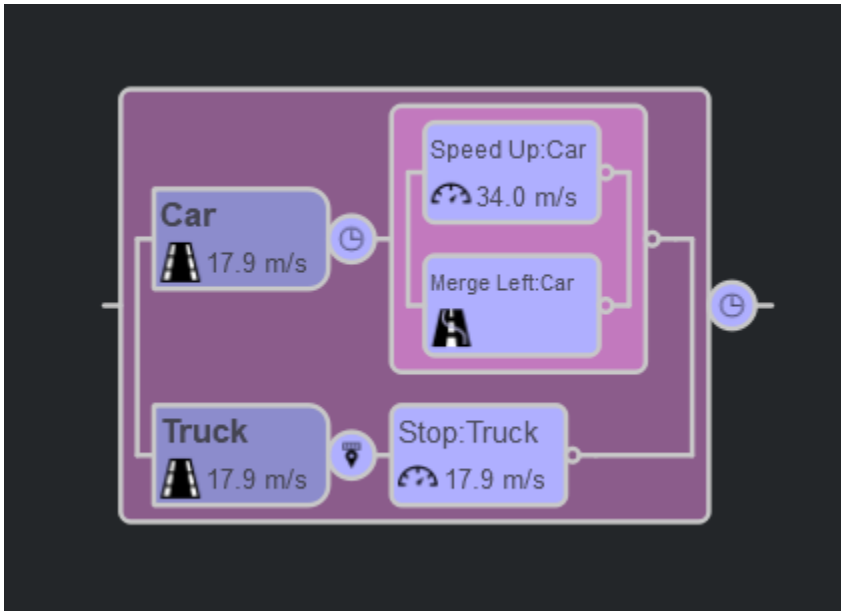
The boxes are action phases. The first action phase for each vehicle specifies the initial state of the vehicles at the start of simulation.
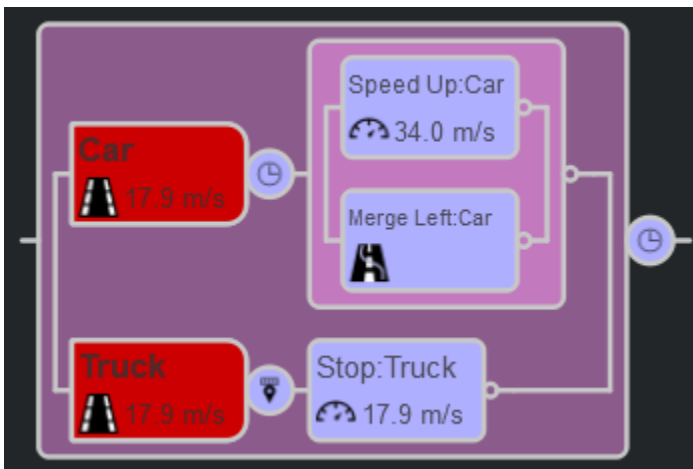
The subsequent action phases specify the actions that the vehicles perform during simulation.
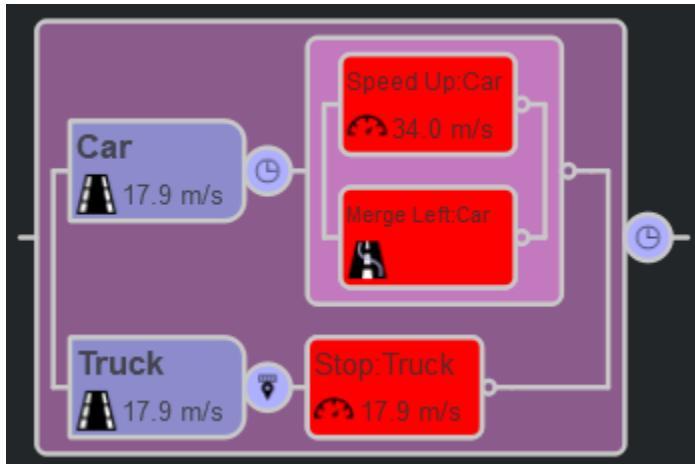
The circular nodes are conditions. These conditions trigger the next action phase to occur during simulation. For example, the car starts its next action phase after a certain amount of simulation time elapses. The truck starts its next action phase after it reaches a certain distance from a point.
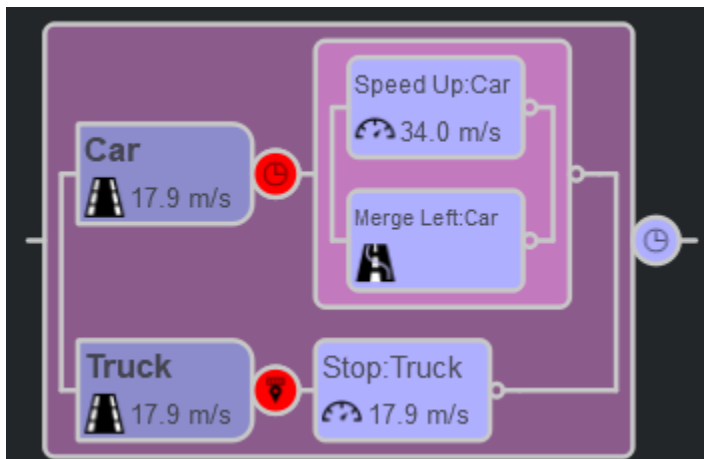
The scenario also includes an end condition. In this case, the scenario ends after a certain amount of simulation time elapses. Scenario end conditions also include a default fail condition in which the simulation ends as soon as any actor collides with another actor. If a scenario does not have an end condition, then the simulation continues indefinitely.
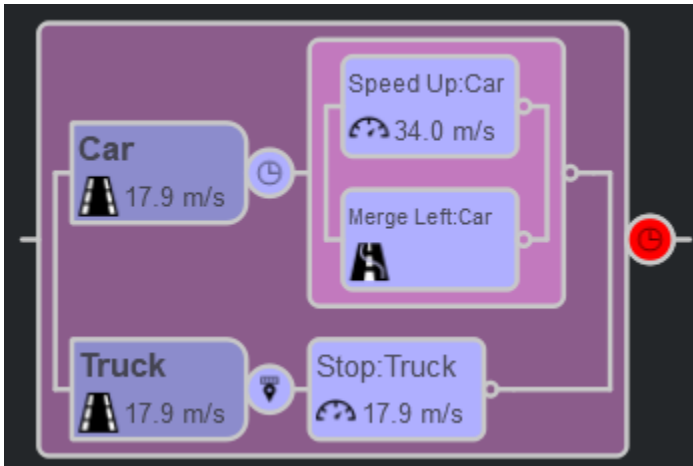
The rows containing the separate vehicle action phases and conditions are serial phases, which occur in order during simulation.
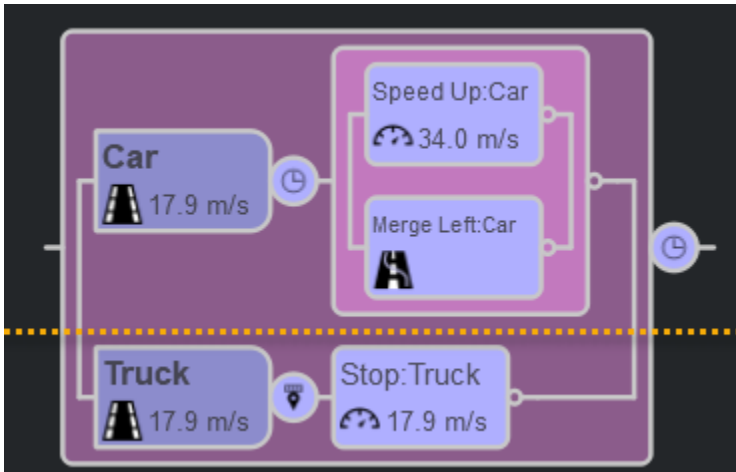
The action phases for the car in the interior box make up a parallel phase. These action phases occur in parallel during simulation.
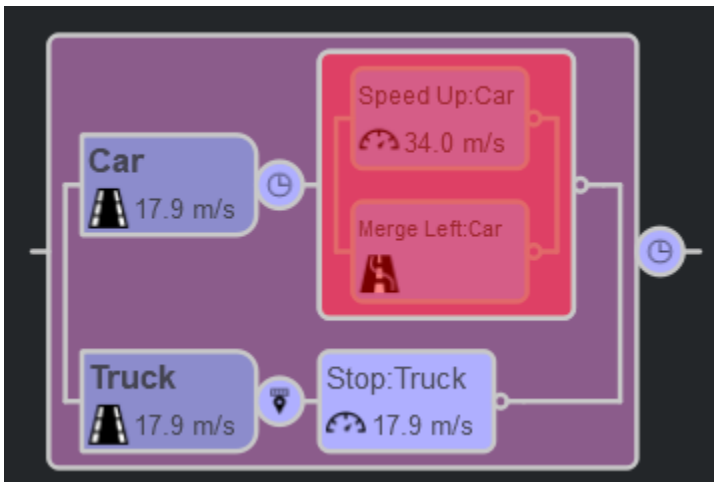
The scenario itself is also a parallel phase that consists of two serial phases: one for the truck and one for the car.
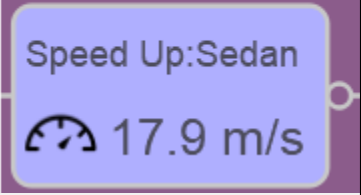
Initial Action Phases

When you drag a vehicle or other actor asset into a scenario, the **Logic** editor adds a new initial action phase for that actor. The initial action phase sets the state of the actor at the start of the simulation.
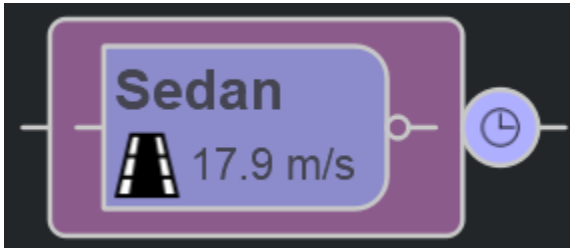
Initial action phases correspond to actors in the scenario. To add a new initial action phase, drag a new actor from the **Library Browser** into the scenario. All actors in a scenario must have a corresponding initial action phase. Deleting an initial action phase for an actor results in a simulation failure.

You can set and modify initial action phases from the **Attributes** pane. This table shows the attributes common to all action phases, including initial action phases.

Attribute	Description
Name	Name of the action phase. In the corresponding action phase box, this name is prepended to the actor name. Use these names to make the scenario logic more readable.
Actor	Actor that performs the action. To select an actor, first click the attribute box. Then, select an actor from the scenario editing canvas or the **Logic** editor. To frame the camera around the actor, click the Frame object in the scene button .

Initial Speed Action

The **Initial Speed** action sets the speed of the actor at the start of simulation. You must set this action, and actors can have only one **Initial Speed** action. This table shows the **Initial Speed** attributes you can set.

Attribute	Description
Motion	Motion behavior of the actor at the start of the simulation. This attribute is read-only, and is set based on whether you specify a path for the actor to follow. • `Follow Lane` — Actor follows its current lane. For a lane-following example, see "Design Lane Following Scenario" on page 3-2. • `Follow Path` — Actor follows a predefined path. For a path-following example, see "Design Path Following Scenario" on page 3-28.
Relative to	Method used to set the initial speed of the actor. The option you select changes what the **Speed** attribute represents. • `Absolute` — Actor starts the simulation at **Speed** m/s. • `Actor` — Actor starts the simulation at **Speed** m/s relative to **Reference Actor**, given the speed direction specified by **Direction**. **Default**: `Absolute`
Speed	Actor speed, in meters per second. Depending on the **Relative to** value you select, **Speed** is either the absolute speed of the actor at the start of simulation or a speed relative to another actor. If you specify an actor to move at the same speed as the reference actor (**Relative to** specified as `Actor`, and **Direction** to `Same as`), then the **Speed** attribute does not apply. **Default**: `17.88 m/s`

Attribute	Description
Reference Actor	Reference actor that the initial actor **Speed** value is relative to. To select an actor, first click the attribute box. Then, select an actor from the scenario editing canvas or the **Logic** editor. To frame the camera around the actor, click the Frame object in the scene button . This option applies only when you set **Relative to** to `Actor`.
Direction	Relative speed of the actor compared to the specified **Reference Actor**. • `Faster than` — Actor starts the simulation at **Speed** m/s faster than the reference actor. • `Slower than` — Actor starts the simulation at **Speed** m/s slower than the reference actor. • `Same speed as` — Actor starts the simulation at the same speed as the reference actor. This option applies only when you set **Relative to** to `Actor`.

Change Behavior Parameter Actions

`Change Behavior Parameter` actions set the initial values of parameters defined in the behavior file for an actor. Behavior files enable you to control the behavior of vehicle actors either from within RoadRunner Scenario or from external simulators, such as CARLA or Simulink. These behavior parameters are included in exports to ASAM OpenSCENARIO. For more details on behaviors, see "Specify and Assign Actor Behaviors" on page 3-138.

Consider a behavior file, `MyBehavior.rrbehavior`, that has an acceleration parameter with a default value of 30 m/s (**Name** = `Acceleration`, **Value** = `30`).
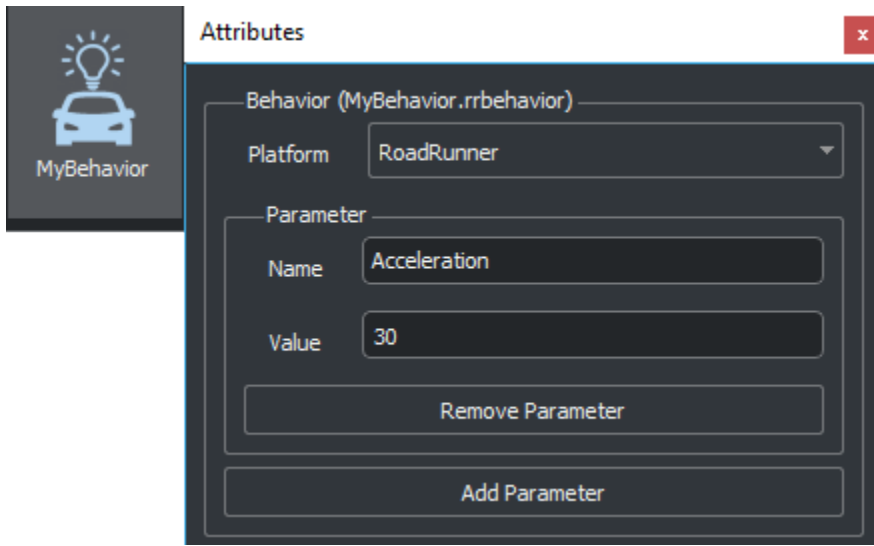
Suppose a vehicle in the scenario has its **Behavior** asset set to this behavior file.
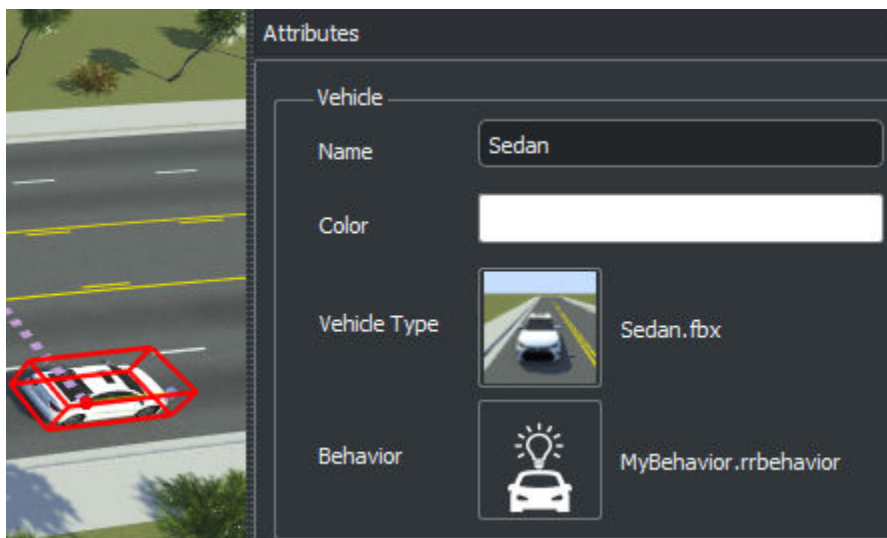
If you add a `Change Behavior Parameter` action to an action phase, you can select the `Acceleration` parameter under **Name** and then set this parameter to a new **Value**.
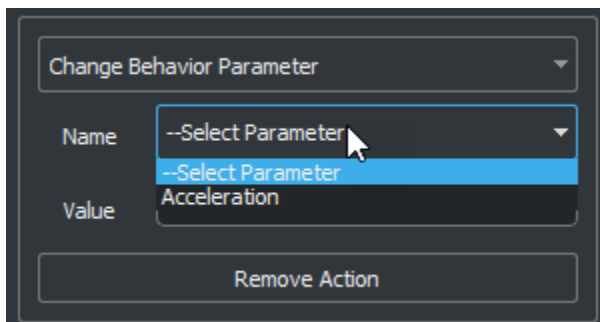
In initial action phases, you can add as many behavior parameters as are available in the behavior file. In subsequent action phases, you can set only one behavior parameter per phase. To change multiple behavior parameters, you must create parallel phases. The **Logic** editor displays the names of all behavior parameters being changed. This sample logic displays multiple behavior parameter changes on the initial phase and subsequent behavior parameter changes in parallel phases.
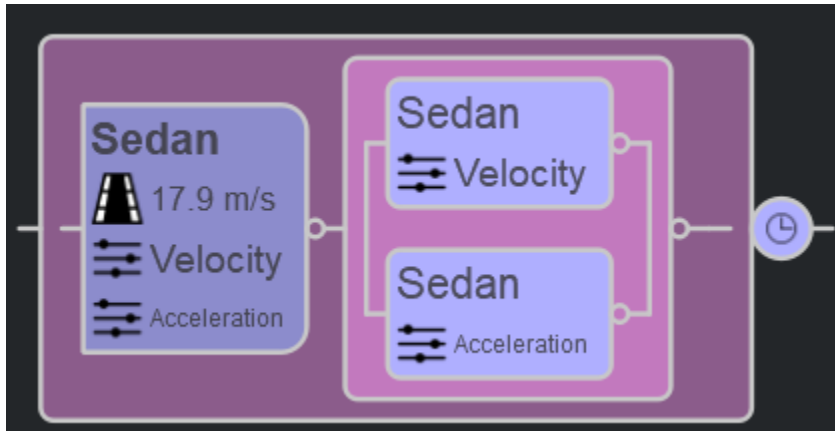
Change Global Parameters

Global parameters in RoadRunner Scenario adjust conditions for the entire scenario. Global parameters are flexible, and you can use them to represent different types of scenario parameters such as speed limit, time of day, and weather. To create a new global parameter, right-click in the right pane of the **Library Browser**, click **New**, then click **Parameter Collection**. A parameter collection is a group of parameters that can be used in multiple scenarios.
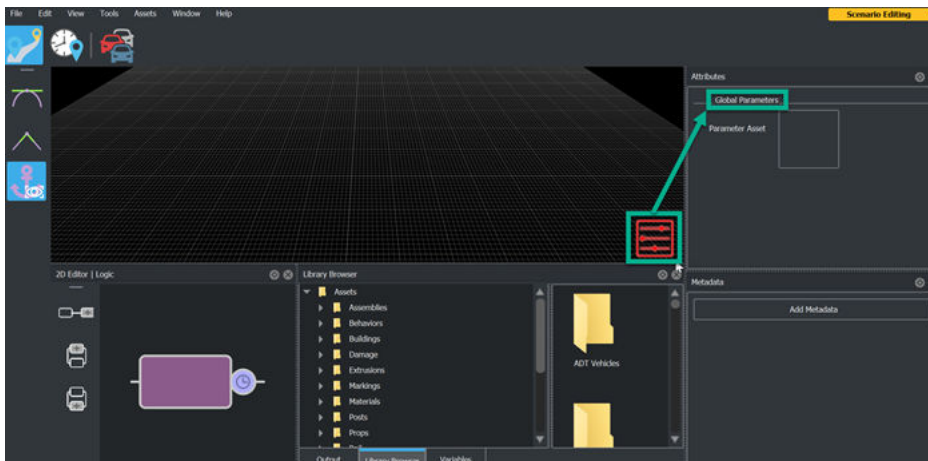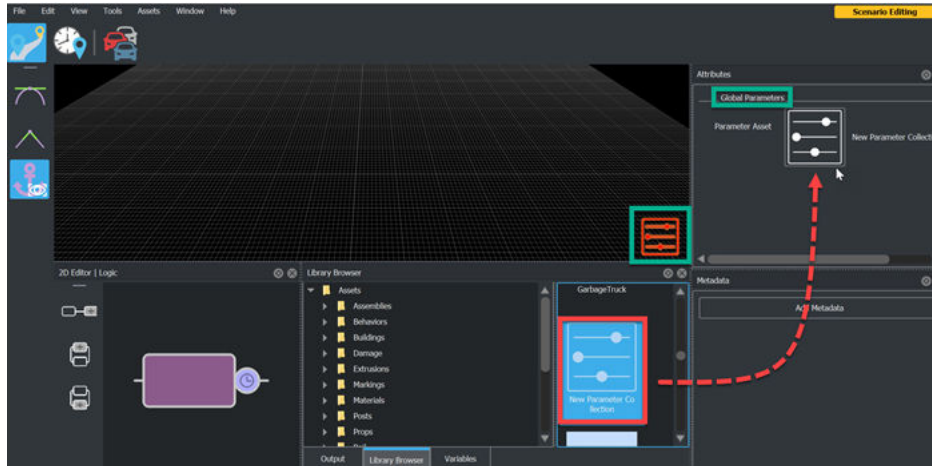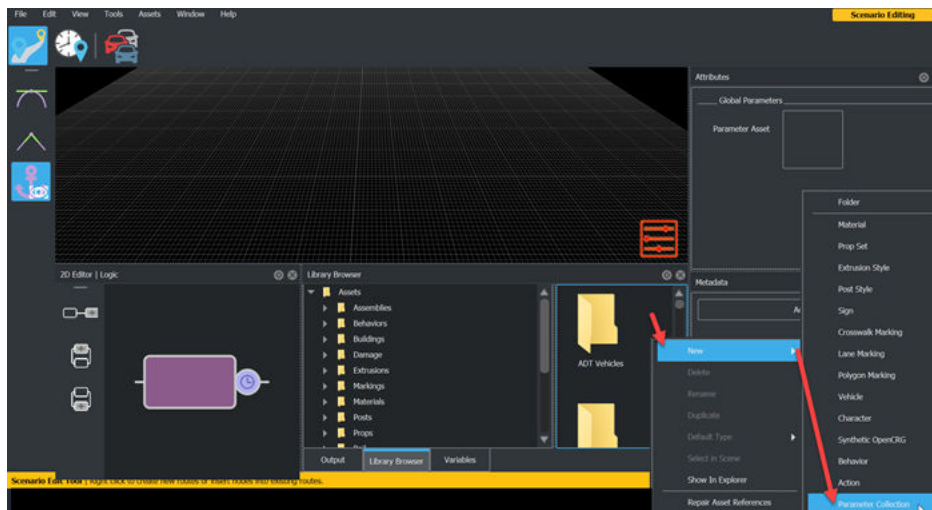
To assign a parameter collection to a scenario, drag the parameter collection asset from the **Library Browser** to the **Parameter Asset** field in the **Attributes** pane. You can set initial parameters by specifying their names in the corresponding **Name** fields and values in the corresponding **Value** fields.

To change global parameters during simulation, add an action in the **Logic Editor**, and, in the **Attributes** pane, set the **Action Type** to `Change Global Parameter`. To check the value of a global parameter during simulation, use a global parameter condition . To add a global parameter condition, create a condition in the **Logic Editor**, and, in the **Attributes** pane, set **Add Conditions** to `Global Parameter`. You can combine behavior and global parameters, as well as parameter actions and conditions, to design your scenario logic.

Parameter Data Types

RoadRunner Scenario supports these data types for parameters: `string`, `double`, `uint16`, `int32`, `uint32`, `boolean`, and `datetime`. In parameter collections, you can specify the data type of each parameter value.
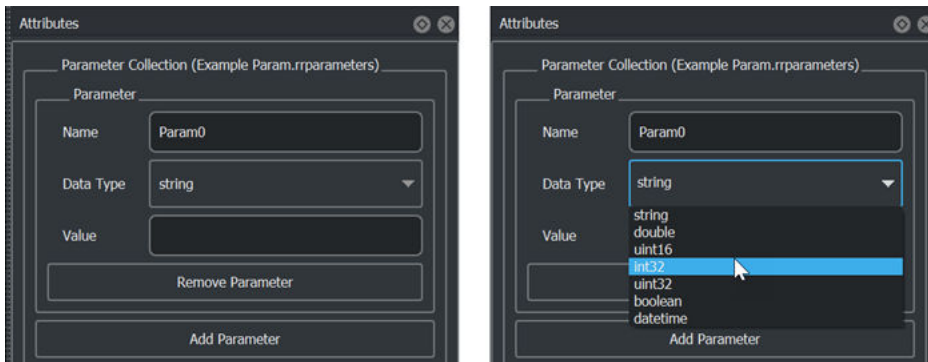
To define the data type for a parameter, select a parameter asset in the **Library Browser** and then, in the **Attributes** pane, select a data type from the **Data Type** list. You can then specify the value of this parameter in the **Value** field. Note that a parameter collection can contain parameters with various data types. Parameter assets imported from MATLAB use the `string` data type by default.

The names and data types of the parameters defined in a parameter collection asset are also displayed as elements of the **Name** list in the **Attributes** pane of conditions and actions.
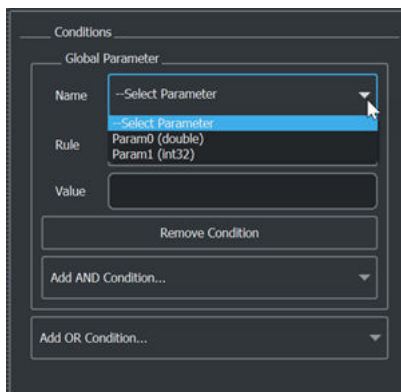
The **Value** field of a parameter must be a valid value for the selected data type. RoadRunner Scenario validates parameter values upon simulation or export, and if it cannot convert the value to the selected data type, then the validation fails and returns an error in the **Output** pane.
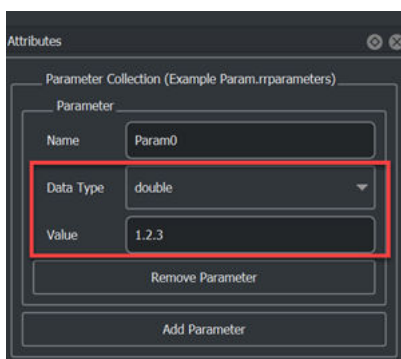
> ERROR: Scenario contains critical issues. Refer to the output panel for more details.
> WARNING: -------- Scenario Validation Report --------
> WARNING: Logic Issues:
> WARNING: CRITICAL ERROR: Failed to convert '1.2.3'. Navigate to parameter collection asset to resolve this error.
> WARNING: CRITICAL ERROR: Failed to convert '1.2.3'. Navigate to global parameters panel to resolve this error.
> WARNING: -------- End of Report --------

If you add a parameter to a parameter collection asset after the parameter collection has been added to the global parameters, RoadRunner Scenario displays an alert in the **Attributes** pane. Click **Update parameters from asset** in the global parameters **Attributes** pane to update the global parameters with the new parameter and clear the alert.
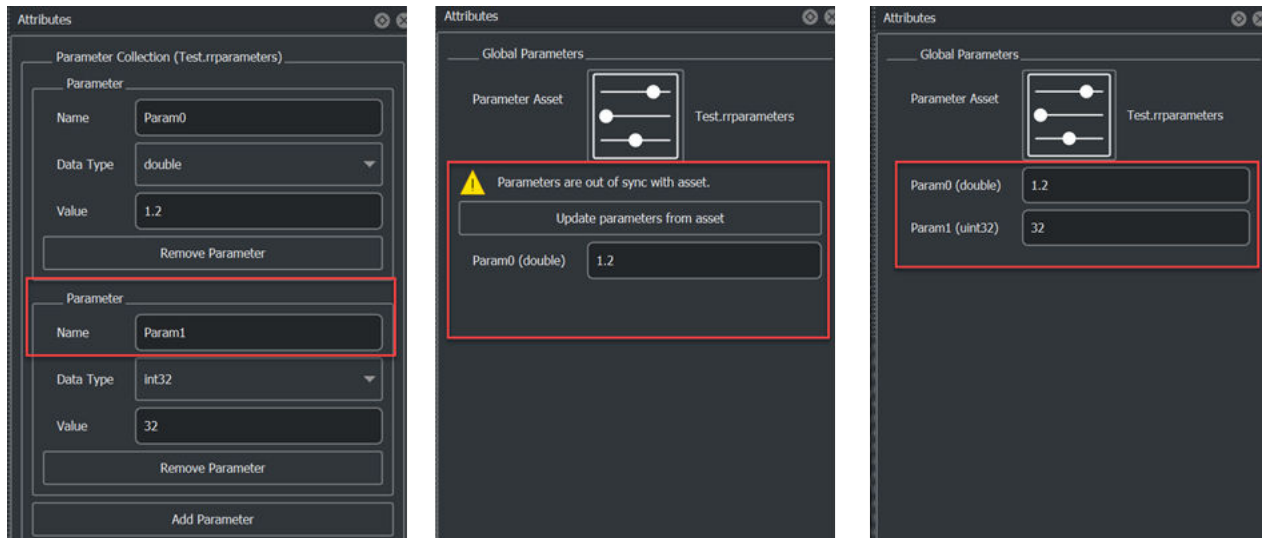
If you remove a parameter from a parameter collection asset after the parameter collection has been added to the global parameters, RoadRunner Scenario displays an error in the **Attributes** pane, and both simulation and export fail and return additional errors in the **Output** pane. Click **Update parameters from asset** in the global parameters **Attributes** pane to update global parameters to reflect the removed parameter. You must then update any actions and conditions that reference the removed parameter.
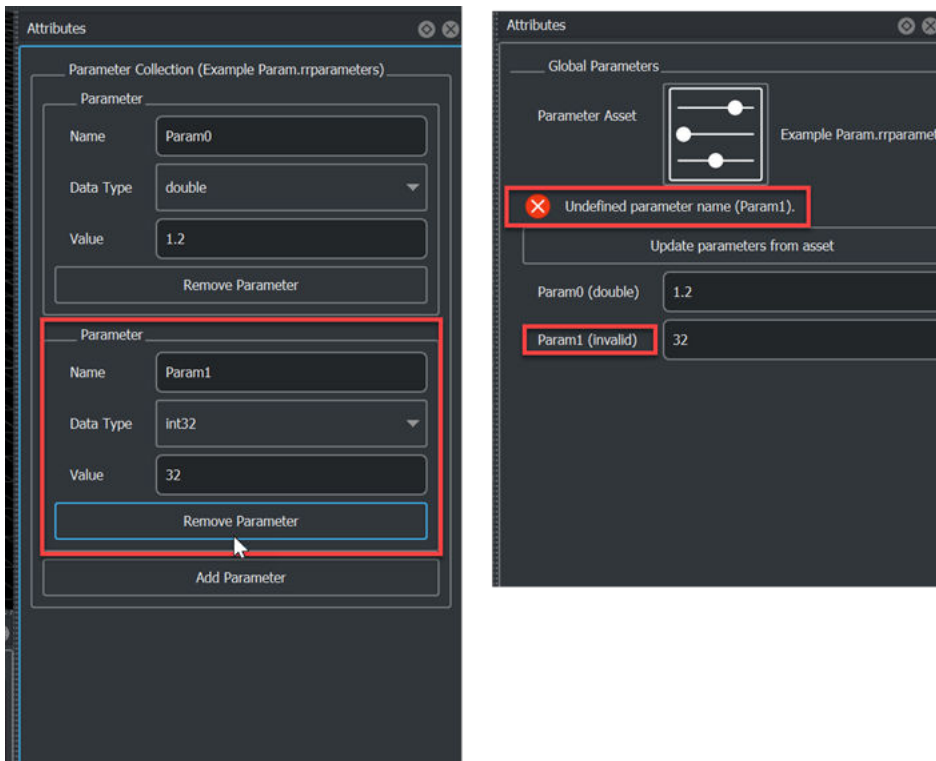
Action Phases

You can add action phases after an initial phase or after any other added action phase. To add an action phase after an existing one, right-click the action phase and select **Add Action Phase After**. In the **Logic** editor, the new phase appears selected and inherits its actor assignment form the previous action phase.

If the previous action phase does not have an assigned actor, or if you selected multiple phases with different actors, then the added phase does not include an actor assignment.

To delete an action phase, select that phase and press **Delete**.

As with initial action phases, subsequent action phases include **Name** and **Actor** attributes. The **Action Type** attribute contains the actions that you can set.

Change Speed Actions

The `Change Speed` action is similar to the `Initialize Speed` action. However, instead of starting the action phase at the specified speed, the actor must transition to that speed using specified actor dynamics.

In the **Attributes** pane for this action, use the `Change Speed` attributes to set the target speed that the actor transitions to after the action starts. This table shows the attributes that you can set.
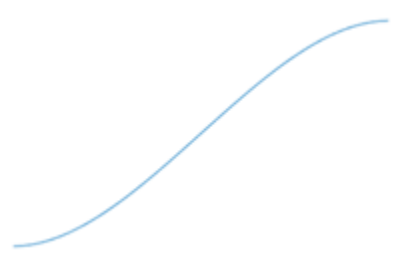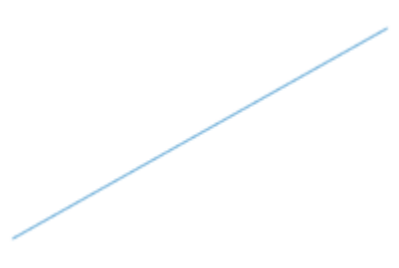
Attribute	Description
Relative to	Method used to set the speed of the actor. The option you select changes what the **Speed** attribute represents. • `Absolute` — Actor travels at **Speed** m/s. • `Actor` — Actor travels at **Speed** m/s relative to **Reference Actor**, given the speed direction specified by **Direction**. **Default**: `Absolute`
Speed	Actor speed, in meters per second. Depending on the **Relative to** value you select, **Speed** is either the absolute speed of the actor or a speed relative to another actor. If you specify an actor to move at the same speed as the reference actor (**Relative to** specified as `Actor`, and **Direction** to `Same as`), then the **Speed** attribute does not apply. **Default**: `17.88 m/s`
Reference Actor	Reference actor that the initial actor **Speed** value is relative to. To select an actor, first click the attribute box. Then, select an actor from the scenario editing canvas or the **Logic** editor. To frame the camera around the actor, click the Frame object in the scene button . This option applies only when you set **Relative to** to `Actor`.
Direction	Relative speed of actor compared to the specified **Reference Actor**. • `Faster than` — Actor ends the simulation at **Speed** m/s faster than the reference actor. • `Slower than` — Actor ends the simulation at **Speed** m/s slower than the reference actor. • `Same speed as` — Actor ends the simulation at the same speed as the reference actor. This option applies only when you set **Relative to** to `Actor`.

Attribute	Description
Speed Sampling	Sampling method that the actor uses to achieve a speed relative to the reference actor. • `At start of action` — Actor sets its relative speed based on the speed of the reference actor at the start of the action. • `Continuous` — Actor sets its relative speed based on the speed of the reference actor over the course of the action. This option applies only when you set **Relative to** to `Actor`.

Use the **Dynamics** attributes to set the dynamics that the actor uses to achieve the speed change. This table shows the attributes that you can set.

Attribute	Description
Dynamics Type	Dynamics that the actor uses to achieve the action, specified as one of these options: • `Over distance` — Actor achieves the action over the distance specified by **Distance**. • `Over time` — Actor achieves the action over the time specified by **Time**. • `With acceleration` — Actor achieves the action with the acceleration specified by **Acceleration**. **Default**: `Over distance`
Distance	Distance, in meters, over which the actor achieves the action. This attribute applies only when you set **Dynamics Type** to `Over distance`. **Default**: `10.00 m`
Time	Time, in seconds, over which the actor achieves the action. This attribute applies only when you set **Dynamics Type** to `Over time`. **Default**: `10.00 s`

Attribute	Description
Acceleration	Acceleration, in meters per second squared, that the actor uses to achieve the action. This attribute applies only when you set **Dynamics Type** to `With acceleration`. **Default**: `10.00 m/s2`

Attribute	Description
Dynamics Profile	Dynamics profile that the actor uses to achieve the action, specified as one of these options: • `Cubic` — Action follows a cubic polynomial over time or distance. • `Linear` — Action changes linearly over time or distance. • `Step` — Action changes step-wise over time or distance. This attribute applies only when you set **Dynamics Type** to `Over distance` or `Over time`. **Default**: `Cubic`

Change Lane Actions

Use a `Change Lane` action to change the lane of an actor using the specified dynamics.

For an example that uses lane change actions, see "Design Lane Change Scenario" on page 3-10.

If you set **Relative to** to `Current Lane`, then the target lane is relative to the lane that the actor is on. You can then set the **Direction** and **Lane Offset** attributes to specify the number of lanes, in either direction, that the target lane is offset from the current lane. This table shows a sample target lane specification.
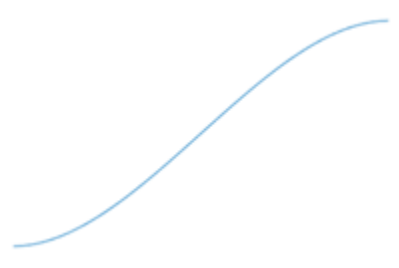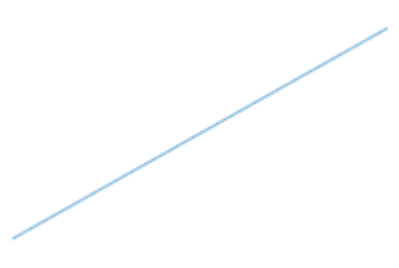
Target Lane Attributes	Description
Relative To — `Current Lane` **Direction** — `To the right` **Offset** — `2 lane(s)`	Target lane is two lanes to the right of the current lane of the actor.

If you set **Relative to** to `Actor`, then the target lane is relative to the current lane of the actor specified by **Reference Actor**. You can then set the **Direction** and **Lane Offset** attributes to specify the number of lanes, in either direction, that the target lane is offset from the lane of the reference actor. This table shows sample target lane specifications.

Target Lane Attributes	Description
Relative To — `Actor` **Reference Actor** — `Sedan2` **Direction** — `To the left` **Lane Offset** — `3 lane(s)`	Target lane is three lanes to the left of the lane that `Sedan2` is in.
Relative To — `Actor` **Reference Actor** — `Sedan2` **Direction** — `Same Lane`	Target lane is the same lane that `Sedan2` is in.

Use the **Dynamics** attributes to set the dynamics that the actor uses to achieve the lane change. This table shows the attributes that you can set.

Attribute	Description
Dynamics Type	Dynamics that the actor uses to achieve the action, specified as one of these options: • `Over distance` — Actor achieves the action over the distance specified by **Distance**. • `Over time` — Actor achieves the action over the time specified by **Time**. • `With acceleration` — Actor achieves the action with the acceleration specified by **Acceleration**. **Default**: `Over distance`
Distance	Distance, in meters, over which the actor achieves the action. This attribute applies only when you set **Dynamics Type** to `Over distance`. **Default**: `10.00 m`
Time	Time, in seconds, over which the actor achieves the action. This attribute applies only when you set **Dynamics Type** to `Over time`. **Default**: `10.00 s`
Acceleration	Acceleration, in meters per second squared, that the actor uses to achieve the action. This attribute applies only when you set **Dynamics Type** to `With acceleration`. **Default**: `10.00 m/s2`

Attribute	Description
Dynamics Profile	Dynamics profile that the actor uses to achieve the action, specified as one of these options: • `Cubic` — Action follows a cubic polynomial over time or distance. • `Linear` — Action changes linearly over time or distance. • `Step` — Action changes step-wise over time or distance. This attribute applies only when you set **Dynamics Type** to `Over distance` or `Over time`. **Default**: `Cubic`

Change Lateral Offset Actions
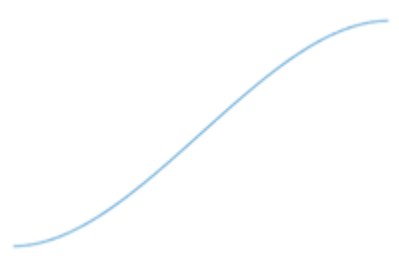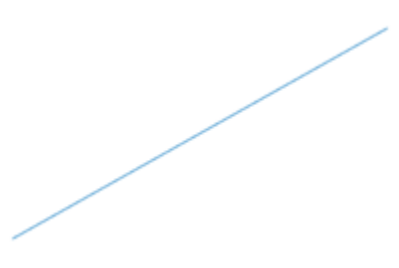
Use a `Change Lateral Offset` action to change how much an actor is offset from its lane center using the specified dynamics. This figure shows sample lateral offset actions applied to a red sedan.
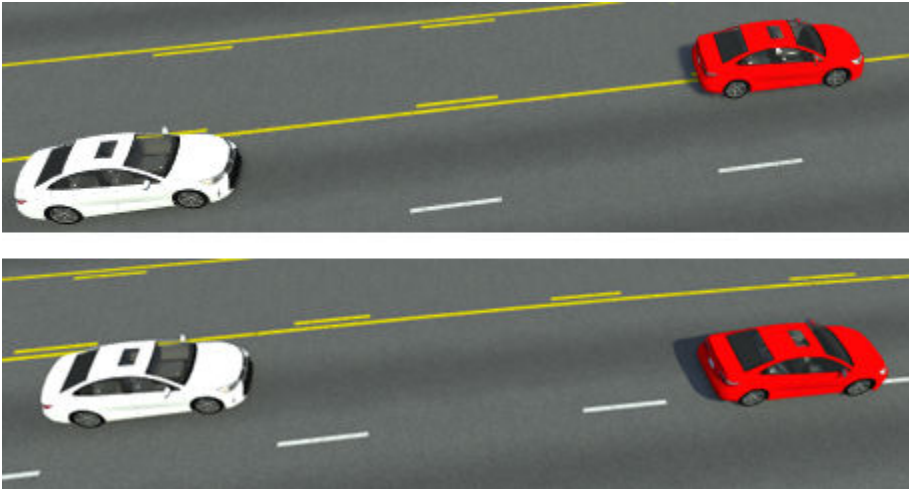
For an example that uses lateral offset actions, see "Design Lane Swerve Scenario" on page 3-20.

This table shows the attributes that you can set in the **Attributes** pane for this action.

Attribute	Description
Lateral Offset	Lateral offset, in meters, of the actor from the center of its lane. Lateral offset is positive to the right of the actor and negative to the left of the actor. This figure shows lateral offset values of `-1.5` and `1.5` meters, respectively. **Default**: `0.00 m`
Time	Time, in seconds, over which the actor achieves the action. **Default**: `5.00 s`

Attribute	Description
Dynamics Profile	Dynamics profile that the actor uses to achieve the action, specified as one of these options: • `Cubic` — Lateral offset change follows a cubic polynomial over time. • `Linear` — Lateral offset changes linearly over time. • `Step` — Lateral offset changes step-wise over time. This attribute applies only when you set **Dynamics Type** to `Over distance` or `Over time`. **Default**: `Cubic`

Change Longitudinal Distance Actions

Use a `Change Longitudinal Distance` action to change the longitudinal distance between an actor and the reference actor using the specified dynamics. You can use this action to maintain a constant longitudinal distance gap between two actors.

For a built-in vehicle, you can use the **Change Longitudinal Distance** action only if its reference actor is also in the same lane.

This figure shows a red sedan and white sedan traveling on the same lane while maintaining a constant longitudinal distance gap.

For an example that uses longitudinal distance actions, see "Design Overtake Using Longitudinal Distance Condition Scenario" on page 3-44.

This table shows the attributes that you can set.

Attribute	Description
Relative Position	Indicates the relative position of the actor with respect to the reference actor. • `Ahead of` — The actor is ahead of the reference actor. • `Behind` — The actor is behind the reference actor. **Default**: `Behind`
Reference Actor	Reference actor that the actor moves relative to. The reference actor is selected from the canvas.
Distance Type	Specification of longitudinal distance as either time distance or space distance. • `Space` — The longitudinal distance is measured in terms of the physical (space) distance between an actor and the reference actor. • `Time` — The longitudinal distance is measured in terms of the time interval between an actor and the reference actor. **Default**: `Space`
Space Distance Offset or **Time Distance Offset**	Target longitudinal distance gap between an actor and the reference actor. The name of the graphical control depends on the **Distance Type** that you select. • `Space Distance Offset` — Target longitudinal distance gap, measured in the space dimension. • `Time Offset Distance` — Target longitudinal distance gap, measured in the time dimension.

Attribute	Description
Measure From	Method of calculation of longitudinal distance between an actor and the reference actor. • `Bounding Boxes` — The longitudinal distance is measured between the edges of the bounding boxes of the two actors. • `Origins` — The longitudinal distance is measured between the origin points of the two actors.
Sampling Mode	The method by which the Longitudinal Distance action is applied. • `At start of action` — The actor achieves the target longitudinal distance gap to the reference actor during the **Change Longitudinal Distance** action phase. • `Continuous` — The actor can achieve the target longitudinal distance gap to the reference actor even outside the bounds of the **Change Longitudinal Distance** action phase. The action continues until another action controlling longitudinal distance is activated or the simulation ends.
Dynamic Constraints	This section describes any constraints that an actor must observe while executing the longitudinal distance action. These are the different constraint types: • `Asset Constraints` — The actor must follow the constraints included in its own list of attributes. To view the attributes of a vehicle, click on the vehicle thumbnail in the **Library Browser**, and check the **Attributes** pane on the right-hand side. • `Custom` — The actor must comply with the vehicle acceleration or deceleration values that you have specified in the **Max Acceleration**, **Max Deceleration** and **Max Speed** text boxes. • `Unlimited` — The actor can achieve the target longitudinal distance with no constraints.

User-Defined Actions

Use a `User Defined` action to dispatch custom parameters of a user-defined action from RoadRunner Scenario to a MATLAB System object or Simulink behavior model for further processing. After processing, the modified actor behavior is written back to the scenario.

Because the RoadRunner Scenario built-in actor behavior model ignores user-defined actions, a user-defined action must be sent to a cosimulation actor configured in MATLAB or Simulink. Use MATLAB functions, such as `getAction`, or Simulink blocks, such as RoadRunner Scenario Reader and RoadRunner Scenario Writer, to retrieve and process user-defined actions.

A user-defined action can represent vehicle characteristics, such as braking force or steering angle, that can be modeled using basic actor attributes like pose or velocity.

For an example that uses user-defined actions, see "Design Vehicle Following User-Defined Actions Scenario" on page 3-50.

This table shows the attributes that you can set.

Attribute	Description
Action Asset	Action asset file that contains custom parameter names and values. To create an action asset file: 1. Right-click in the space next to the asset folders list in the **Library Browser** and select **New > Action** from the context menu. 2. An action asset file with the `.rraction` file extension is automatically created. Enter a name for the action asset file.
Action Name	Action name. If you are mapping a user-defined action to a `Simulink.Bus` object for a Simulink actor behavior model, then you must use the action name entered here.
Parameter Name	Name of custom parameter within a user-defined action.
Parameter Value	Value of custom parameter.

Wait Actions

Unlike other actions, the `Wait` action is not tied to a specific actor. Use `Wait` actions when you want to pause for a set amount of time after the previous action is completed.

For example, suppose you want an actor to change lanes, wait five seconds after the completing the lane change, and then change speeds. Using a **Duration** condition to specify the delay does not work.

Duration conditions become active as soon as the previous action starts, not when it finishes. This means that the 5 second delay might end before the actor even completes its lane change.

To resolve this issue, use a `Wait` action with a 5 second **Duration** condition, instead.

Change Behavior Parameter

The `Change Behavior Parameter` action functions the same way as the similar actions uses in initial phases, as described in "Initial Action Phases" on page 3-73. The only difference is that, in non-initial action phases, you can specify only one behavior parameter change per action.

Conditions

Conditions specify the transitions between action phases. Each condition is associated with the action phase that immediately precedes it. The actor continues to perform this previous action until it meets the specified condition.

When you create a new scenario, the **Logic** editor adds a condition that triggers the end of the simulation.
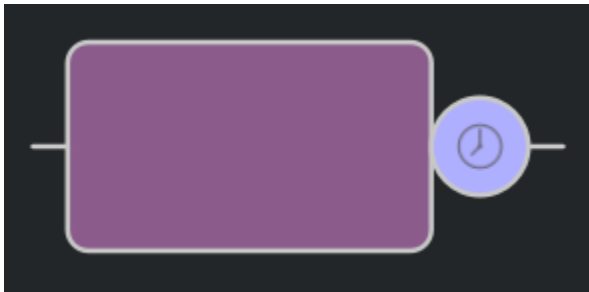
You can add additional conditions after any action phase (or parallel action phase) by following these steps:

1   In the **Logic** editor, click one of the open circular nodes.
2   In the **Attributes** pane, click **Add Condition**, and set the attributes for that condition.

Conditions are optional. If you do not specify a condition, then, after the previous action phase is complete, the next action begins immediately.

Simulation Time Condition

If you set **Condition** to `Simulation Time`, then the actor performs the previous action until the simulation time specified by **Time** elapses. Units are in seconds.
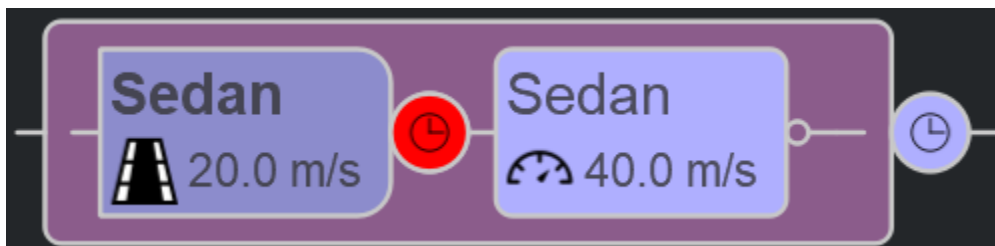
Actor Speed Condition

If you set **Condition** to `Actor speed`, then the actor performs the previous action until an actor reaches a specified speed. The specified speed can be relative to the speed of another actor.

This table shows the attributes that you can set.

Attribute	Description
Actor	Actor that you want to reach either the speed specified by **Speed** or the speed of the actor specified by **Reference Actor**. To select an actor, first click the attribute box. Then, select an actor from the scenario editing canvas or the **Logic** editor. To frame the camera around the actor, click the Frame object in the scene button .
Relative to	Method used to set the speed condition. The option you select changes what the **Speed** attribute represents. • `Absolute` — Condition is met when the actor reaches **Speed** m/s. • `Actor` — Condition is met when the actor is traveling **Speed** m/s relative to **Reference Actor**, given the speed direction specified by **Direction**. **Default**: `Absolute`

Attribute	Description
Speed	Speed value, in meters per second. Depending on the **Relative to** value you select, **Speed** is either an absolute speed or a speed relative to another actor. If you specify an actor to move at the same speed as the reference actor (**Relative to** specified as `Actor`, **Direction** to `Same as`), then the **Speed** attribute does not apply. **Default**: `17.88 m/s`
Reference Actor	Reference actor that the actor travels relative to. To select an actor, first click the attribute box. Then, select an actor from the scenario editing canvas or the **Logic** editor. To frame the camera around the actor, click the Frame object in the scene button ![icon]. This option applies only when you set **Relative to** to `Actor`.
Direction	Relative speed of the actor compared to the specified **Reference Actor**. • `Faster than` — Condition is met when the actor travels **Speed** m/s faster than the reference actor. • `Slower than` — Condition is met when the actor travels **Speed** m/s slower than the reference actor. • `Same speed as` — Condition is met when actor the travels at the same speed as the reference actor. This option applies only when you set **Relative to** to `Actor`.
Speed Sampling	Sampling method that the actor uses to achieve a speed relative to the reference actor. • `At start of action` — Actor sets its relative speed based on the speed of the reference actor at the start of the action. • `Continuous` — Actor sets its relative speed based on the speed of the reference actor over the course of the action. This option applies only when you set **Relative to** to `Actor`.
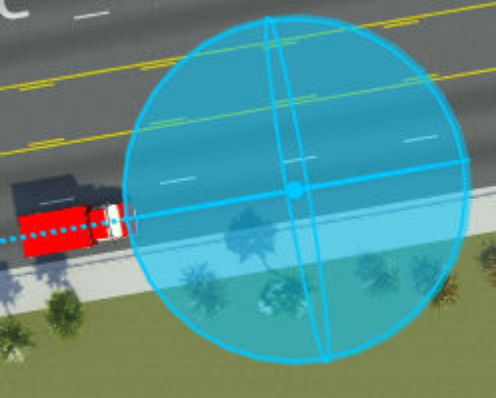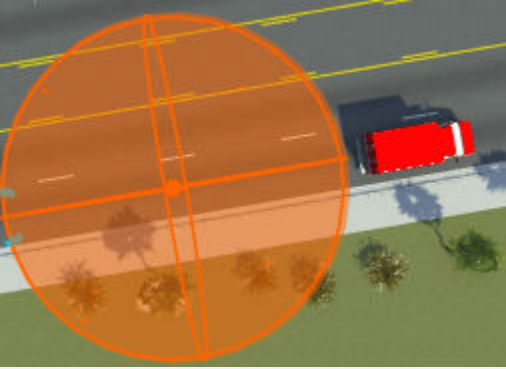
Distance to Point Condition

If you set **Condition** to `Distance to Point`, then the actor performs the previous action until an actor is a certain distance away from a specified point.

This table shows the attributes that you can set.

Attribute	Description
Actor	Actor that you want to reach the point specified by **Point**. To select an actor, first click the attribute box. Then, select an actor from the scenario editing canvas or the **Logic** editor. To frame the camera around the actor, click the Frame object in the scene button .
Point	Target path waypoint. To select a point, first click the attribute box. Then, select a point from the scenario editing canvas or the **Logic** editor. To frame the camera around the actor, click the Frame object in the scene button .
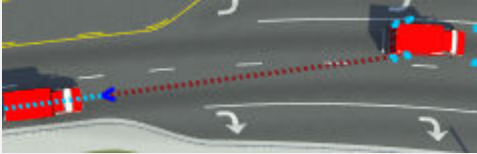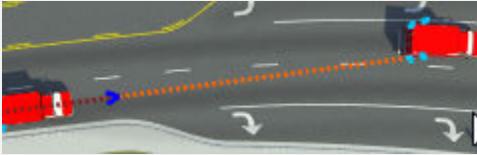
Attribute	Description
Rule	Rule used to determine the distance-to-point condition, specified as one of these options: • `Less than` — The actor meets the condition when it is less than **Threshold** meters from the point. • `Greater than` — The actor meets the condition when it is more than **Threshold** meters from the point. **Default**: `Less than`
Threshold	Radial distance around the point, in meters, at which the actor meets the condition specified by **Rule**. **Default**: `0.00 m`
Height Offset	Height of the point, in meters, above the ground. **Default**: `0.500 m`

Distance to Actor Condition

If you set **Condition** to `Distance to Actor`, then the actor performs the previous action until an actor is a certain distance away from another actor.

This table shows the attributes that you can set.

Attribute	Description
Actor	Actor that you want to reach the actor specified by **Reference Actor**. To select an actor, first click the attribute box. Then, select an actor from the scenario editing canvas or the **Logic** editor. To frame the camera around the actor, click the Frame object in the scene button .
Reference Actor	Reference actor that you want the actor specified by **Actor** to reach. To select an actor, first click the attribute box. Then, select an actor from the scenario editing canvas or the **Logic** editor. To frame the camera around the actor, click the Frame object in the scene button .
Rule	Rule used to determine the distance-to-actor condition, specified as one of these options: • `Less than` — The actor meets the condition when it is less than **Threshold** meters from the reference actor. • `Greater than` — The actor meets the condition when it is more than **Threshold** meters from the reference actor. **Default**: `Less than`

Attribute	Description
Threshold	Linear distance, in meters, from the reference actor at which the actor meets the condition specified by **Rule**. **Default**: `0.00 m`

Longitudinal Distance to Actor Condition

If you set **Condition** to `Longitudinal Distance to Actor`, then the actor performs the previous action until it is at a certain longitudinal distance from another actor.

For an example that uses longitudinal distance conditions, see "Design Overtake Using Longitudinal Distance Condition Scenario" on page 3-44.

This table shows the attributes that you can set.

Attribute	Description
Actor	Actor for which you want to apply the longitudinal distance condition.
Relative Position	Indicates the relative position of the actor with respect to the reference actor. • `Ahead of` — The actor is ahead of the reference actor. • `Behind` — The actor is behind the reference actor.
Reference Actor	Reference actor that the actor moves relative to. The reference actor is selected from the canvas.
Rule	Determines the condition that the longitudinal distance between **Actor** and **Reference Actor** must satisfy by a certain **Threshold** value. Specified as one of these options: • `Less than` — The actor meets the condition when its longitudinal distance to the reference actor is less than a certain threshold value. • `Greater than` — The actor meets the condition when its longitudinal distance to the reference actor is greater than a certain threshold value.

Attribute	Description
Threshold	Longitudinal distance, in meters, from the reference actor at which the actor meets the condition specified by **Rule**. For example, consider that **Relative Position** is set to `Behind`, **Rule** is set to `Greater than` and **Threshold** is set to `5.00 m`. This longitudinal distance condition is satisfied when the actor attains a distance greater than 5.00 metres behind the reference actor. **Default**: `0.00 m`
Measure From	Measure of the longitudinal distance between the **Actor** and **Reference Actor** with respect to the vehicle boundaries. `Bounding boxes` — The longitudinal distance is measured between the bounding boxes of the two actors. `Origin` — The longitudinal distance is measured between the origin points of the two actors.
Measure Distance	Method of calculation of longitudinal distance between the **Actor** and the **Reference Actor**. • `Along lane curvature` - The longitudinal distance is measured as the distance between the two actors, projected onto the curvature of the lane in which they are running. • `Longitudinal distance only` - The longitudinal distance is measured as the distance between the two actors, projected onto the heading direction.

Collision Condition

If you set **Condition** to `Collision`, then the actor performs the previous action until the actor specified by **First Actor** collides with the actor specified by **Second Actor**.

By default, **First Actor** and **Second Actor** are set to any actor, meaning that the scenario ends upon the first collision between any two actors.

Duration Condition

If you set **Condition** to `Duration`, then the actor performs the previous action for the amount of time specified by **Duration**. Units are in seconds.
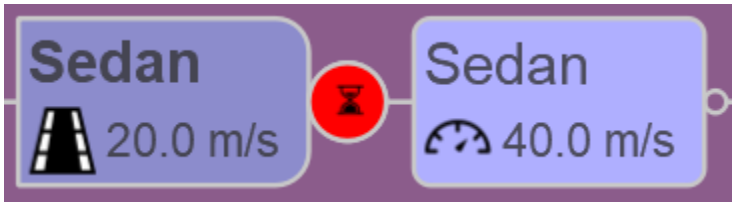
AND Condition

If you set **Condition** to AND, then the actor transitions to the next action only when all the preceding conditions joined by an AND condition have been completed. If any of the conditions joined by an AND condition do not complete, then the actor does not transition to the next action.
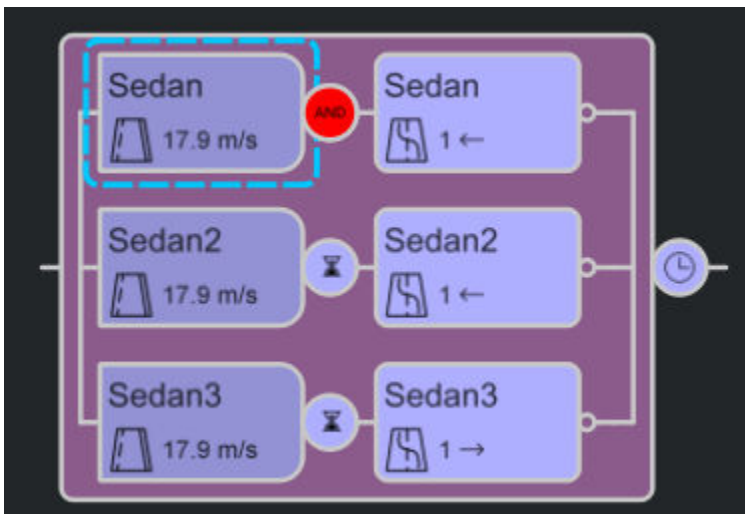
Multiple Conditions

In the **Attributes** pane for a condition, by clicking **Add Condition**, you can add multiple conditions. The condition node changes to reflect that multiple conditions are set.
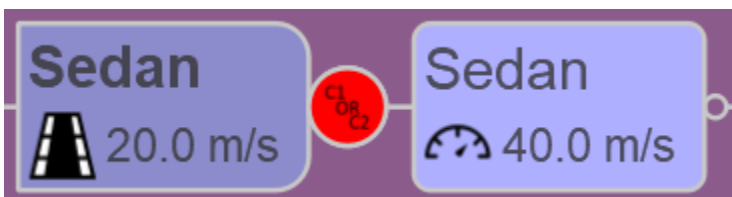
When the actor meets any one of the conditions specified, then the entire condition is considered met and the scenario transitions to the next action phase.

Fail Conditions

The condition node at the end of the scenario contains an extra **Fail Conditions** section in the **Attributes** pane.
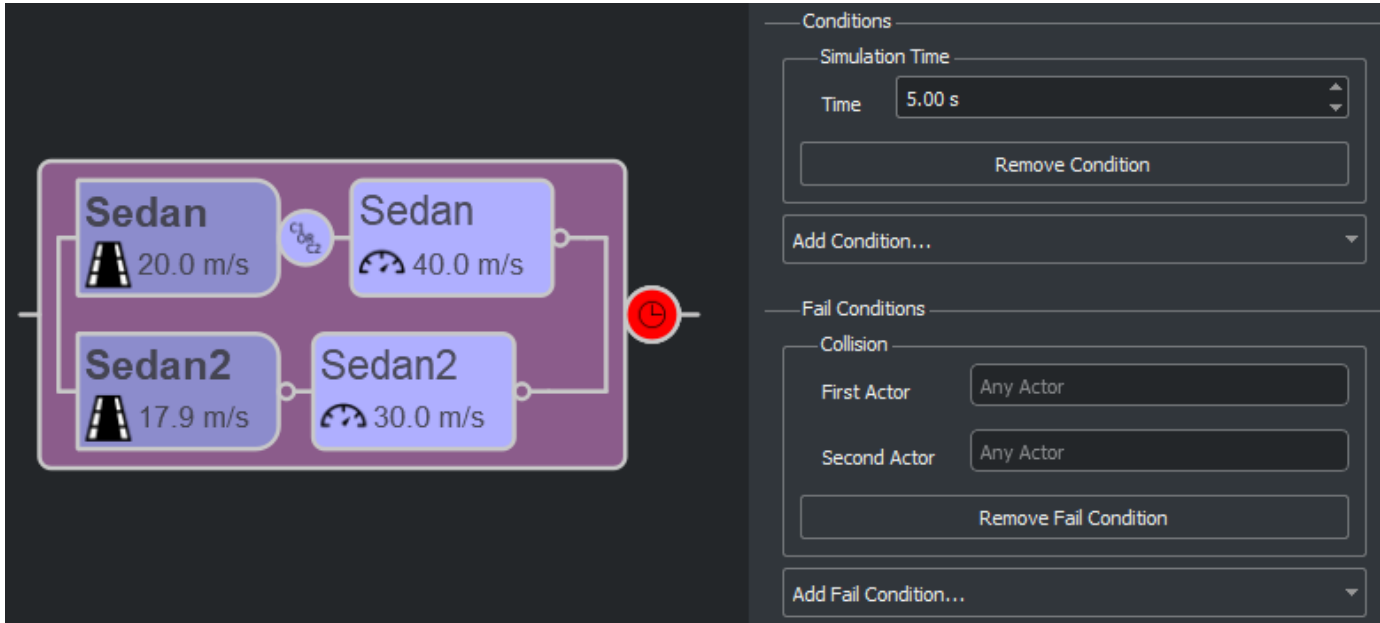
Use fail conditions to end the scenario as soon as one of the specified conditions is met. You can specify any of the previously described conditions. The default fail condition is when any actor collides with any other actor.

Phase State Conditions

Check the runtime state of a phase in the **Logic Editor** by using phase state conditions. To model a phase state condition, create a condition in the **Logic Editor** and, in the **Attributes** pane, under **Conditions**, select `Phase State`. The phase state condition checks the status of a phase during simulation to determine whether its state is `running` or `complete`.
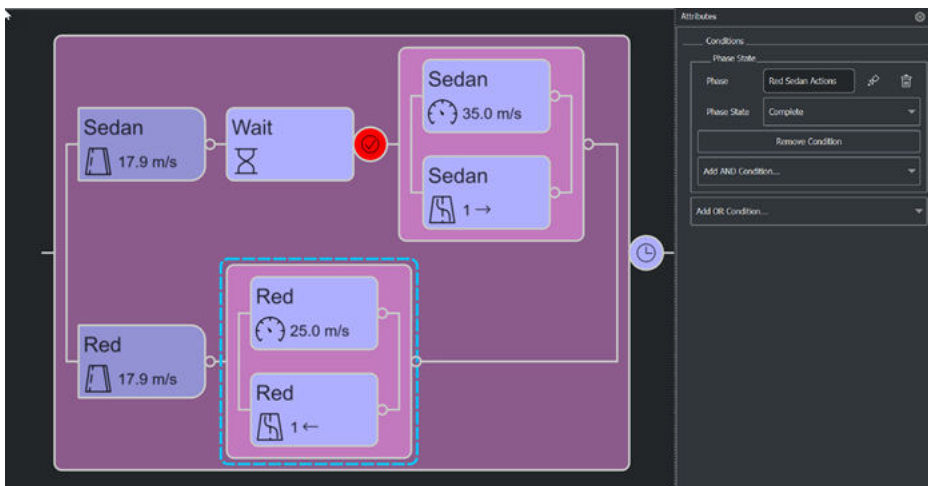
Phase state conditions enable more complex synchronization between actor actions, beyond serial and parallel execution.

In this example, the white sedan has parallel `Change Lane` and `Change Speed` actions after a `Wait` action with a phase state condition linked to the `Change Lane` and `Change Speed` actions of the red sedan. As a result, the white sedan waits for the red sedan to finish its lane transition, then changes lanes and accelerates once the lane transition is complete.
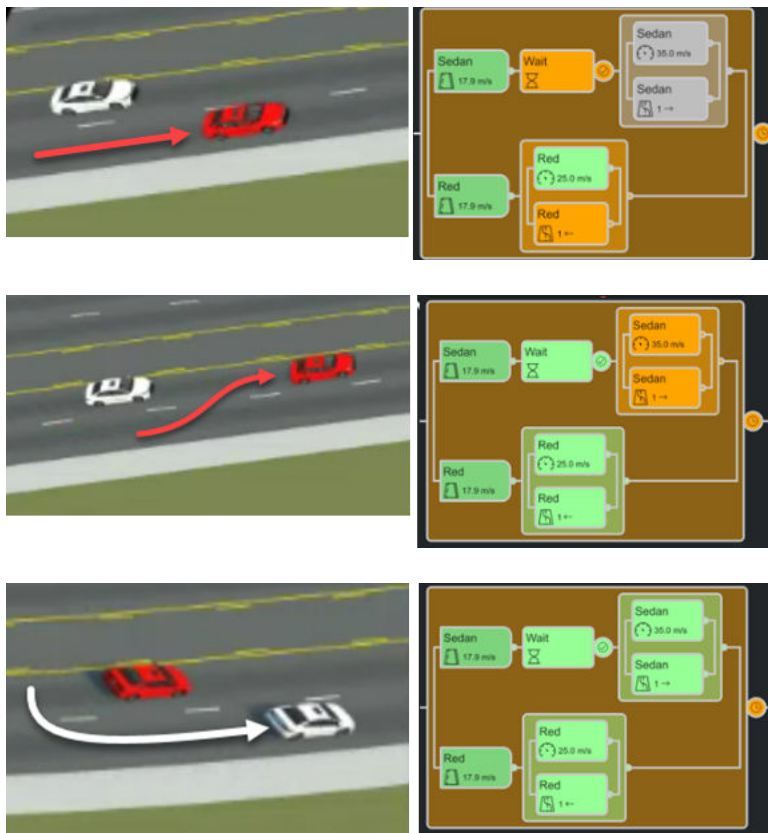
Serial Phases

Serial phases are collections of action phases and optional conditions that happen consecutively during simulation. Typically, the actions of a specific actor are all part of one serial phase. When you right-click an action phase and select **Add Action Phase After**, you add a new action phase that occurs after the clicked phase. By default, the created action phase inherits the attributes of the previous phase.

Parallel Phases

Parallel phases are collections of phases that happen at the same time during simulation. Serial phases that correspond to each vehicle in a scenario collectively make up a parallel phase, where the actions of the actors happen in parallel. You can create parallel phases for any action phase except initial action phases.

To create a parallel action phase, right-click an existing action phase and select **Add Action Phase Above** or **Add Action Phase Below**. The new action phase appears above or below the existing one. The phase is initialized with the actor of the existing phase but has no actions specified.
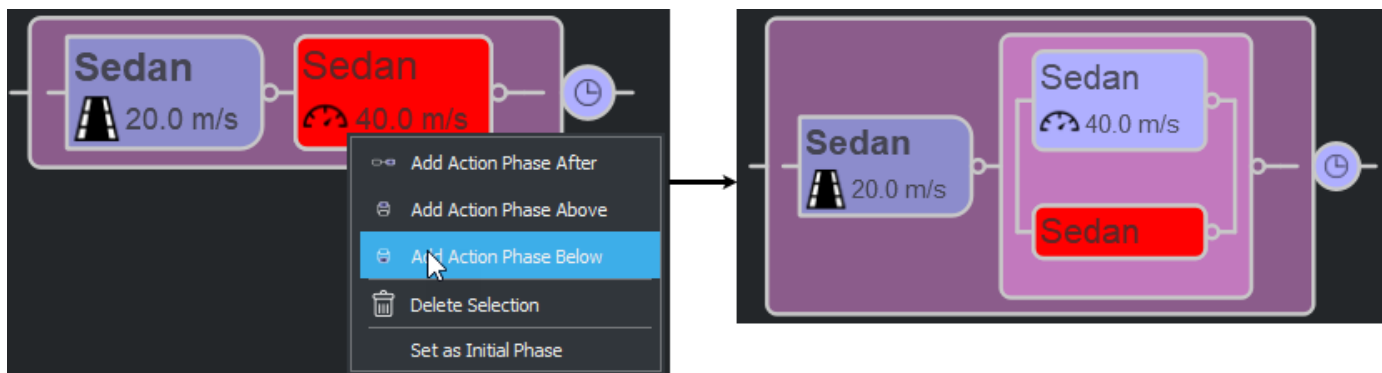
Use these parallel phases to organize your scenario logic. For example, suppose your scenario contains a chain of speed changes and lane changes. You can specify the speed change actions in one serial phase and the lane changes in another.

Logic Editor During Simulation

During simulation, the action phases and conditions change color to indicate the progression of the scenario. The colors have these meanings:

- Green — The action phase or condition is complete.
- Orange — The action phase or condition is in progress.
- Gray — The action phase or condition has not yet been reached.
- Gray and Red — The action phase or condition has been skipped.
- Green and Red — The action phase or condition has been interrupted.

If an action is interrupted, the vehicle actor invokes this default control behavior:

- If a longitudinal action is interrupted, the actor retains its current speed.
- If a lateral action is interrupted, the actor corrects its yaw to maintain the lateral offset from its last source.

Actions that are set to run continuously, and do not have a specified end condition, remain orange in the **Logic** editor and continue indefinitely.

These figures show samples of the **Logic** editor pane during simulation.
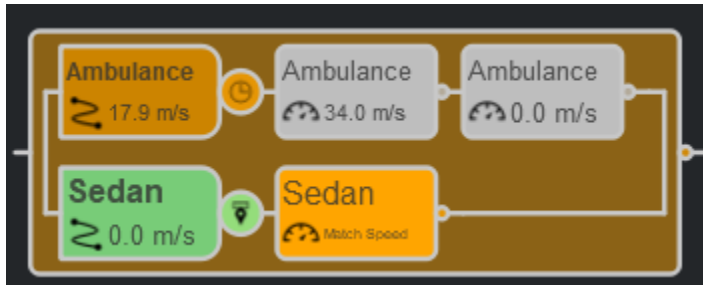
Logic Editor Limitations

- After the initial phase, you can set only one action per phase. To set multiple, simultaneous action phases, define the actions in parallel phases.
- `Change Lane` actions are not supported for actors containing predefined paths. During simulation, the actors follow the paths and do not perform the lane change actions.
- Setting `Change Lane` and `Change Lateral Offset` actions in the same serial phase is not supported.
- The `Actor Speed` condition is not supported for export to ASAM OpenSCENARIO 1.0.
- The `Change Lane` and `Change Lateral Offset` actions might not work as expected on lane loops. A lane loop is a sequence of connected lanes along multiple roads that loop back to the road that the vehicle is on. In some lane loops, the current lane of the vehicle and its adjacent lane are considered part of the same lane sequence, as shown in this figures.

The `Change Lane` and `Change Lateral Offset` actions determine their behavior based on the current lane of the vehicle and its adjacent lanes. For example, suppose a vehicle performs an action relative to another lane, such as a lane change to the left. If RoadRunner Scenario detects that the current lane of the vehicle and the lane to the left are part of the same lane sequence, then the action ends immediately. The lane change does not occur because RoadRunner Scenario considers both lanes to be a single lane.

By default, RoadRunner Scenario detects lane loops within 200 meters in front of or behind the vehicle along its lane. To reduce this distance so that current and adjacent lanes in a loop are considered separate lanes, in the `SimulationConfiguration.xml` file, decrease the `MaxSearchDistance` parameter. For more details, see Simulation Configuration.

See Also

Related Examples

- "Explore and Simulate a Simple Scenario" on page 1-13
- "Design Lane Following Scenario" on page 3-2
- "Design Lane Change Scenario" on page 3-10
- "Design Lane Swerve Scenario" on page 3-20
- "Design Path Following Scenario" on page 3-28

Scenario Anchoring System

RoadRunner Scenario uses an anchoring system in which you can specify the positions of objects as being relative to points called anchors. In a scenario, an anchor can be:

- A point on a road or junction
- A point specifying the location of an actor
- A path waypoint

You can anchor to any location-based scenario object, such as vehicles or path waypoints. When you move an anchor, the objects anchored to it move with it. This feature enables you to move an entire scenario to a new location by dragging a single point. Also, you can relocate anchors and the attached scenarios within the scene and between scenes. For more information on anchor and scenario relocation, see "Relocate Scenarios" on page 3-117.

Consider a scenario in which four vehicles are either directly or indirectly children of a road anchor. The orange arrows in this figure indicate the anchor parentage in the scenario. The road anchor is in lavender.
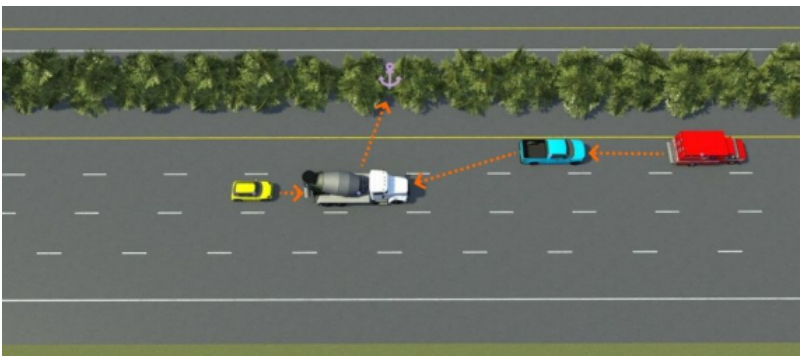
If you move the white cement truck backward, the other three vehicles anchored either directly or indirectly to it also move backward. The road anchor remains fixed.
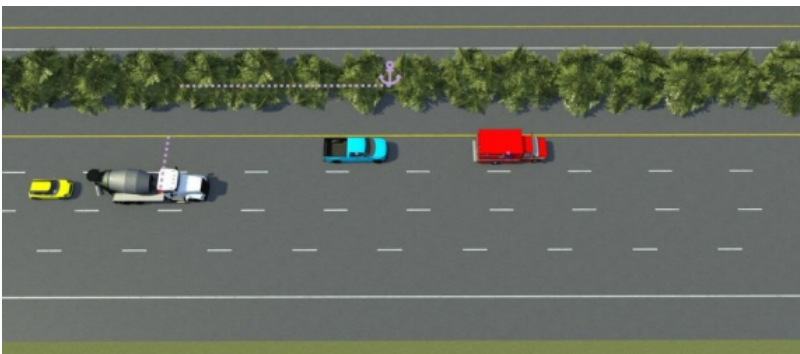
If you move the blue pickup truck forward, only the red ambulance moves with it, because the ambulance is the only other vehicle for which the pickup truck is an anchor parent.

If you move the yellow car, no other vehicles move with it because no other objects have it as an anchor parent. The same case applies for the red ambulance.

If you move the road anchor, all four vehicles move with it, enabling you to move the scenario to an entirely new location in a scene.
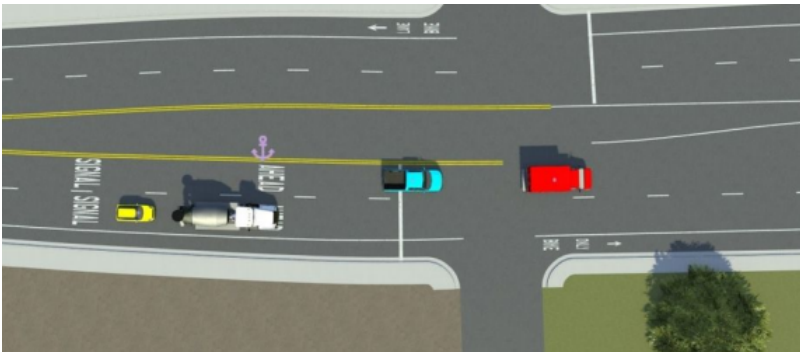
If you the change the shape of a road, the vehicles adapt to the changes while maintaining their relative positioning constraints.
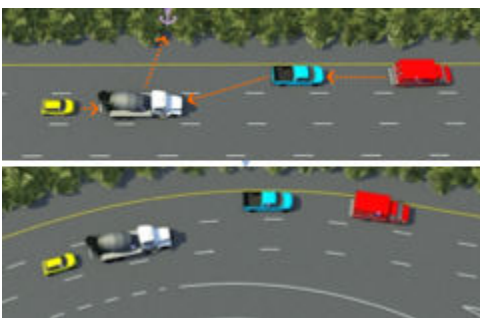
Anchor Object to Road

When you first add a scenario object, such as a vehicle, on or near a road, the scenario adds an anchor that is in the center of the road and parallel with the object.
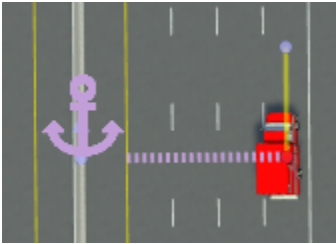
By default, each object you add to a scenario has anchoring enabled. In the **Attributes** pane for that actor, under **Point Offsets**, the **Enable Anchoring** attribute is selected. This behavior applies to vehicles and other actors, as well as to path waypoints.

If you add additional objects to the same road, these objects also attach to this road anchor. If you drag an object to a new road, the parent anchor of the object switches to the road anchor of the new road. To prevent an object from changing anchors, in the **Attributes** pane, select **Lock To Anchor**.

Move Objects Relative to Anchor

Each anchoring-enabled object has a **Point Offsets** section in its **Attributes** pane that indicates its forward offset and lane offset from the anchor.

When you drag an object along a lane, then, in the **Attributes** pane, under **Forward Offsets**, the **Offset** value of the object from its parent anchor changes. Units are in meters. This table shows sample changes in forward offsets.

Positive Forward Offset	Negative Forward Offset

When you drag an object into a different lane, then, in the **Attributes** pane, under **Lane Offset**, the offset of the object from the reference lane changes. This table shows the **Lane Offset** attributes that you can set.

Attribute	Description
Relative To	Specify the location that the anchor is relative to. The only option available is **Road Edge**.
Offset From	Specify whether the lane offset of the object is measured from the leftmost or rightmost lane in the travel direction of that object by selecting either **Left Lane** or **Right Lane**.

Attribute	Description
Lane Offset	Select the number of lanes by which the object is offset from the lane.
Travel Direction	Select whether the object is offset with the travel direction of the anchor or against its travel direction.
Lateral Offset	Specify how many meters the object is offset from the center of its lane. **Lateral Offset** is negative to the left and positive to the right for offset from lane center

For example, this vehicle is offset two lanes from its left-lane road edge. The vehicle is following the travel direction of the anchor, and no lateral offset is applied.
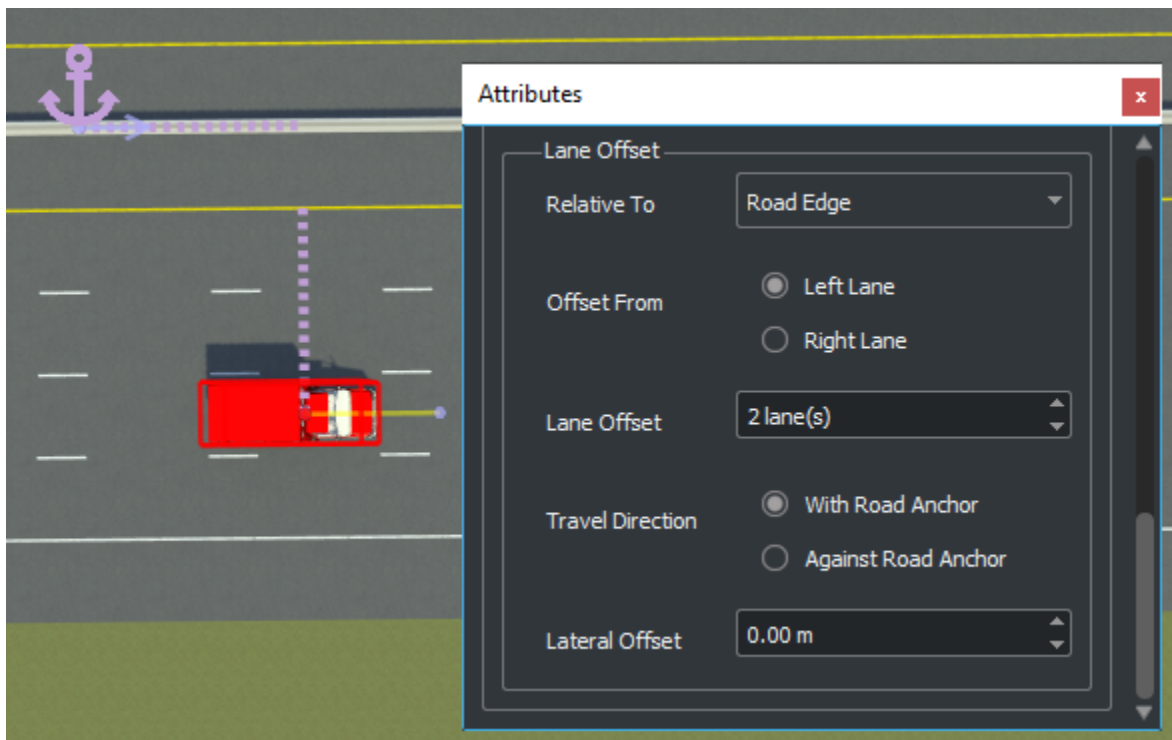
When relocating scenarios within a scene, actors maintain their lane-relative positions. For example, suppose a car is in the rightmost lane (zero lanes from the road edge) and a turning lane forms in the middle of the road. If you move the anchor along the road, the car moves into the turning lane to maintain a relative position of zero from the road edge.

To address this, set predefined paths for cars. For an example, see the "Design Path Following Scenario" on page 3-28 example.

Manually Add Road Anchors

RoadRunner Scenario automatically creates a new anchor any time you add a vehicle to a scenario.

You can manually add new road anchors to a scene by selecting the **Road Anchor Tool** ⚓. Manually adding anchors enables you to add multiple road anchors to a single road, which can be useful on long roads.

To manually add an anchor to a road, follow these steps:

1 Switch to scene editing mode. From the top-right corner of the RoadRunner application, select **Scenario Editing**, then **Scene Editing**.
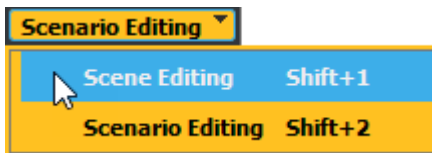
2
From the RoadRunner toolbar, click the **Road Anchor Tool** ⚓.

3 Click to select the road to which you want to apply an anchor.
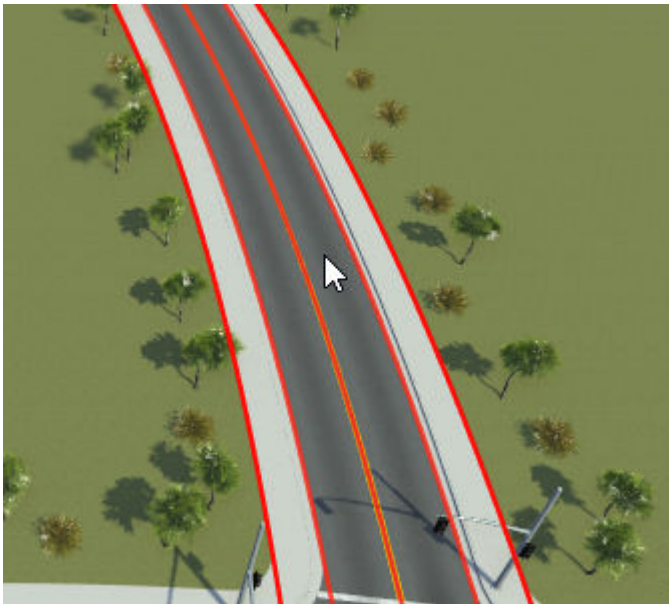
4 Right-click the road to apply the anchor to the center of the road.
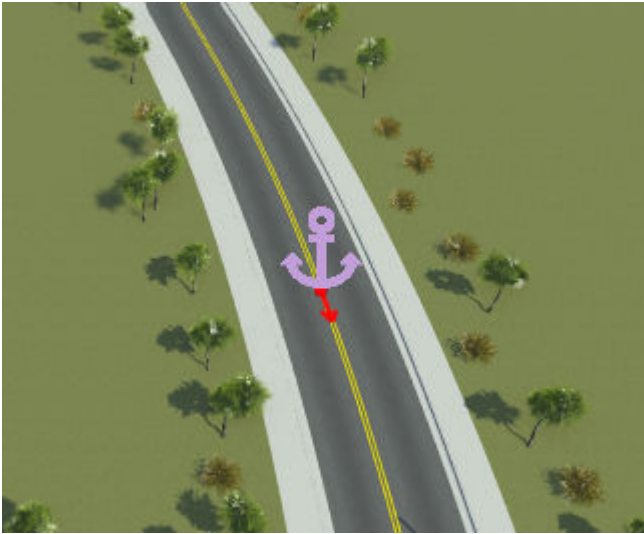
5   Switch back to scenario editing mode. From the top-right corner of the RoadRunner application, select **Scene Editing**, then **Scenario Editing**.
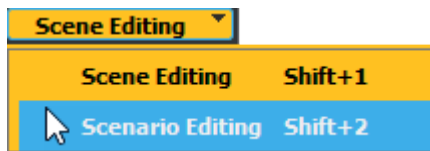
You can now attach objects to this anchor as you would any other road anchor.

Modify Anchor Attributes

To modify anchor attributes, select the anchor, or the actor or path waypoint that is an anchor to another object, and make your changes in the **Attributes** pane. The anchor attributes of scenario objects, such as vehicles and path waypoints, are saved to the scenario. When you change these attributes, they affect only the other objects in the scenario. The attributes of road anchors are saved to the scene. If other scenario files use this scene and reference the anchor you change, then those files can also be affected.

Note When choosing anchor names, choose something easy to remember so that the scenarios are easier to relocate later.

For example, suppose you change the **Name** attribute of a road anchor to `ScenarioStart`.
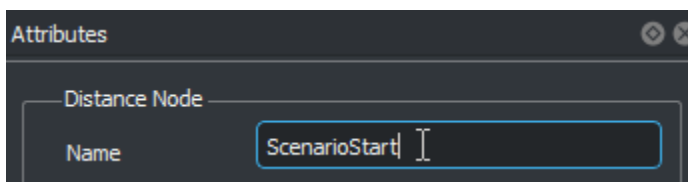
For actors within the current scenario, the name of their parent anchor updates automatically to reflect the new name. However, in other scenario files that use this scene, the parent anchor name

does not change. Instead, objects previously attached to this anchor become unanchored, and you must manually change their actor parent.

Change Anchor Parent

To change the anchor parent of an object, follow these steps:

1 Click the object, such as a vehicle, to select it.

2 In the **Attributes** pane, click the name of the anchor that is defined in the **Anchor** attribute. RoadRunner Scenario highlights the scenario editing canvas and **Logic** editor with blue lines to indicating the areas from which you can select a new parent anchor.

3 In the areas highlighted by the blue lines, select a new parent anchor. If you select an action phase from the **Logic** editor, then RoadRunner Scenario updates the anchor to the actor that is associated with that action phase.

To find the location of the current parent anchor in the scenario, click the frame button next to

Anchor attribute value .

Change Travel Direction of Actors

If you select a road anchor, then, in the **Attributes** pane, you can change its **Travel Direction** setting to change the travel direction of all objects either directly or indirectly attached to that anchor. The **Forward** and **Backward** travel direction options are based on the direction in which the road was created.

Travel Direction Set to Forward	Travel Direction Set to Backward
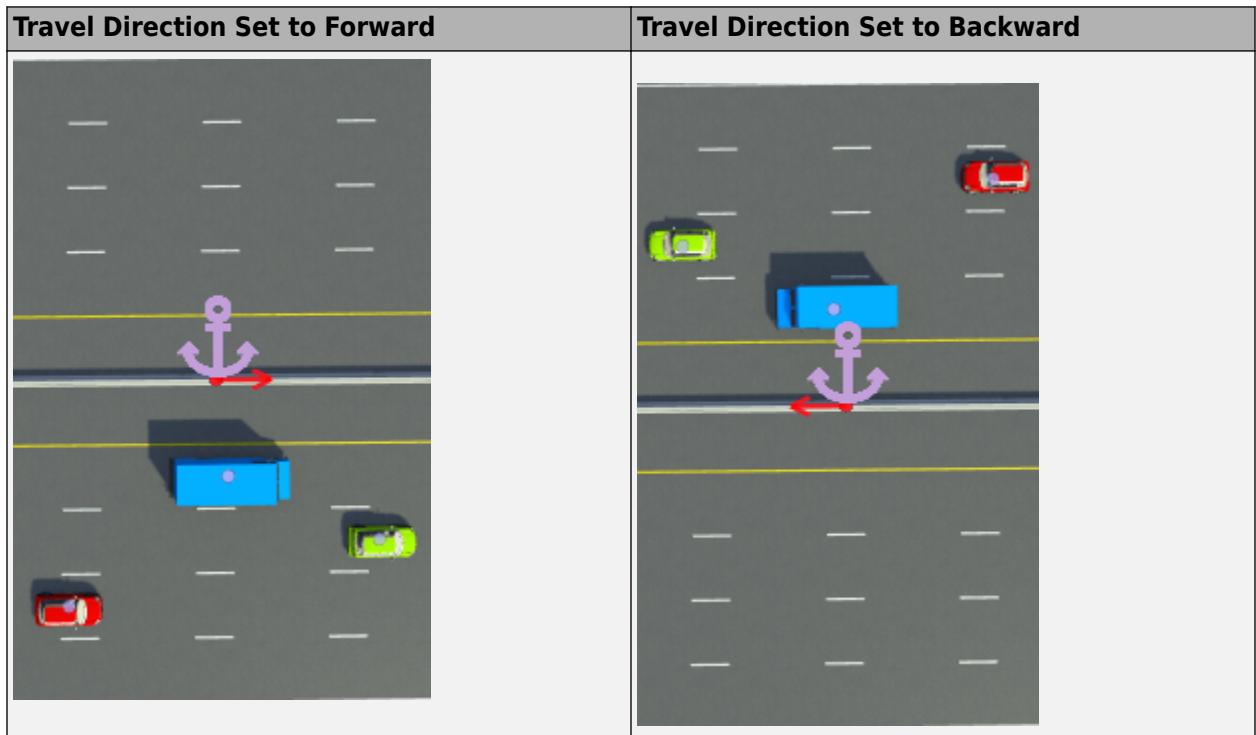
Align Objects Using Anchors

You can also use anchors as alignment points. For example, suppose you want to align two vehicles that have the same road anchor by their front bumpers.
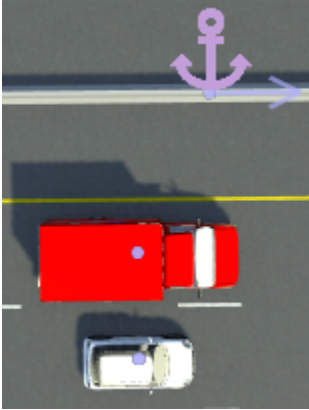
Follow these steps:

1 Select the first vehicle. Then, in the **Attributes** pane, under **Forward Offset**, select **Manual Reference Line**. You can now change how the **Offset** value from the anchor is measured. By default, **Offset** is measured from the middle of a vehicle.

2 Set **Reference Line** to **Front**. The offset of the vehicle is now measured from the front of the vehicle.

3 Set **Offset** to 0.

4 Repeat the previous steps for the second vehicle. The two vehicles now have their front bumpers aligned with the road anchor. If you drag the road anchor or adjust its **Distance** value in the **Attributes** pane, the front bumpers of the cars remain aligned.
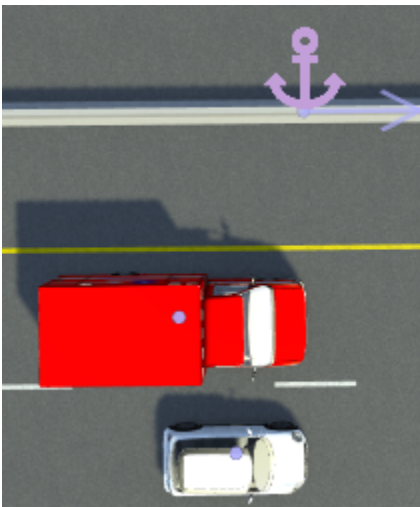
Set Anchors for Path Waypoints

When you specify paths for an actor, the path waypoints anchor to the road by default. Dragging an actor does not change the position of the path waypoint.

Path waypoints have the same **Forward Offset** and **Lane Offset** attributes as actors, except for the **Manual Reference Line** attribute.

Relocate Scenarios to Other Scenes

When you load a scenario into a new scene, objects in the scenario try to attach to the road anchor with the same name as the one in the previous scene. If an anchor with the same name is not present in the scenario, then all objects associated with that anchor become unanchored. For more details on relocating scenarios into new scenes, see "Relocate Scenarios" on page 3-117.

See Also

Related Examples
*   "Explore and Simulate a Simple Scenario" on page 1-13
*   "Relocate Scenarios" on page 3-117

Relocate Scenarios

In RoadRunner Scenario, you can move scenarios around within a scene and relocate scenarios to different scenes entirely. By relocating scenarios, you can test how driving algorithms handle the same scenario in various environments.

Relocate Scenario Within a Scene

RoadRunner Scenario uses an anchoring system in which the actors and path waypoints are positioned relative to specific anchor points, such as a point on a road or a point specifying an actor location. If you drag a road anchor along a road, any actors that are attached to that anchor move with it. For example, this figure shows a road anchor, and the scenario attached to it, moving from a straight portion of a road to a curved portion.
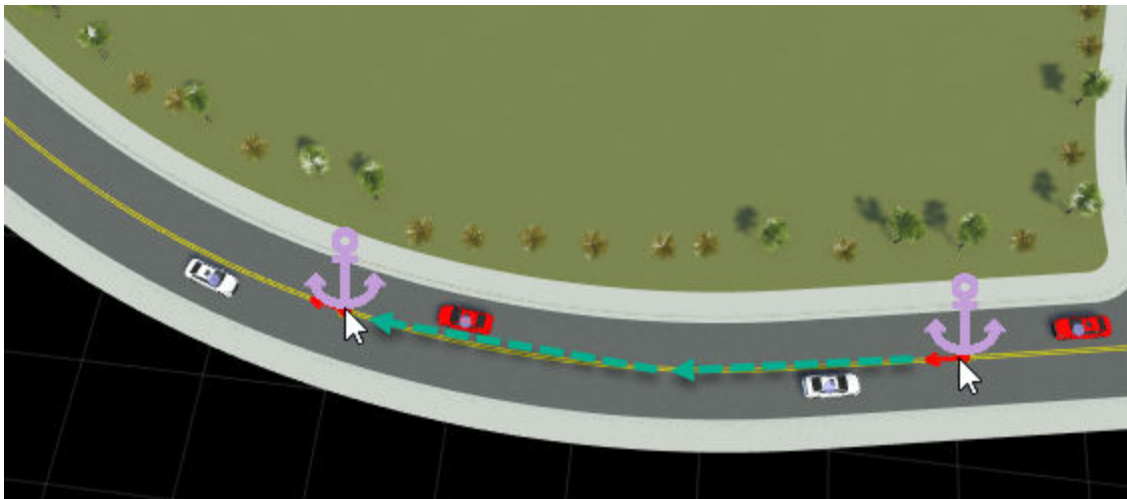
You can drag road anchors along only the road that they belong to. To move a scenario to an entirely new road, you must update the parent road anchor of each actor. For more details on changing parent anchors, see "Change Anchor Parent" on page 3-114.

When relocating scenarios along a road, actors maintain their lane-relative positions. For example, suppose a car is in the rightmost lane (zero lanes from the road edge) and a turning lane forms in the middle of the road. If you move the anchor along the road, the car moves into the turning lane to maintain a relative position of zero from the road edge.
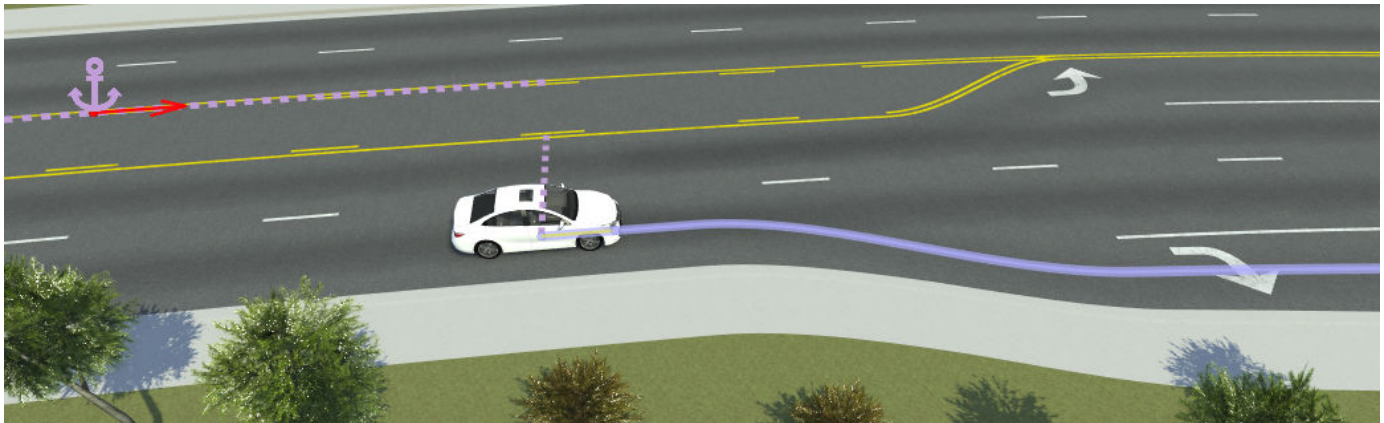
If you want a vehicle to stay in the same lane throughout a scenario, then you must set a predefined path for it. For an example of setting a predefined path, see "Design Path Following Scenario" on page 3-28.

Relocate Scenario to New Scene

RoadRunner Scenario provides both an interactive and a programmatic way to load an existing scenario into a new scene.

Note When you load a scenario into a new scene, the new scene must contain a road anchor with the same name as the parent anchor of the scenario actors. If no such anchor exists in the new scene, then the actors that were attached to that anchor become unanchored and attach to the scenario center point.

When developing scenes for use with multiple scenarios, consider using consistent names for all road anchors in the scenario. To change a road anchor name, select the anchor and update the **Name** attribute. To add new road anchors to scenes, use the **Road Anchor Tool**.

Relocate Scenarios Interactively

By default, when you open a scenario, the scenario opens into the scene that it was last saved with. To open a scenario into a different scene, follow these steps:

1   Switch to scene editing mode. From the top-right corner of the RoadRunner application, select **Scenario Editing**, then **Scene Editing**.
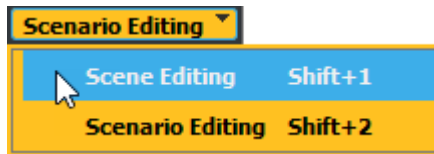
2   Open the scene that you want to load the scenario into. From the **File** menu, select **Open Scene**, and browse for the scene. For example, this figure shows the `SanAntonio` scene, which is one of the scenes included by default in the `Scenes` folder of RoadRunner projects.
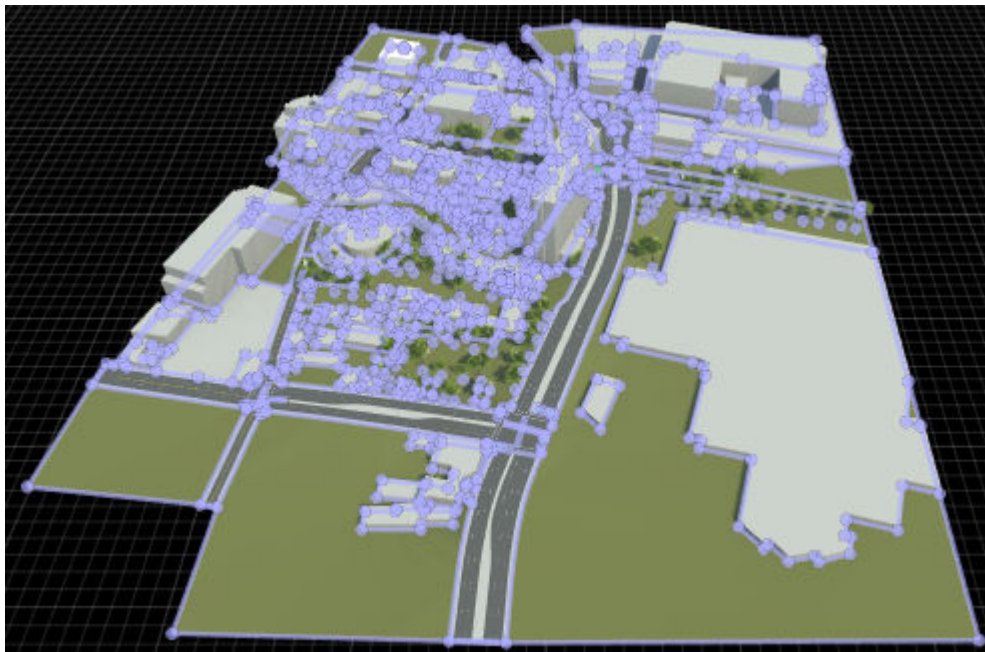
3   Switch to scenario editing mode. From the top-right corner of the RoadRunner application, select **Scene Editing**, then **Scenario Editing**.
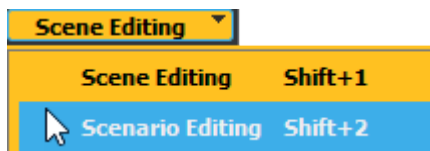
4   Load the scenario into the opened scene. From the **File** menu, select **Open Scenario into Current Scene**, and browse for the scenario. For example, this figure shows the `TrajectoryCutIn` scenario, which is one of the scenarios included by default in the `Scenarios` folder of RoadRunner projects.

In this example, loading the `TrajectoryCutIn` scenario into the `SanAntonio` scene worked because:

- The scene contains a road anchor named `ScenarioStart`.
- The actors in the `TrajectoryCutIn` scenario have the `ScenarioStart` anchor as their parent.

The `ScenarioBasic` scene also has an anchor named `ScenarioStart`, so you can load the scenario into this scene as well.
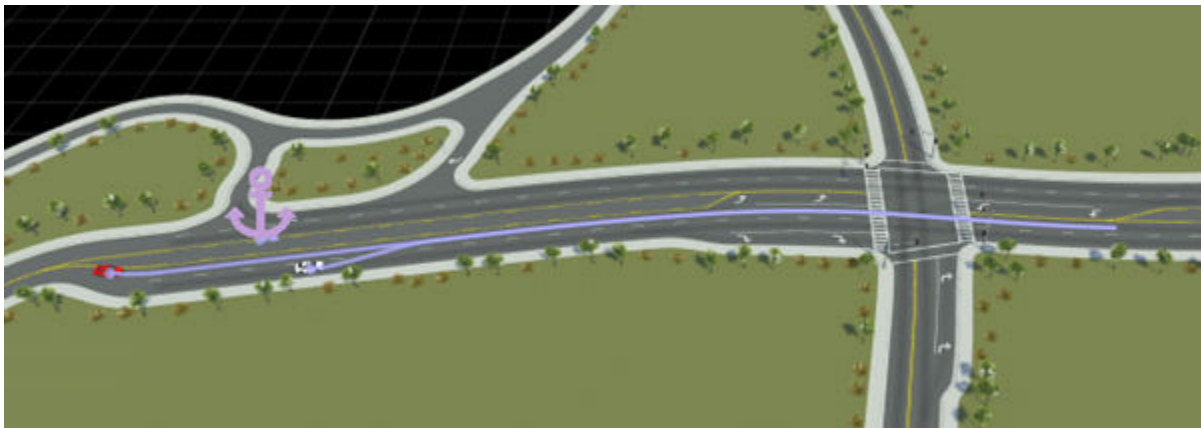
If you load the scenario into one of the default scenes that does not have a `ScenarioStart` anchor, such as `FourWayStop`, then the actors attach instead to the scenario center point, which can cause unexpected results. For example, in this figure, the actors drive off the scene and do not follow the lanes.
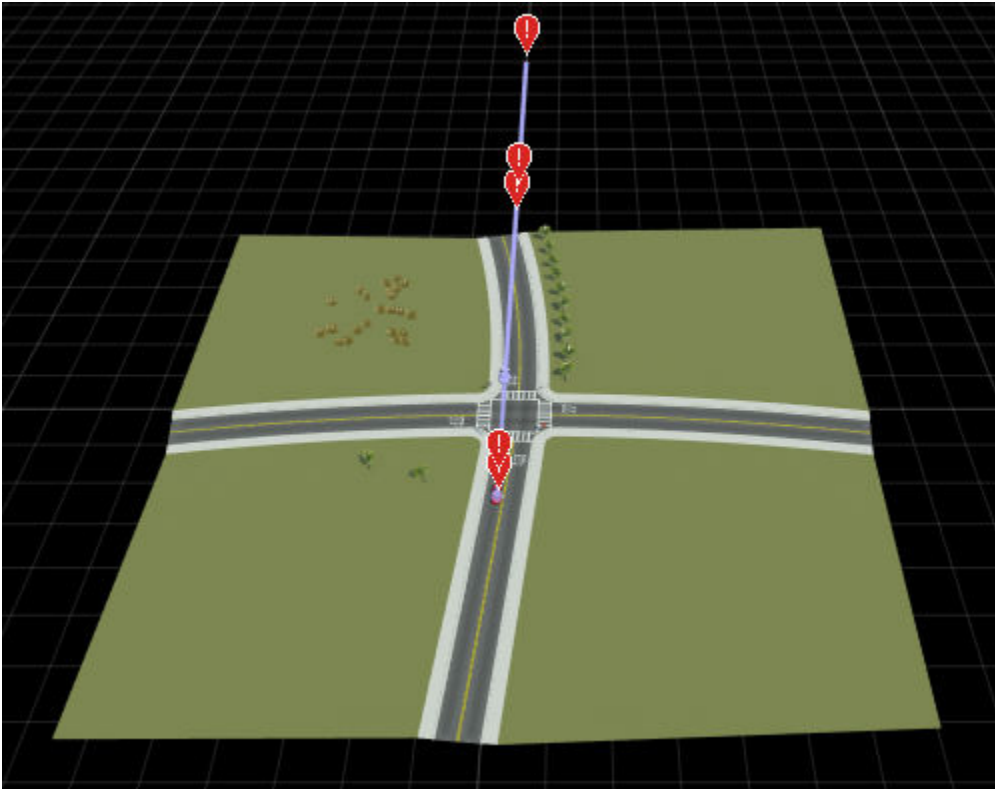
To resolve this issue, you can add a `ScenarioStart` anchor to the scene by using the **Road Anchor** tool.

Relocate Scenarios Programmatically

You can programmatically relocate a scenario to a new scene by using MATLAB functions. For example, this MATLAB code reproduces the example loading the `TrajectoryCutIn` scenario into the `SanAntonio` scene. It assumes that you have a project located under `C:\RR\MyProject` and you are using Windows.

```
projectFolder = fullfile("C:","RR","MyProject");
rrApp = roadrunner(projectFolder);
openScene(rrApp,"SanAntonio");
openScenario(rrApp,"TrajectoryCutIn",keepCurrentScene=true);
```

For more details on using these functions to relocate scenarios, see `SaveScenario`.

Alternatively, you can use the language-neutral gRPC API. This command-line code is equivalent to the previously shown MATLAB code.

```
AppRoadRunner --projectPath="C:\RR\MyProject"
CmdRoadRunnerApi "LoadScene(file_path='SanAntonio')"
CmdRoadRunnerApi "LoadScenario(file_path='TrajectoryCutIn' keep_current_scene='true')"
```

For more details on using the gRPC API to relocate scenarios, see "Reuse Scenarios in Multiple Scenes Using gRPC API" on page 4-13.

See Also

More About

- "Scenario Anchoring System" on page 3-108
- "Reuse Scenarios in Multiple Scenes Using gRPC API" on page 4-13

Validate Scenarios

RoadRunner Scenario checks for issues in a scenario at various stages of the scenario design process:

- As you edit a scenario, live warnings and errors display in the **Logic** editor, **Attributes** pane, or scenario editing canvas.
- When you simulate a scenario, RoadRunner Scenario checks that the simulation can run and displays scenario issues in the **Output** pane.
- When you export a scenario, the **Output** pane displays any issues with exporting valid scenario files.

Use these reports to debug your scenarios so that you can export valid scenarios.

Editing Checks

While editing a scenario, live warnings and errors can display in the **Logic** editor, the **Attributes** pane, or in the scenario editing canvas. In the scenario shown, a `Sedan` car has a path set to cross a dividing median lane into the opposing lane. The error displays visually as a red exclamation tag at the midpoint of the path segment that violates the check.
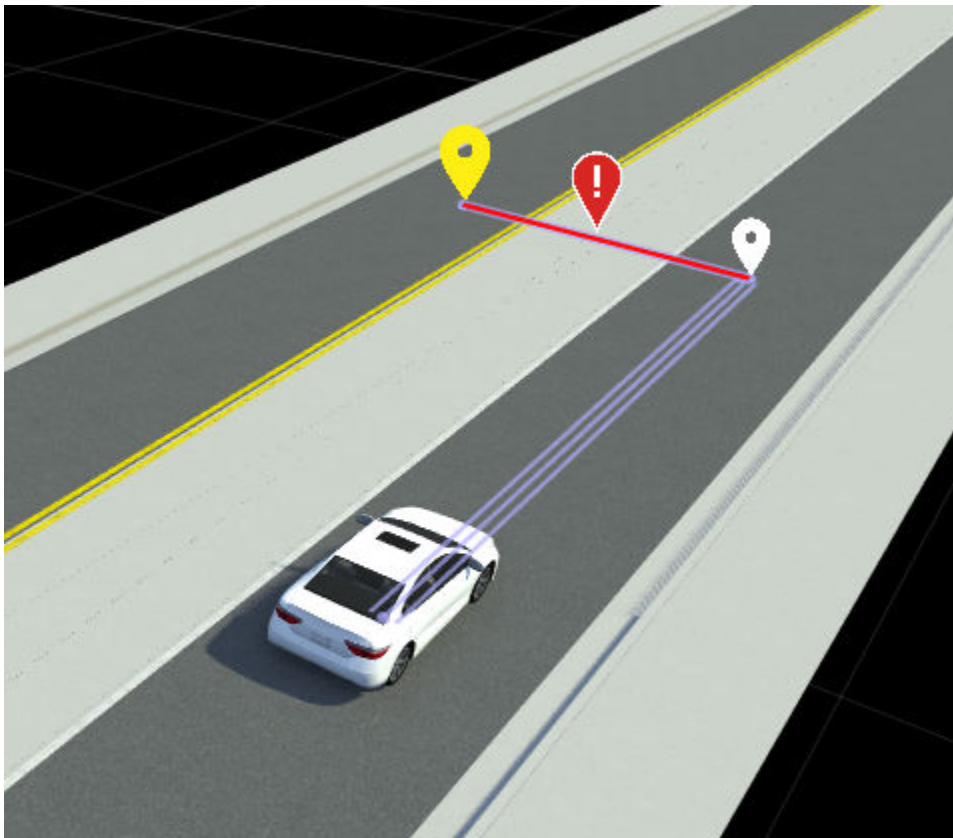
The check also appears as an error message in the **Attributes** pane.
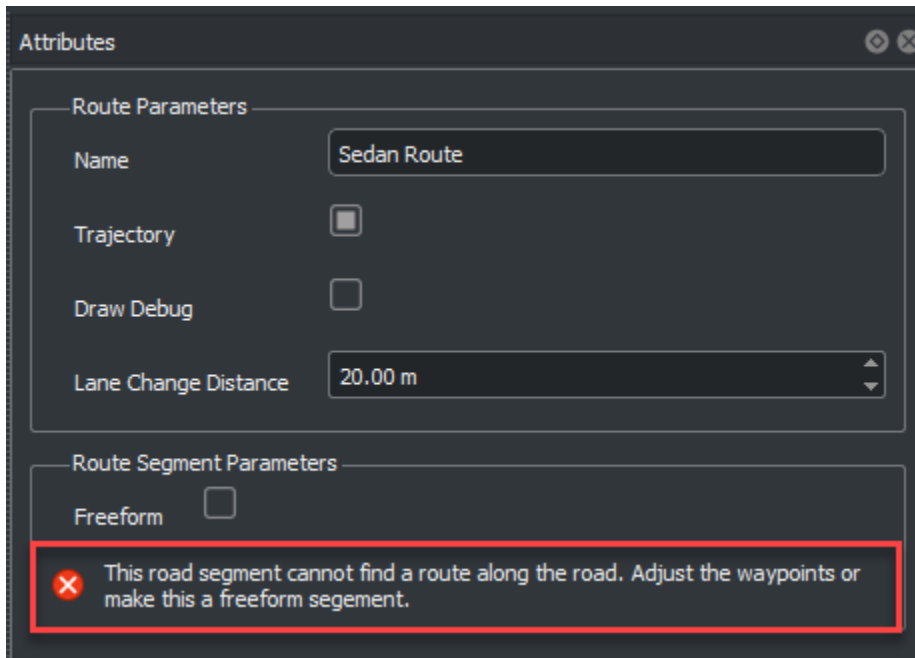
The error message shows the failed check and recommends a fix. For this particular error, the fix requires either altering the vehicle trajectory to move along valid paths, or to enable the vehicle to follow a freeform segment on this path.

Runtime Checks

Runtime checks occur at two points during a simulation. Prior to starting the simulation, the entire scenario is set up and configured to start running. RoadRunner Scenario applies a series of checks during this setup. Once the simulation starts running, certain events can cause the simulation to stop executing. This list shows an example of each of these check types.

- A runtime setup check occurs when a part of the scenario either violates a known constraint or is not defined. In the scenario shown, two sedans are driving in separate lanes.

The leading sedan, **Sedan2**, is configured with a speed change action phase relative to another actor.
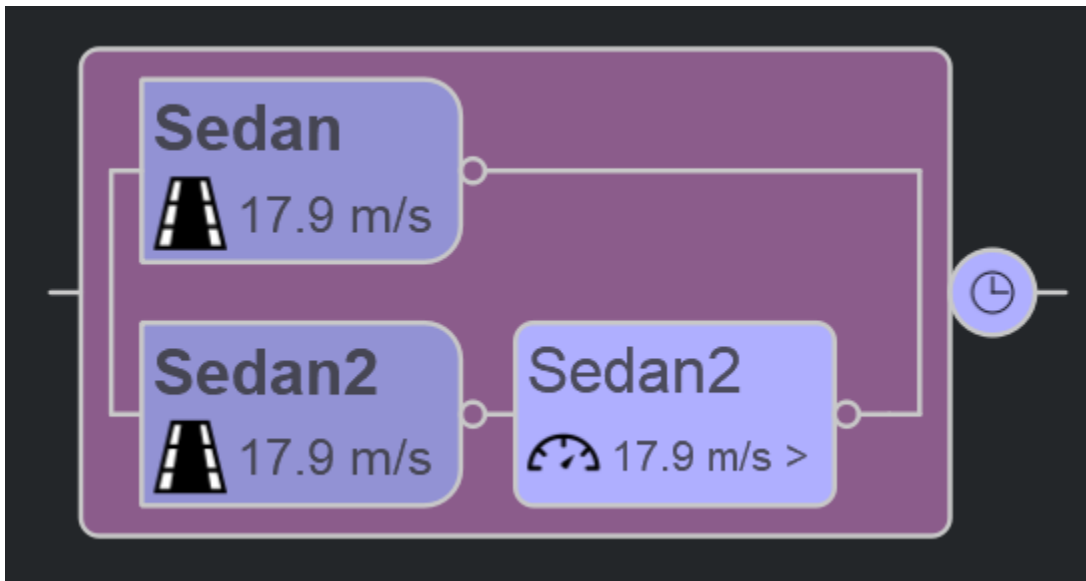
The action phase attributes, do not assign a **Reference Actor**, leaving the parameter set to
`<none>`.
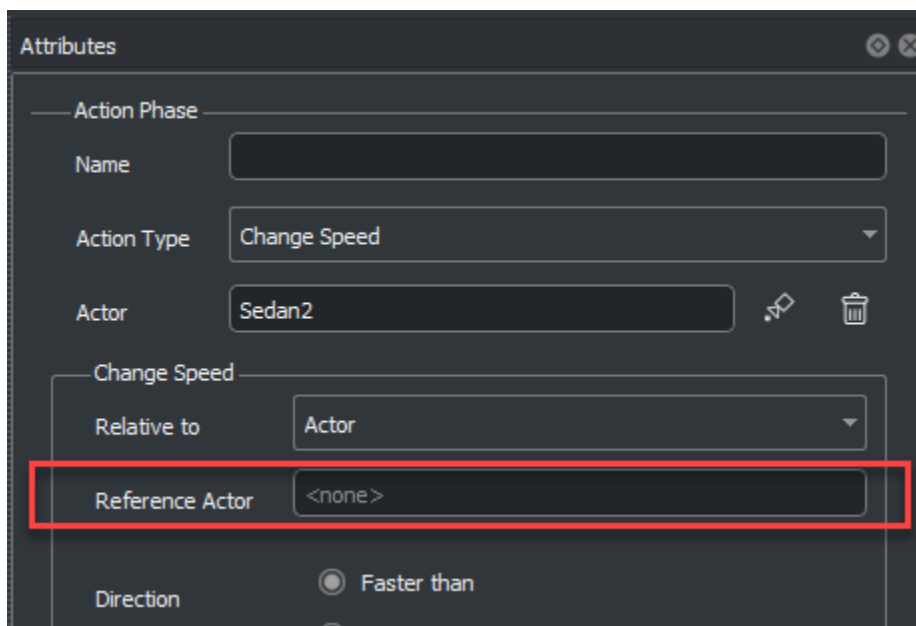
During simulation setup, the **Output** panel shows the issue as an error in red. A yellow hyperlink connects to the action phase attributes that fail the setup check, in this case the unset **Reference Actor** attribute.
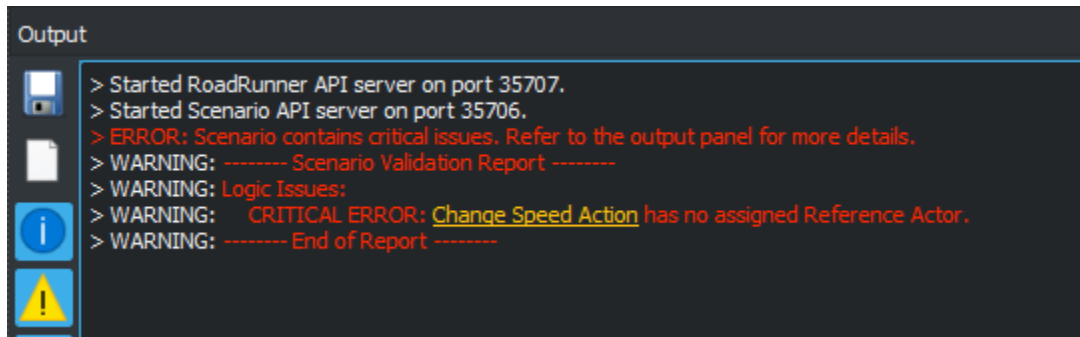
- RoadRunner Scenario performs a runtime check whenever an event occurs that requires the simulation to stop execution. For example, when a collision occurs between two cars, as shown in the figure, the simulation stops.
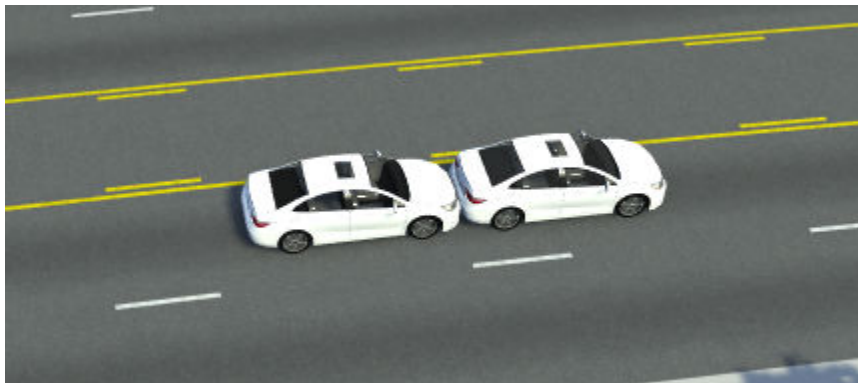
The **Output** panel displays the details of the error. Because the error condition occurs as part of the simulation, the error displays in white text rather than red. Runtime errors that occur as part of the simulation stop the simulation at the moment of the error. This enables you to investigate the error, in this case a collision, without missing the context of the error.
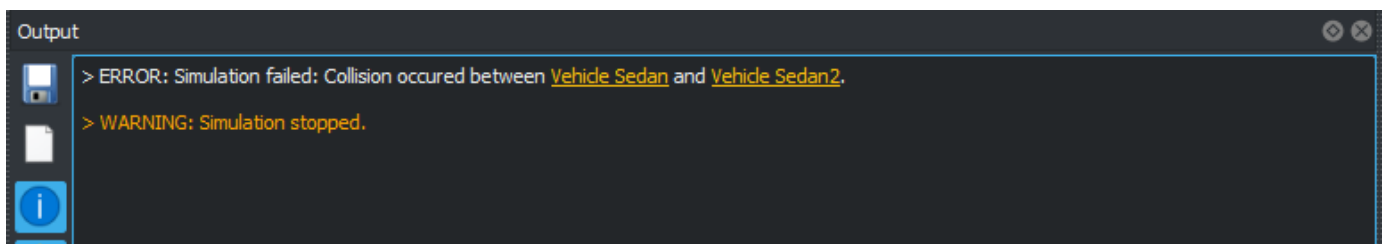
Export Checks

When exporting a scenario to ASAM OpenSCENARIO, the same runtime checks are applied to the scenario. Additionally, RoadRunner Scenario performs checks that are specific to this export format. For more information on exporting scenarios, see "Export to ASAM OpenSCENARIO" on page 5-2.

When a check fails, the warnings and errors are shown in the output of the export modal window. The results and errors are also shown in the **Output** pane in RoadRunner Scenario. Use the **Output** pane to find and address the failed checks before running the export operation again.
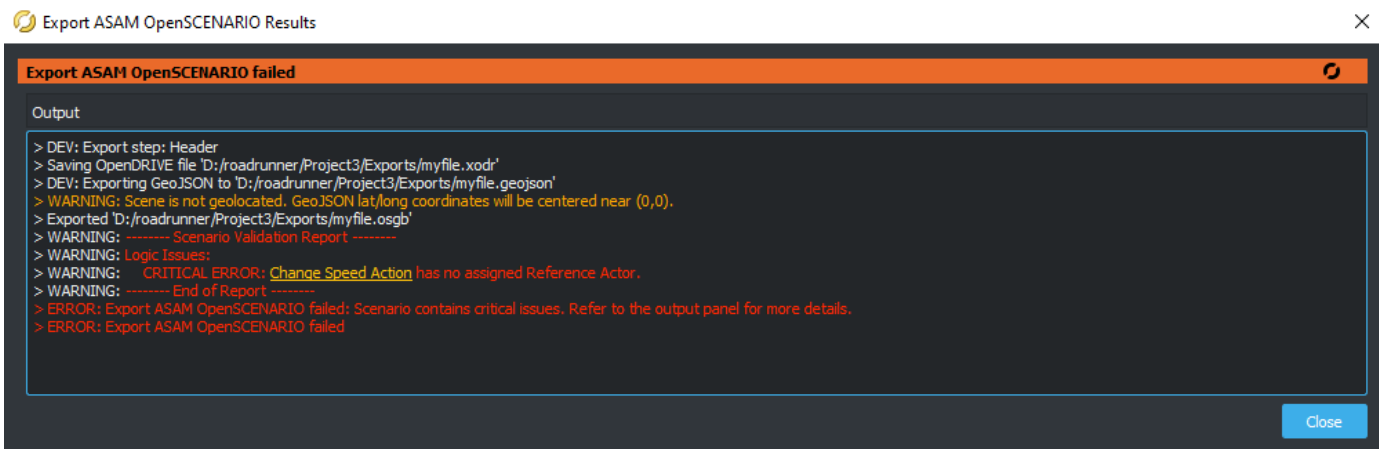
Note All the errors shown in the modal window also show in the **Output** pane, we recommend using the **Output** pane to debug any issues.

See Also

"Export to ASAM OpenSCENARIO" on page 5-2 | **Simulation Tool**

Built-In Behavior for Vehicles

RoadRunner Scenario defines autonomous behavior for vehicles. If you do not specify custom behavior for a vehicle, then that vehicle follows built-in behavior during simulation. RoadRunner Scenario supports these built-in behaviors for vehicles:

- Lane-following behavior — Vehicle drives along the center of its lane.
- Lane-changing behavior — Vehicle changes its lane when it receives a `Change Lane` action.
- Lateral offset behavior — Vehicle moves laterally to shift away from the center of its lane when it receives a `Change Lateral Offset` action.
- Longitudinal distance behavior — Vehicle keeps the specified distance or time gap from the reference actor when the vehicle receives a `Change Longitudinal Distance` action.
- Path-following behavior — Vehicle drives along the specified path. If you specify timing data for one or more waypoints along the path, the vehicle adjusts its speed to reach those waypoints at the specified times.

Note The speed of a vehicle must be greater than zero to simulate these behaviors.

Lane-Following Behavior

If you add a vehicle to a scenario and do not specify a path or custom behavior for the vehicle, then that vehicle drives along the center of its lane in the direction specified by the **Travel Direction** attribute of its lane. For more information about how to design a scenario with lane-following behavior, see "Design Lane Following Scenario" on page 3-2.

When a road ends, the vehicle chooses a lane in the new road that has the same travel direction as its lane in the current road. This behavior enables the vehicle to make turns into the appropriate lane, as shown in these images.

The vehicle drives in the right-turn lane toward an intersection.	The vehicle turns right onto the intersecting road, continuing its direction of travel.
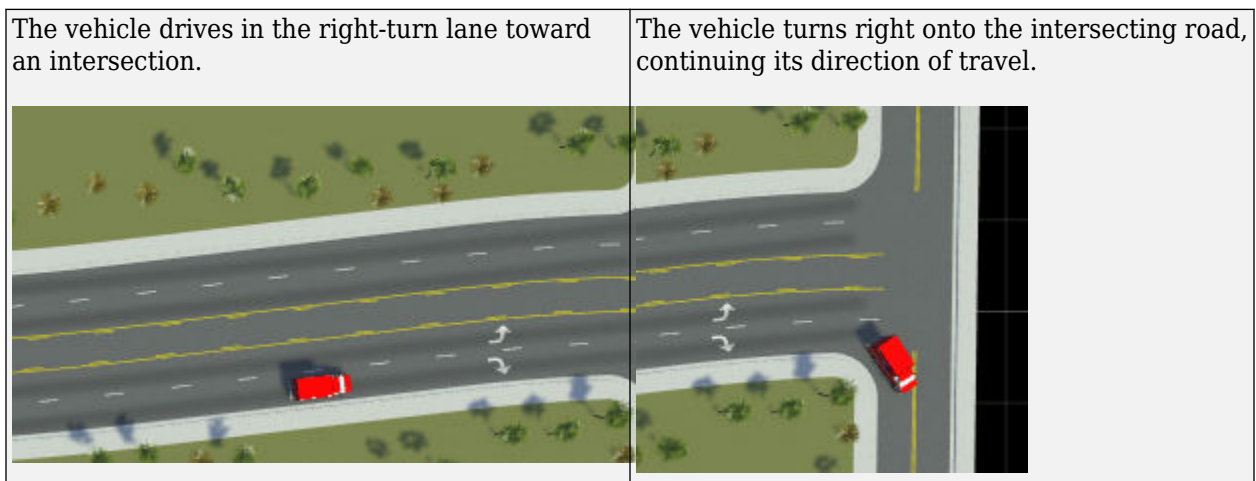
If there is no successor lane, or if the travel directions of the current lane and the successor lane do not match, then the vehicle stops upon reaching the end of its current lane. For example, this image shows a vehicle stopped at the end of a merging lane that does not have a successor lane.
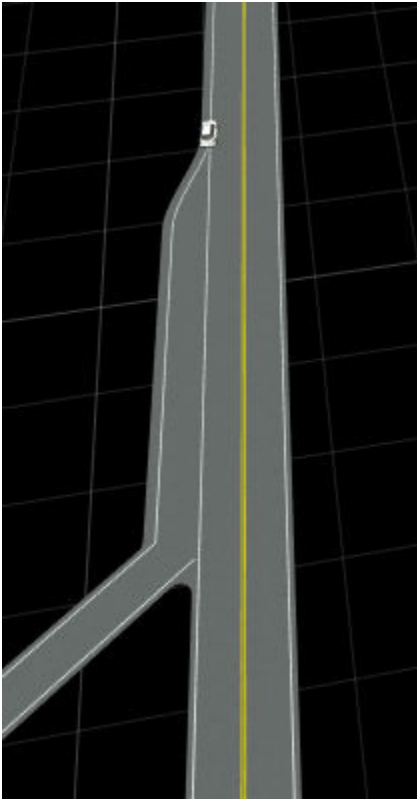
At junctions, the vehicle selects the lane that has the smallest change in curvature. For example, at this T-intersection, the vehicle goes straight instead of making a sharp turn.

Note When you set the **Dynamics Type** attribute of a `Change Speed` action to `With Acceleration`, you must specify an appropriate sign for the value of the **Acceleration** attribute. If the sign of the **Acceleration** value does not enable the vehicle to achieve the specified target **Speed** value, the vehicle ignores the sign.

Limitations

- Vehicles do not check for collisions with other vehicles or with static obstacles.
- If you place a vehicle such that its center is off the road or on a non-drivable lane, then the vehicle remains stationary during simulation. For example, if a vehicle is on a `Biking` lane, it remains stationary.
- Vehicles only consider lane centers while driving. For example, a vehicle can drive on lanes that are narrower than the width of the vehicle.
- Driving on lanes with a `Bidirectional` or `Undirected` travel direction is not supported.
- Vehicles ignore the specified **Speed Limit** attribute of their lane. Instead, each vehicle follows the specified **Speed** attribute of the `Initialize Speed` or `Change Speed` action.

Lane-Changing Behavior

The lane-changing behavior enables vehicles to perform the `Change Lane` action during simulation. For a vehicle to perform this action, a target lane must exist that satisfies these conditions:

- The target lane is a drivable lane.
- The **Travel Direction** attribute of a target lane matches the **Travel Direction** attribute of the current lane.

If a valid target lane exists, the vehicle transitions from the center of the current lane to the center of a target lane based on the specified attributes of the `Change Lane` action. For more information about how to specify attributes and design a scenario with lane-changing behavior, see "Design Lane Change Scenario" on page 3-10.

Note To simulate built-in lane-changing behavior, you must not specify a path for the vehicle. Otherwise, the vehicle ignores the `Change Lane` action and continues following the specified path.

To perform a `Change Lane` action relative to an actor, the vehicle first identifies its target lane as the lane that contains the center of the referenced actor at the instant the target receives the action. The vehicle then performs its action relative to the identified target lane. This behavior enables a vehicle to find an appropriate target lane even when the referenced actor is in the middle of a lane change when the vehicle receives `Change Lane` action.

When the vehicle cannot reach the specified target lane, it searches for the nearest possible valid lane. If a valid lane exists, then the vehicle performs a `Change Lane` action. Otherwise, the vehicle ignores this action and continues driving in the current lane. For example, consider these two scenarios:

- `Change Lane` action occurs

  A vehicle is traveling in the left-most lane on a three-lane, one-way road. If you specify the target lane as a lane that is offset from the current lane by three lanes, a drivable target lane does not exist on this road. In this case, the vehicle considers the drivable lane offset from its current lane by two lanes, and performs a `Change Lane` action.

Vehicle Before Change Lane Action	Vehicle After Change Lane Action
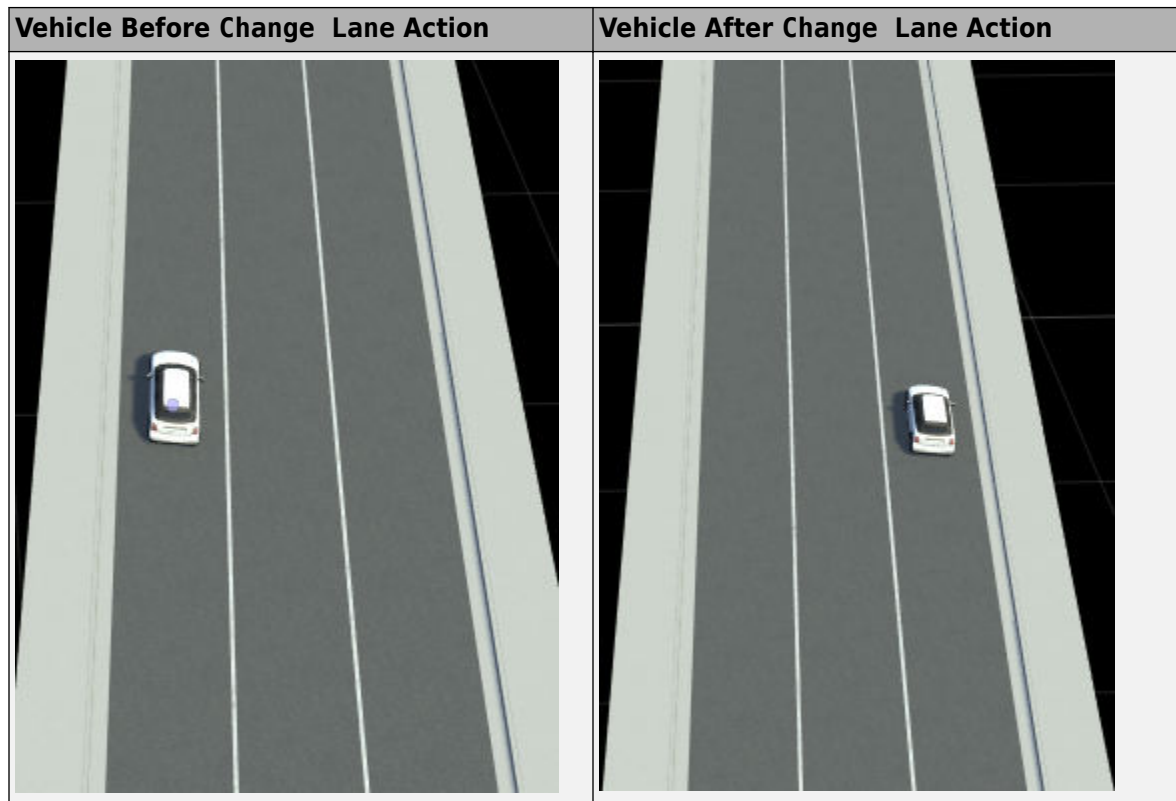
- Change Lane action does not occur

  A vehicle is driving on a two-way road that has only one driving lane for each travel direction. If you specify a target lane to the right of the current lane, the vehicle does not perform a Change Lane action.

If it is not feasible to perform a Change Lane action with the specified dynamics, the vehicle overrides the specified dynamics and performs the action instantaneously using the Step type of **Dynamics Profile**. For example, if you specify to change a lane over a distance that is less than the lane width, the vehicle ignores the specified dynamics and performs the action instantaneously.

Limitations

- Vehicles do not check for collisions with other vehicles or with static obstacles.

- When the specified target lane is offset by two or more lanes from the current lane, the vehicle sequentially checks whether each intermediate lane is a valid lane or not. If all the intermediate lanes are valid, the vehicle transitions to the specified target lane. Otherwise the vehicle transitions to the last valid intermediate lane starting from the current lane. If the intermediate lane next to the current lane is invalid, the vehicle does not perform the `Change Lane` action.

  If the vehicle cannot transition to the specified target lane, the `Change Lane` action phase must have a valid end condition to complete the action. Otherwise, the vehicle cannot perform all the remaining actions.

- When you specify a target lane relative to an actor that is traveling on an unconnected road segment, the specified lane change action does not occur, and the vehicle continues traveling in the current lane. For example, in this image, a white vehicle and a red vehicle are traveling on two distinct roads that are not connected to each other. For a white vehicle, if you specify a `Change Lane` action relative to the red vehicle, the white vehicle ignores this action.
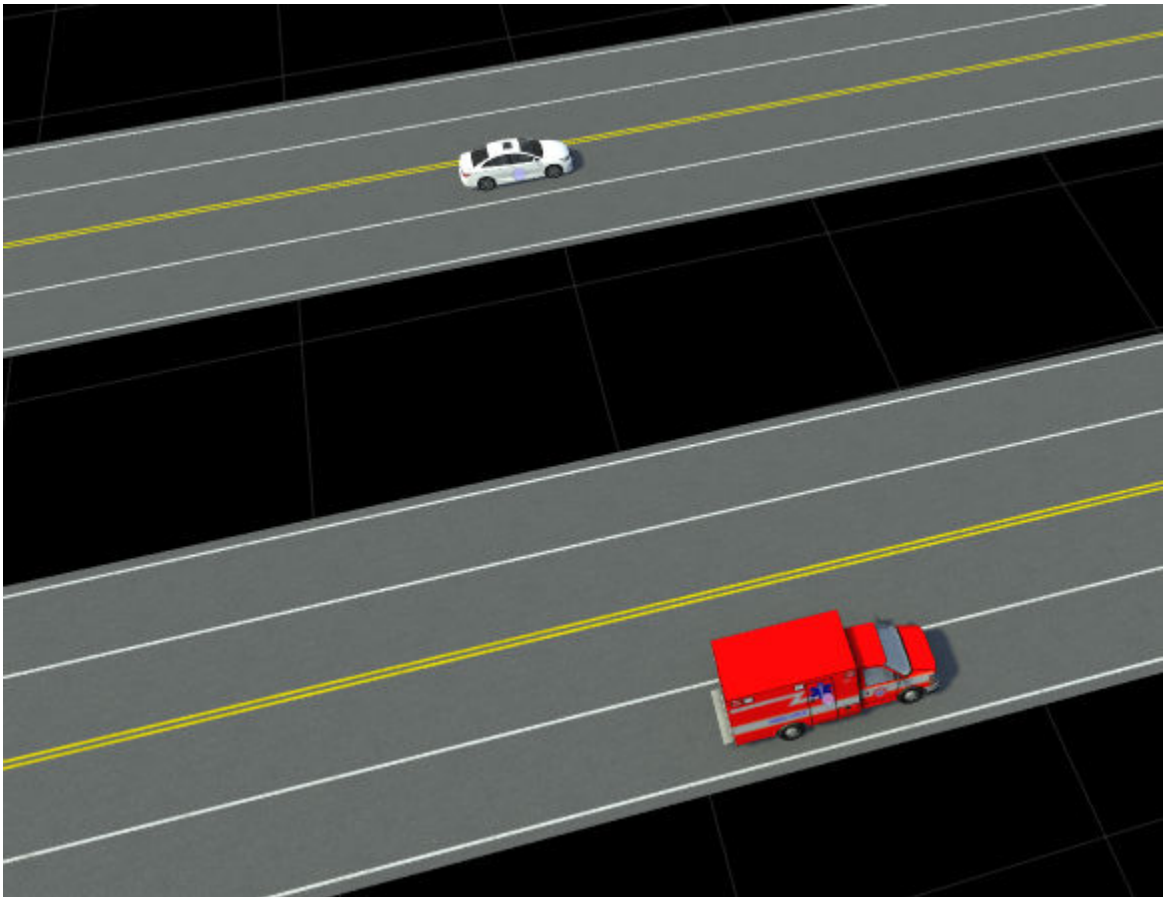
- If a vehicle is approaching the end of its road when it receives the `Change Lane` action, the vehicle does not perform the action if the remaining length of the road is insufficient for the vehicle to complete the action over specified distance or time. For example, consider a vehicle is 5 meters away from the end of the road when it receives a `Change Lane` action that specifies to reach the target lane over 10 meters. Because the specified distance is greater than the remaining length of the road, the vehicle does not perform this action.

- If a vehicle is traveling off the road when it receives a `Change Lane` action, the vehicle does not perform the action. For example, if you specify a lateral offset value greater the width of the road

for `Change Lateral Offset` action, the vehicle travels off the road when performing the `Change Lateral Offset` action. If the vehicle then receives a `Change Lane` action, the vehicle ignores the action and remains stationary until the simulation ends.

- If you set **Dynamics Type** attribute to `With lateral velocity`, then `Cubic` type of **Dynamics Profile** is not supported and the vehicle performs a `Change Lane` action using `Linear` type of **Dynamics Profile**.

- When you specify parallel action phases containing the `Change Lateral Offset` and `Change Lane` actions, the vehicle considers the bottom-most lateral action and ignores all other parallel lateral actions. For example, if a `Change Lane` action occurs above a `Change Lateral Offset` action in a parallel action phase, then the vehicle ignores the `Change Lane` action during simulation.
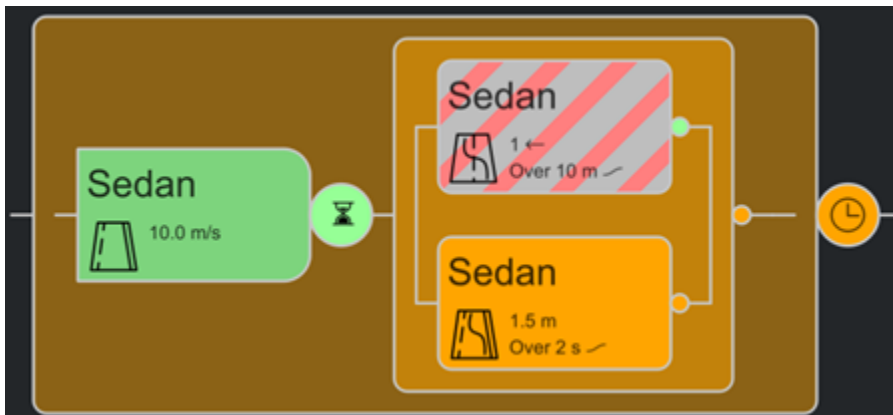
Lateral Offset Behavior

The lateral offset behavior enables vehicles to perform a `Change Lateral Offset` action that shifts them away from their lane centers during simulation. This enables you to design scenarios in which vehicles swerve from side to side within their lanes. For more information, see "Design Lane Swerve Scenario" on page 3-20.

The lateral offset behavior works for both types of **Motion** behaviors:

- `Follow Lane` — The `Change Lateral Offset` action applies a lateral shift to a vehicle such that it maintains the specified offset from its lane center during simulation.

- `Follow Path` — The vehicle maintains the specified lateral offset from the defined path of the vehicle during simulation.

A vehicle maintains the specified offset from its trajectory while traveling until you specify a new `Change Lateral Offset` action or a `Change Lane` action.

When your specified value for the **Time** attribute is not sufficient to attain the specified **Lateral Offset** value, the vehicle overrides your specified dynamics and performs a `Change Lateral Offset` action that attains the specified **Lateral Offset**.

Limitations

- Vehicles do not check for collisions with other vehicles or with static obstacles.

- When you specify parallel action phases containing the `Change Lateral Offset` and `Change Lane` actions, the vehicle considers the bottom-most lateral action and ignores all other parallel

lateral actions. For example, if a `Change Lateral Offset` action occurs above a `Change Lane` action in a parallel action phase, then the vehicle ignores the `Change Lateral Offset` action during simulation.
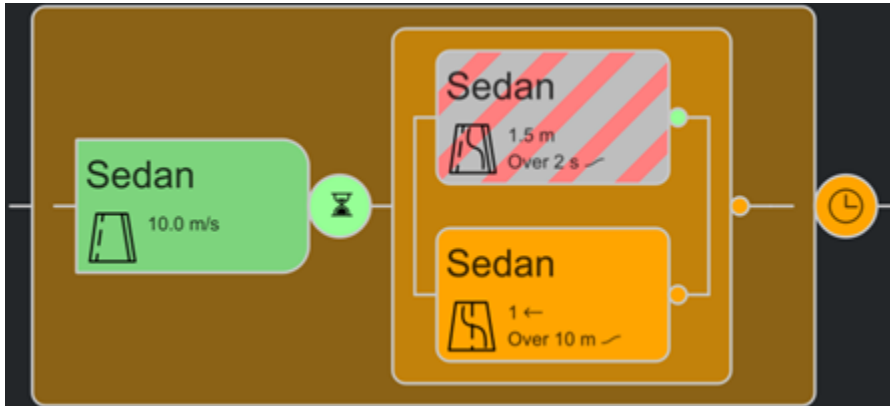
- The `Change Lateral Offset` action does not consider the **Travel direction** attribute of a lane. You must specify appropriate lateral offset value to avoid a vehicle traveling in the wrong direction on two-way roads.

- When you select a vehicle in the scenario canvas and specify a **Lateral Offset** value from the **Attributes** pane, the vehicle ignores your specified value while performing a `Change Lateral Offset` action. Instead, the vehicle considers the **Lateral Offset** value specified for this action when you define scenario logic.

Lateral Offset Attribute for Initial Vehicle Placement	Lateral Offset Attribute for Change Lateral Offset Action

Longitudinal Distance Behavior

The longitudinal distance behavior enables vehicles to perform a `Change Longitudinal Distance` action to maintain the specified longitudinal distance or time gap from other referenced vehicles. For more information about how to specify attributes and design a scenario with longitudinal distance behavior, see "Design Overtake Using Longitudinal Distance Condition Scenario" on page 3-44.

Note To simulate built-in longitudinal distance behavior, you must not specify a path for the vehicle. Otherwise, the vehicle ignores the `Change Longitudinal Distance` action.

To perform a `Change Longitudinal Distance` action, the vehicle first checks if the **Reference Actor** is traveling in its lane. If the **Reference Actor** is traveling in the lane of the vehicle, then the vehicle adjusts its speed to maintain the specified longitudinal distance or time gap from the **Reference Actor**. If a vehicle does not find the **Reference Actor** in its current lane, then the vehicle ignores the `Change Longitudinal Distance` action.

If you do not specify an end condition for the `Change Longitudinal Distance` action, then the vehicle ends the `Change Longitudinal Distance` action based on the value of the **Sampling Mode** parameter:

- `At start of action` — The vehicle ends the action when it achieves the specified longitudinal distance or time gap from the **Reference Actor**.
- `Continuous` — The vehicle continues following the action after achieving the specified longitudinal distance or time gap from the **Reference Actor**. The vehicle ends the action when one of these events occur:

  - The vehicle receives a `Change Speed` action.

    The vehicle receives a new `Change Longitudinal Distance` action.
  - The **Reference Actor** or the vehicle moves to a different lane.

During simulation, when a vehicle ends the `Change Longitudinal Distance` action, the vehicle continues traveling at the last computed longitudinal speed until it receives a `Change Speed` or a new `Change Longitudinal Distance` action.

Limitations

- Vehicles do not check for collisions with other vehicles or with static obstacles.
- When you specify parallel action phases containing the `Change Longitudinal Distance` and `Change Speed` actions, the vehicle considers the bottom-most longitudinal action and ignores all other parallel longitudinal actions. For example, if a `Change Longitudinal Distance` action occurs above a `Change Speed` action in a parallel action phase, then the vehicle ignores the `Change Longitudinal Distance` action during simulation.
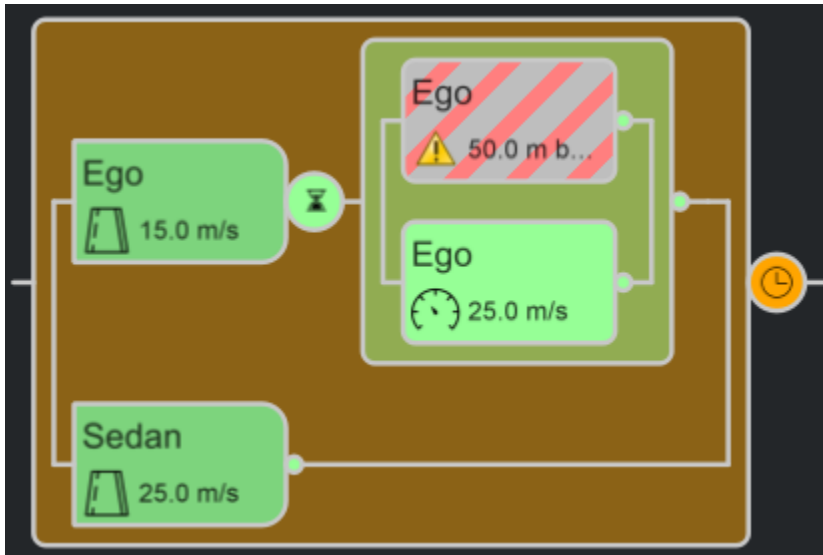
Path-Following Behavior

The path-following behavior enables vehicles to drive along your specified paths during simulation. This enables you to design more complex and unique scenarios in which you can specify custom driving path that can go on- and off-road. For more information, see "Design Path Following Scenario" on page 3-28.

A vehicle drives along your specified path until it reaches the final waypoint. Upon reaching the final waypoint, the vehicle remains stationary for the rest of the simulation.

If you specify the **Time** attribute for one or more waypoints along the path, and set the **Relative to** attribute of an `Initialize Speed` or `Change Speed` action to `Waypoint Time Data`, the vehicle adjusts its speed to reach those waypoints at the specified times when executing that action. The vehicle automatically calculates the required speed to reach a waypoint based on its **Time** value. If you also specify a **Speed** attribute for a waypoint, the vehicle reaches that waypoint at the specified time with the specified speed.

If you specify a nonzero value for the **Wait Time** attribute of the waypoint, the vehicle waits for the specified amount of time upon reaching that waypoint.

The vehicle starts following the specified **Timing Data** of waypoints after the corresponding `Change Speed` action is triggered. If the vehicle has already crossed one or more waypoints before the action is triggered, the vehicle ignores the **Timing Data** for those waypoints.

Limitations

- Vehicles do not check for collisions with other vehicles or with static obstacles.
- To get the expected timing behavior at waypoints, the value of the **Time** attribute for the first waypoint must be 0. RoadRunner Scenario does not support nonzero value for the **Time** attribute of the first waypoint.
- If the specified **Timing Data** for the next waypoint is unrealistic, the vehicle stops at the current waypoint and remains stationary for the rest of the simulation. For example, if the **Time** attribute of the next waypoint has the same value as that of the current waypoint, the vehicle stops at the current waypoint.

See Also

Related Examples

- "Design Lane Following Scenario" on page 3-2
- "Design Lane Change Scenario" on page 3-10
- "Design Lane Swerve Scenario" on page 3-20
- "Design Path Following Scenario" on page 3-28
- "Design Overtake Using Longitudinal Distance Condition Scenario" on page 3-44

More About

- "Define Scenario Logic" on page 3-70

Specify and Assign Actor Behaviors

In cosimulation, the behavior of any vehicle actor in the simulation can be managed by three simulation environments: RoadRunner Scenario, Automated Driving Toolbox, or CARLA. Choose a simulation environment based on the specific goals of your application.

Actor Behavior in RoadRunner

You can specify actor behavior directly in RoadRunner Scenario using the **Logic** editor, a graphical interface for defining the logic of a scenario. You can access the graphical **Logic** editor from the **2D Editor** pane. The scenario logic defined in this editor consists of a series of actions with optional conditions that trigger those actions. For more information on using the **Logic** editor, see "Define Scenario Logic" on page 3-70.

Actor Behavior in MATLAB and Simulink

Automated Driving Toolbox provides a cosimulation framework for simulating scenarios in RoadRunner Scenario with actors modeled in MATLAB and Simulink. For detailed information on the blocks, functions, and objects used to specify actor behavior from Automated Driving Toolbox, see "Overview of Simulating RoadRunner Scenarios with MATLAB and Simulink" (Automated Driving Toolbox).

Actor Behavior in CARLA

You can also develop actor behavior in the CARLA simulation environment. Using the cosimulation bridge, you can use actor behaviors from RoadRunner Scenario in a CARLA simulation. For information on specifying the actor behaviors in CARLA for cosimulation, see "Configure RoadRunner Scenario Model" on page 6-7.

See Also

More About
- "Define Scenario Logic" on page 3-70
- "Overview of Simulating RoadRunner Scenarios with MATLAB and Simulink" (Automated Driving Toolbox)
- "Configure RoadRunner Scenario Model" on page 6-7

Camera Control in RoadRunner Scenario

RoadRunner Scenario enables you to visualize large-scale and small-scale details of a 3D environment that can span many kilometers or miles. The **Simulation Tool** in RoadRunner Scenario supports in-editor playback for visualizing scenarios. The interactive camera types present in the **Simulation Tool** enable you to view this large 3D space quickly and effectively. To use the camera options, you must select the **Simulation Tool**. This example shows you the fundamentals of camera controls in the RoadRunner Scenario simulation environment.

Visualize Scenario Simulation using Camera Options

Open a prebuilt scenario, and then use the **Simulation Tool** to simulate the scenario.

From the **File** menu, select **Open Scenario**. Then, select the `TrajectoryCutIn` scenario, which is one of the prebuilt scenarios included by default in the `Scenarios` folder of RoadRunner Scenario.

In this scenario, one vehicle cuts in front of another vehicle. The vehicles follow predefined paths.
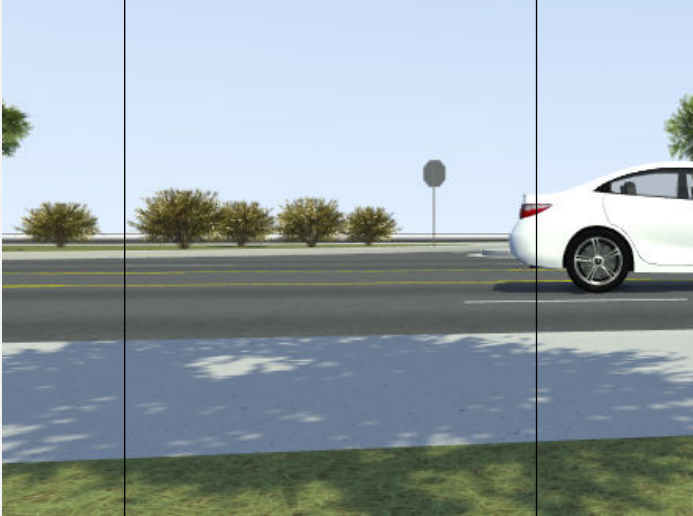
On the RoadRunner Scenario toolbar, click the **Simulation Tool**.
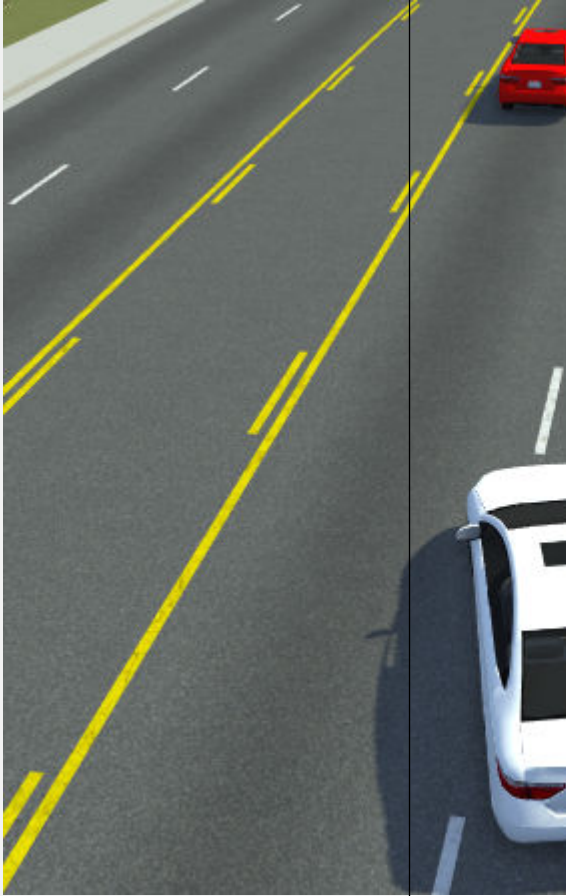
In the **Simulation** pane, under **Camera**, select an appropriate **Camera View** type. The **Camera View** drop-down has several camera options to conform to various simulation needs. Each camera view has associated attributes, which must be set corresponding to the camera view selected. This table shows the camera view types and their associated attributes.

3-139

Camera View Types	Description	Attributes
Default editor camera	Camera mode used for scene editing. Returns the view to default camera when switching from other camera types.	Simulation playback in **Default editor camera** view.
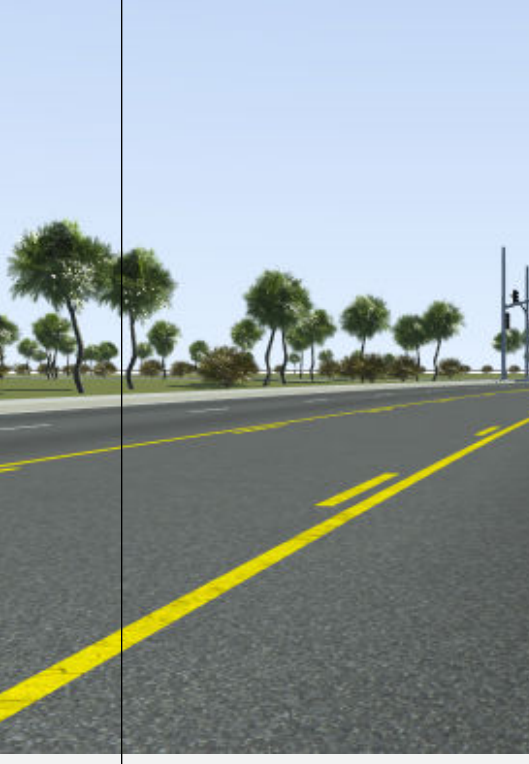
Camera View Types	Description	Attributes	
Orbit	Locks the camera position on to the selected **Actor**, allowing orbital control. Orbital control includes rotation, zoom in, zoom out, and pan about the vehicle.	**Actor**	Actor on which the camera is positioned during simulation. The camera is positioned in the same direction as the forward moving direction of the actor. The **Actor** drop-down lists all the vehicles in the scenario. Select an actor from the drop-down.
		Lock to Actor Orientation	Locks the camera to the orientation of the selected actor. This ensures that the camera rotates and moves with the selected actor. The starting position is the center of the vehicle, but can be panned like default editor camera with same controls. By default, this option is not selected.
		Simulation playback in **Orbit** camera view with the **Ego** vehicle selected as the **Actor**. The camera is positioned at a	

Camera View Types	Description	Attributes
		spot around the orbit of the **Ego** vehicle, the white sedan.

Camera View Types	Description	Attributes	
Follow	Positions the camera behind the selected actor.	**Actor**	Actor on which the camera is positioned during simulation. The camera is positioned in the same direction as the forward moving direction of the actor. The **Actor** drop-down lists all the vehicles in the scenario. Select an actor from the drop-down.
		Distance	Follow distance, in meters, for the camera. This attribute specifies the distance between the camera and the back of the selected actor.
		Height	Follow height, in meters, for the camera. This attribute specifies the height at which the camera is positioned above the actor while following it.
		Simulation playback in **Follow** camera view with the **Ego** vehicle selected as the **Actor**. The camera is positioned at a	

Camera View Types	Description	Attributes
		distance of 5.0 meters behind the vehicle and at a height of 6.0 meters above **Ego** vehicle, the white sedan.

Camera View Types	Description	Attributes	
Front	Positions the camera at the front of the selected actor.	**Actor**	Actor on which the camera is positioned during simulation. The camera is positioned in the same direction as the forward moving direction of the actor. The **Actor** drop-down lists all the vehicles in the scenario. Select an actor from the drop-down.
		Simulation playback in **Front** camera view with the **Ego** vehicle selected as the **Actor**. The camera is positioned at the front of the **Ego** vehicle, the white sedan. The red sedan is ahead of the white sedan. Hence, it is visible from the front camera positioned on the white sedan.	

Camera View Types	Description	Attributes

Note You can switch between the vehicles in the scenario in **Camera View** by pressing **Tab** keyboard shortcut. To shift backwards between the vehicles, press **Shift+Tab** keyboard shortcut.

You can switch between the camera types in **Camera View** by pressing **C** keyboard shortcut. To shift backwards between the camera types, press **Shift+C** keyboard shortcut.

After selecting the **Camera View**, under **Simulation Controls** click **Play** to simulate the scenario.

The scenario locks for editing and the simulation plays back in the scenario editing canvas. You can visualize the scenario based on the **Camera View** selected. For more details on simulating scenarios, see **Simulation Tool**.

See Also
Simulation Tool | **Scenario Edit Tool**

Related Examples
- Simulation Configuration
- "Explore and Simulate a Simple Scenario" on page 1-13
- "Camera Control in RoadRunner"

Programmatic Scenario Interfaces

Generate Scenario Variations Using gRPC API

This example shows how to programmatically vary attributes of a scenario and export the scenario variations to the ASAM OpenSCENARIO 1.0 file format. By taking a single scenario and varying certain aspects of it programmatically, you can quickly generate hundreds or even thousands of scenarios on which to test autonomous vehicle algorithms.

How the RoadRunner gRPC API Works

RoadRunner provides an API service for importing and exporting scenes and scenarios programmatically. This API is built using the open-source, language-neutral gRPC framework, which enables you to make remote procedure calls (RPCs) to the RoadRunner server to control the application programmatically. For more background, see "Control RoadRunner Programmatically Using gRPC API".

You can compile the RoadRunner API service into any programming language supported by gRPC and write clients to remotely control RoadRunner in that language. RoadRunner also provides a precompiled version of this API as a command-line tool. This example uses this precompiled helper command to perform these operations.

> **Note** This example primarily uses Windows commands and file paths to call the API, but this API also works on Linux.

Open RoadRunner and Start API Server

To use the command-line API to control RoadRunner remotely, you must first start the API server on an IP network port. Start the API server by opening a RoadRunner project, which you can do programmatically by using the `AppRoadRunner` executable file.

From the command line, navigate to the folder that contains the `AppRoadRunner` executable file. For example, this code shows the default installation location of the executable file on Windows:

```
cd "C:\Program Files\RoadRunner R2022b\bin\win64"
```

This code shows the default installation location of the executable file on Linux:

```
cd /usr/local/RoadRunner_R2022b/bin/glnxa64
```

Open RoadRunner to an existing project and set the port over which to run the server. Make these updates to the code:

- Replace the `projectPath` option value, `C:\RR\MyProject`, with a path to a valid RoadRunner project on your system. If you do not have an existing project, open RoadRunner and create one interactively. See "RoadRunner Project and Scene System".

- (Optional) Replace the `apiPort` option value, `54321`, with an IP network port number of your choice, between `1024` and `65535`, inclusive. If you omit the `apiPort` option, then RoadRunner opens to its default port of `35707`.

```
AppRoadRunner --projectPath="C:\RR\MyProject" --apiPort=54321
```

RoadRunner opens to a new scene in the specified project.

The **Output** pane displays the port on which the RoadRunner API server is running.

> Started RoadRunner API server on port 54321.

Switch to scenario editing mode. In the top-right corner of RoadRunner, select **Scene Editing**, then **Scenario Editing**. The **Output** pane displays an additional message indicating that the Scenario API server is running on its default port. This server is for cosimulating scenarios with MATLAB and Simulink, or with external simulators such as CARLA. The commands used in this example do not communicate with this server.

> Started RoadRunner API server on port 54321.
> Started Scenario API server on port 35706.

Load Scenario

Load the scenario that you want to create variations of. In this example, use the `TrajectoryCutIn.rrscenario` file, which is one of the default scenarios included with RoadRunner projects. From the **File** menu, select **Open Scenario**. Then, in the `Scenarios` folder, select `TrajectoryCutIn.rrscenario`.

In this scenario, one vehicle cuts into the same lane as another vehicle after a specified distance.

Preview this scenario by clicking the **Simulation Tool** button , and then clicking **Play**. In the **Logic** editor, click the `Match Lead: Ego` box and in the **Attributes** pane, set the **Acceleration** to 3.50 m/s2.
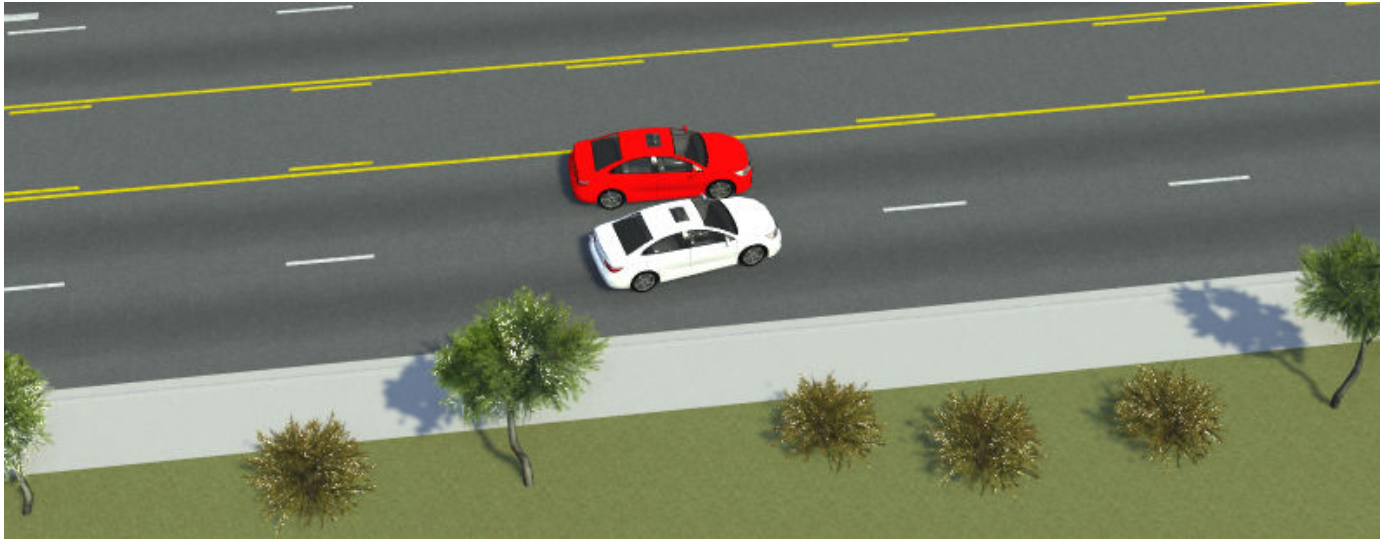
Define Scenario Variables

Using the scenario editing canvas, interactively define variables for the scenario attributes that you want to change programmatically.

Define Initial Speed Variable

Define a variable for the initial speed of the red sedan.

1
   Click the **Scenario Edit Tool** button to enable logic editing.

2   In the **Logic** editor, click the initialization box for the red `Sedan` vehicle.
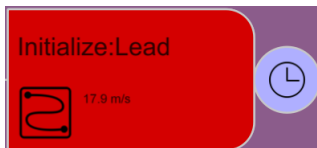
   The **Attributes** pane contains attributes that define the initial actions of the vehicle at the start of simulation. For example, the red sedan has an initial **Speed** value of `17.88` meters per second.
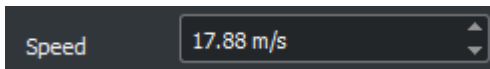
3   Create a variable for the initial speed of the vehicle. In the **Attributes** pane, right-click the **Speed** attribute name, and then click **Create Variable**.
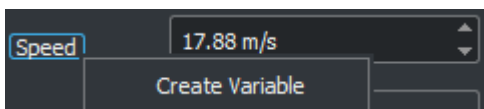
   RoadRunner switches focus to the **Variables** table, which now includes the `ChangeSpeed_TargetSpeed` variable and its value.

	Name	Value	
1	ChangeSpeed_TargetSpeed	17.88	🗑

4 In the **Name** field, rename the variable to `Red Sedan Initial Speed` to provide more context for the variable.

	Name	Value	
1	Red Sedan Initial Speed	17.88	🗑

Define Trigger Distance Variable

Define a variable for the waypoint in the path of the red sedan that triggers the white car, `Sedan`, to change lanes.

1

In the **Logic** editor, select the node for the **Distance To Point** condition for `Sedan`.

The scenario editor displays a blue sphere around a point near the beginning of the red sedan path. During simulation, when the red sedan enters this sphere, the white sedan begins its path. The **Attributes** pane displays attributes for changing this **Distance to Point** condition, such as the height of the point above the ground (**Height Offset**) and the radius around the point (**Threshold**).
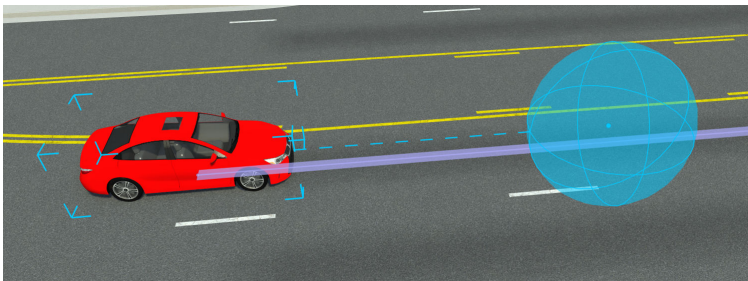
2 Select the path for the red sedan, and then select the path waypoint from which to offset the blue sphere.
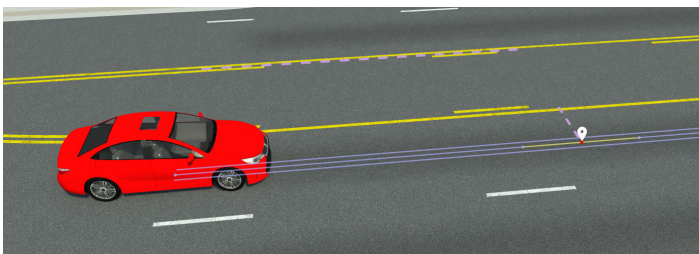
The **Attributes** pane displays attributes for the point. For example, under **Offset**, the **Forward Offset** attribute specifies the distance, in meters, by which the point is offset from the vehicle.

3 Create a variable for the forward offset. In the **Attributes** pane, under **Forward Offset**, right-click the **Offset** attribute name, and then click **Create Variable**.
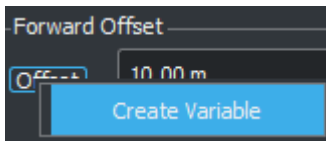
RoadRunner switches focus to the **Variables** table and displays a new
`{AttributeID}_FowardOffset` variable and its value of `10` meters in the table.

4 In the **Name** field, rename the variable to `Trigger Distance`.
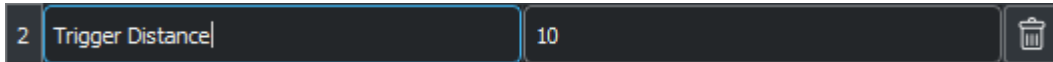
Define Vehicle Color and Type Variables

Define variables for the color and type of the white car, `Sedan`.

1 In the scenario editor, select the white car.
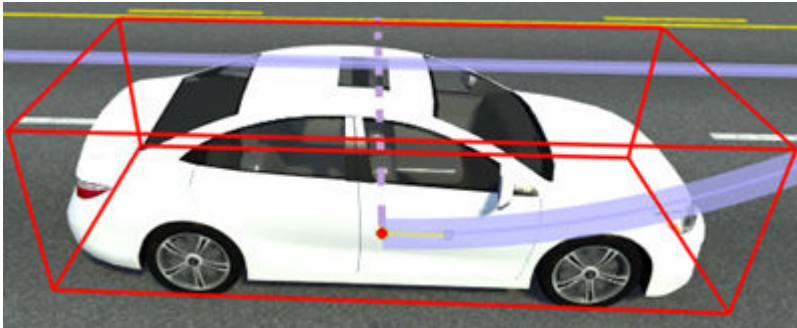
2 In the **Attributes** pane, right-click the **Color** attribute name and click **Create Variable**.
RoadRunner adds a `Sedan_Color` variable with a value of `#ffffff` (white) to the **Variables**
table.

3 Right-click the **Vehicle Type** attribute name and click **Create Variable**. RoadRunner adds a
`Sedan_AssetRef` variable to the **Variables** table. The variable value is the path to the asset that
RoadRunner uses to render the vehicle.

Modify Variables Programmatically

In the scenario editor, you can modify the variable values by updating their values in the **Variables**
table. Alternatively, you can modify the variables you defined by using the RoadRunner RPC methods.
The inputs for these methods are defined in the `roadrunner_service_messages.proto` protobuf
file located in this folder:

RRInstallFolder/bin/*platform*/Proto/mathworks/roadrunner

where:

- *RRInstallFolder* is your RoadRunner installation location.
- *platform* is your system platform (`win64` for Windows or `glnxa64` for Linux).

To call these RPC methods, in this example, use the `CmdRoadRunnerApi` helper command. This
helper command is located in the same folder as the `AppRoadRunner` executable file.

Increase the initial speed of the red `Sedan` vehicle to `50` m/s by using the `SetScenarioVariable` method. RoadRunner updates the corresponding `Red Sedan Initial Speed` value in the **Variables** table. In the `serverAddress` option, replace the port number with the `apiPort` value you specified when opening RoadRunner. If you used the default port, omit the `serverAddress` option.

```
CmdRoadRunnerApi "SetScenarioVariable(name='Red Sedan Initial Speed' value='50')" ^
--serverAddress=localhost:54321
```

	Name	Value	
1	Red Sedan Initial Speed	50	🗑

Simulate the scenario by using the `SimulateScenario` method. Slow the pacing down to 1/4 the speed of the simulation to observe the effects of the variable change. The red sedan now collides with the sedan.

```
CmdRoadRunnerApi "SimulateScenario(pacing.value='0.25')"  ^
--serverAddress=localhost:54321
```


Reset the initial speed value to `17.88`, increase the trigger distance to `50` meters, and rerun the simulation with an end time of `10` seconds. With the trigger distance farther out, the sedan cuts into the other lane behind the red sedan instead of in front of it.

```
CmdRoadRunnerApi "SetScenarioVariable(name='Red Sedan Initial Speed' value='17.88')" ^
--serverAddress=localhost:54321
CmdRoadRunnerApi "SetScenarioVariable(name='Trigger Distance' value='50')" ^
--serverAddress=localhost:54321
CmdRoadRunnerApi "SimulateScenario(pacing.value='0.25' simulation_end_time.value='10')" ^
--serverAddress=localhost:54321
```


Modify the `Sedan_Color` and `Sedan_AssetRef` variables to change the white car to a blue pickup truck. Switch back to the **Scenario Edit Tool** and observe the changes to the vehicle in the scenario editor.

```
CmdRoadRunnerApi "SetScenarioVariable(name='Sedan_Color' value='blue')" ^
--serverAddress=localhost:54321
CmdRoadRunnerApi "SetScenarioVariable(name='Sedan_AssetRef' value='<PROJECT>/Assets/Vehicles/PickupTruck.fbx_rrx')" ^
--serverAddress=localhost:54321
```
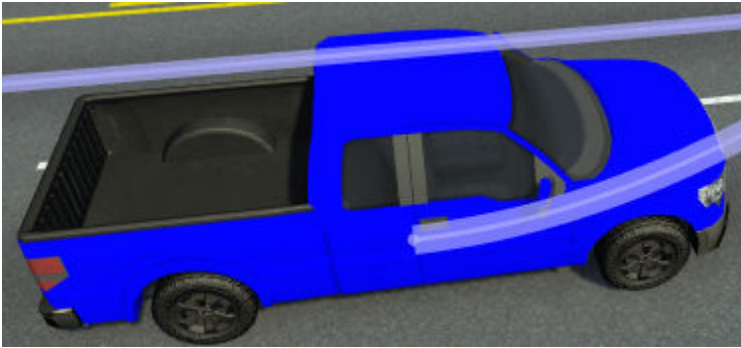
Restore the original conditions of the scenario and save the scenario as `TrajectoryCutInWithVariables` by using the `SaveScenario` method. By default, this method saves the scenario to the `Scenarios` folder of the current project.

```
CmdRoadRunnerApi "SetScenarioVariable(name=Red Sedan Initial Speed' value='17.88')" ^
--serverAddress=localhost:54321
CmdRoadRunnerApi "SetScenarioVariable(name='Sedan_Color' value='white')" ^
--serverAddress=localhost:54321
CmdRoadRunnerApi "SetScenarioVariable(name='Sedan_AssetRef' value='<PROJECT>/Assets/Vehicles/Sedan.fbx')" ^
--serverAddress=localhost:54321
CmdRoadRunnerApi "SaveScenario(file_path='TrajectoryCutInWithVariables')" ^
--serverAddress=localhost:54321
```

Export Single Scenario

Export the scenario to the ASAM OpenSCENARIO 1.0 format by using the `Export` method. For the exported file name, specify the same name as the scene, but with the extension `.xosc`. By default, RoadRunner exports the scene to the `Exports` folder of the current project.

```
CmdRoadRunnerApi "Export(file_path='TrajectoryCutInWithVariables.xosc' format_name='openscenario')" ^
--serverAddress=localhost:54321
```

Navigate to the `Exports` folder of your project and verify that it contains the ASAM OpenSCENARIO file, as well as the associated GeoJSON file, OpenSceneGraph file, and ASAM OpenDRIVE file.

```
cd "C:\RR\MyProject\Exports"
dir
```

```
TrajectoryCutInWithVariables.geojson
TrajectoryCutInWithVariables.osgb
TrajectoryCutInWithVariables.xodr
TrajectoryCutInWithVariables.xosc
```

Note The variables that you create are not included in the exported ASAM OpenSCENARIO file. They apply only during scenario editing and use of the API.

Export Scenario Variations

To export variations of a scenario, you can write a script that calls the `CmdRoadRunnerApi` command in a loop.

Copy the script for your platform to a file named `scenario_variations.bat` (Windows) or `scenario_variations.bash` (Linux) and modify the values for these variables:

- RRPATH — Update to the bin/*platform* folder path of your local RoadRunner installation.
- PROJECT — Update to the path of your RoadRunner project.
- SCENENAME — Update to the path for your scene.
- SCENARIONAME — Update to the path for your scenario.
- VEHICLESPEEDVARIABLE — Update to the name of the variable that sets the initial speed of the vehicle.
- PORT — Update to the IP network port that you want to connect to.

scenario_variations.bat (Windows)

```
@echo off
SetLocal EnableDelayedExpansion
REM Generate variations of cut-in scenario where a vehicle has varying speeds.

REM Set file paths, the variable name, and the port number.
set RRPATH=C:\Program Files\RoadRunner R2022b\bin\win64&
set PROJECT=C:\RR\MyProject&
set SCENENAME=ScenarioBasic&
set SCENARIONAME=TrajectoryCutInWithVariables&
set VEHICLESPEEDVARIABLE=Red Sedan Initial Speed&
set PORT=54321&

REM Open RoadRunner.
cd %RRPATH%
Start "" AppRoadRunner --projectPath=%PROJECT% --apiPort=%PORT%
timeout /t 2 /nobreak>nul& REM Wait for API server to start.

REM Load scene.
CmdRoadRunnerApi "LoadScene(file_path='%SCENENAME%')" --serverAddress=localhost:%PORT%
timeout /t 2 /nobreak>nul& REM Wait for scene to load

REM Load scenario.
CmdRoadRunnerApi "LoadScenario(file_path='%SCENARIONAME%' keep_current_scene='true')" --serverAddress=localhost:%PORT%

timeout /t 2 /nobreak>nul& REM Wait for scenario to load

REM Set initial speed, maximum speed, and speed increment.
set /A "SPEED = 10"
set /A "MAXSPEED = 50"
set /A "INCREMENT = 5"

REM Export initial scenario to a subfolder of "Exports" folder in current project.
set EXPORTPATHBASE=%PROJECT%\Exports\%SCENARIONAME%_%SCENENAME%_%SPEED%ms\%SCENARIONAME%_%SCENENAME%_%SPEED%ms
CmdRoadRunnerApi "Export(file_path='%EXPORTPATHBASE%.xosc' format_name='openscenario')" --serverAddress=localhost:!PORT!

REM Move OpenSceneGraph file for the scene up one level and rename it so it can be reused with other scenarios.
move %EXPORTPATHBASE%.osgb %PROJECT%\Exports\%SCENENAME%.osgb
set OSGPATH=%PROJECT%\Exports\%SCENENAME%.osgb

REM Export scenarios with varying speeds. Each scenario is exported to a separate folder.
:while
if %SPEED% lss %MAXSPEED% (
    set /A "SPEED = SPEED + INCREMENT"
    CmdRoadRunnerApi "SetScenarioVariable(name='!VEHICLESPEEDVARIABLE!' value='!SPEED!')" --serverAddress=localhost:!POR
    set EXPORTPATH=!SCENARIONAME!_!SCENENAME!_!SPEED!ms\!SCENARIONAME!_!SCENENAME!_!SPEED!ms.xosc
    set OSGSETTINGS=open_scenario_settings.path_to_existing_scene_graph='!OSGPATH!'
    CmdRoadRunnerApi "Export(file_path='!EXPORTPATH!' format_name='openscenario' !OSGSETTINGS!)" --serverAddress=localho
    goto :while
)

REM Exit RoadRunner
CmdRoadRunnerApi "Exit()" --serverAddress=localhost:%PORT%
```

```
scenario_variations.bash (Linux)

#!/bin/bash
# Generate variations of cut-in scenario where a vehicle has varying speeds.

# Set paths to RoadRunner, project, scene, and scenario, and set variable name.
RRPATH="/usr/local/RoadRunner_R2022b/bin/glnxa64"
PROJECT="local/RR/MyProject"
SCENENAME="ScenarioBasic"
SCENARIONAME="TrajectoryCutInWithVariables"
VEHICLESPEEDVARIABLE="Read Sedan Initial Speed"
PORT=54321

# Open RoadRunner.
cd "$RRPATH"
./AppRoadRunner --projectPath="$PROJECT" --apiPort="$PORT" &
sleep 2 # Wait for API server to start.

# Load scene.
./CmdRoadRunnerApi "LoadScene(file_path='$SCENENAME')" --serverAddress=localhost:$PORT
sleep 2 # Wait for scene to load.

# Load scenario.
./CmdRoadRunnerApi "LoadScenario(file_path='$SCENARIONAME' keep_current_scene='true')" --serverAddress=localhost:$PORT
sleep 2 # Wait for scenario to load.

# Set initial speed, maximum speed, and speed increment.
SPEED=10
MAXSPEED=50
INCREMENT=5

# Export initial scenario to a subfolder of "Exports" folder in current project.
EXPORTPATHBASE="$PROJECT"/Exports/"$SCENARIONAME"_"$SCENENAME"_"$SPEED"ms/"$SCENARIONAME"_"$SCENENAME"_"$SPEED"ms
./CmdRoadRunnerApi "Export(file_path='$EXPORTPATHBASE.xosc' format_name='openscenario')" --serverAddress=localhost:$PORT

# Move OpenSceneGraph file for the scene up one level and rename it so it can be reused with other scenarios.
mv "$EXPORTPATHBASE".osgb "$PROJECT"/Exports/"$SCENENAME".osgb
OSGPATH="$PROJECT"/Exports/"$SCENENAME".osgb

# Export scenarios with varying speeds. Each scenario is exported to a separate folder.
while [ $SPEED -lt $MAXSPEED ]
do
  let SPEED+=INCREMENT
  ./CmdRoadRunnerApi "SetScenarioVariable(name='$VEHICLESPEEDVARIABLE' value='$SPEED')" --serverAddress=localhost:$PORT
  EXPORTPATH="$SCENARIONAME"_"$SCENENAME"_"$SPEED"ms/"$SCENARIONAME"_"$SCENENAME"_"$SPEED"ms.xosc
  ./CmdRoadRunnerApi "Export(file_path='$EXPORTPATH' format_name='openscenario' \
  open_scenario_settings.path_to_existing_scene_graph='$OSGPATH')" --serverAddress=localhost:$PORT
done

# Exit RoadRunner.
./CmdRoadRunnerApi "Exit()" --serverAddress=localhost:$PORT
```

This script performs these actions:

1 Opens RoadRunner to a project and starts the RoadRunner API server.

2 Loads the scene and scenario.

3 Varies the initial speed variable for the vehicle and exports the scenario to ASAM OpenSCENARIO 1.0. Because the same scene is used for each scenario, the script reuses the OpenSceneGraph file generated by the first export to reduce export time.

4 Exits RoadRunner and shuts down the RoadRunner server.

Close any open instances of RoadRunner and call this script from the command line. For example, if you used the `.bat` script, then call this command from a Windows command prompt.

> **Note** If you receive `Permission Denied` errors when running this script on Windows, you might need to run this script as an administrator. Right-click the script in File Explorer and select **Run as administrator**.

```
scenario_variations
```

To verify that the script exported the scenarios, navigate to the `Exports` folder of the specified project and list the contents of the folder. For example, if you specified a project at path `"C:\RR\MyProject"`, run these commands. The folder contains subfolders for each scenario variation and the OpenSceneGraph file for the scene.

```
cd "C:\RR\MyProject\Exports"
dir
```

```
ScenarioBasic.osgb
TrajectoryCutInWithVariables_ScenarioBasic_10ms
TrajectoryCutInWithVariables_ScenarioBasic_15ms
TrajectoryCutInWithVariables_ScenarioBasic_20ms
TrajectoryCutInWithVariables_ScenarioBasic_25ms
TrajectoryCutInWithVariables_ScenarioBasic_30ms
TrajectoryCutInWithVariables_ScenarioBasic_35ms
TrajectoryCutInWithVariables_ScenarioBasic_40ms
TrajectoryCutInWithVariables_ScenarioBasic_45ms
TrajectoryCutInWithVariables_ScenarioBasic_50ms
```

Navigate to one of the scenario folders to verify that it contains the exported GeoJSON file, ASAM OpenDRIVE file, and ASAM OpenSCENARIO file.

```
cd TrajectoryCutInWithVariables_ScenarioBasic_10ms
dir
```

```
TrajectoryCutInWithVariables_ScenarioBasic_10ms.geojson
TrajectoryCutInWithVariables_ScenarioBasic_10ms.xodr
TrajectoryCutInWithVariables_ScenarioBasic_10ms.xosc
```

Extend Scenario Variation Options

To extend this script further, you can:

- Vary the `Trigger Distance` variable instead of the initial speed variable, and export variations based on when the vehicle begins its cut-in maneuver.
- Vary the `Type` and `Color` variables to export scenarios using various vehicles.
- Vary combinations of these variables, or create new variables, to export multiple permutations of the scenario.

For additional flexibility in using the API, you can also compile the protocol buffer (protobuf) files that define the API into your desired programming language. You can then write client applications in those languages. For more details on compiling protobuf files and writing clients, see "Compile Protocol Buffers for RoadRunner gRPC API".

See Also

AppRoadRunner | LoadScenario | SetScenarioVariable | SimulateScenario | Export

Related Examples

- "Control RoadRunner Programmatically Using gRPC API"
-
-

Reuse Scenarios in Multiple Scenes Using gRPC API

This example shows how to programmatically load scenarios into different scenes and export the scene and scenario combinations to the ASAM OpenSCENARIO 1.0 file format. By loading a scenario into multiple scenes, you can test how your driving algorithm handles specific driving scenarios under varying conditions.

How the RoadRunner gRPC API Works

RoadRunner provides an API service for importing and exporting scenes and scenarios programmatically. This API is built using the open-source, language-neutral gRPC framework, which enables you to make remote procedure calls (RPCs) to the RoadRunner server to control the application programmatically. For more background, see "Control RoadRunner Programmatically Using gRPC API".

You can compile the RoadRunner API service into any programming language supported by gRPC and write clients to remotely control RoadRunner in that language. RoadRunner also provides a precompiled version of this API as a command-line tool. This example uses this precompiled helper command to perform these operations.

Note This example primarily uses Windows commands and file paths to call the API, but this API also works on Linux.

Open RoadRunner and Start API Server

To use the command-line API to control RoadRunner remotely, you must first start the API server on an IP network port. Start the API server by opening a RoadRunner project, which you can do programmatically by using the `AppRoadRunner` executable file.

From the command line, navigate to the folder that contains the `AppRoadRunner` executable file. For example, this code shows the default installation location of the executable file on Windows:

```
cd "C:\Program Files\RoadRunner R2022b\bin\win64"
```

This code shows the default installation location of the executable file on Linux:

```
cd /usr/local/RoadRunner_R2022b/bin/glnxa64
```

Open RoadRunner to an existing project and set the port over which to run the server. Make these updates to the code:

- Replace the `projectPath` option value, `C:\RR\MyProject`, with a path to a valid RoadRunner project on your system. If you do not have an existing project, open RoadRunner and create one interactively. See "RoadRunner Project and Scene System".
- (Optional) Replace the `apiPort` option value, `54321`, with an IP network port number of your choice, between `1024` and `65535`, inclusive. If you omit the `apiPort` option, then RoadRunner opens to its default port of `35707`.

```
AppRoadRunner --projectPath="C:\RR\MyProject" --apiPort=54321
```

RoadRunner opens to a new scene in the specified project.

The **Output** pane displays the port on which the RoadRunner API server is running.

> Started RoadRunner API server on port 54321.

Switch to scenario editing mode. In the top-right corner of RoadRunner, select **Scene Editing**, then **Scenario Editing**. The **Output** pane displays an additional message indicating that the Scenario API server is running on its default port. This server is for cosimulating scenarios with MATLAB and Simulink, or with external simulators such as CARLA. The commands used in this example do not communicate with this server.

> Started RoadRunner API server on port 54321.
> Started Scenario API server on port 35706.

Load and Simulate Scenario

Load the `TrajectoryCutIn` scenario, which is one of the sample scenarios included by default in the `Scenarios` folder of RoadRunner projects. To load the scenario, use the `CmdRoadRunnerApi` helper command. This command is in the same folder as the `AppRoadRunner` executable file.

Call the `LoadScenario` RPC method using this command. In the `serverAddress` option, replace the port number with the `apiPort` value you specified when opening RoadRunner. If you used the default port, omit the `serverAddress` option.

```
CmdRoadRunnerApi "LoadScenario(file_path='TrajectoryCutIn')" --serverAddress=localhost:54321
```

The scenario loads into the default `ScenarioBasic` scene, because `TrajectoryCutIn` was saved with this scene.
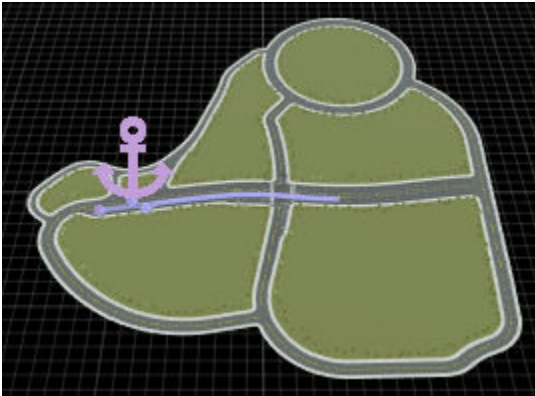
In this scenario, one vehicle cuts into the same lane as another vehicle after a specified distance. Simulate this scenario by using the `SimulateScenario` method. Slow down simulation pacing to 1/2 the speed to better observe the scenario.

```
CmdRoadRunnerApi "SimulateScenario(pacing.value='0.5')" --serverAddress=localhost:54321
```


Load Scenario into Different Scene

Load the `TrajectoryCutIn` scenario into a different scene. First, load the `SanAntonio` scene, which is another of the default scenes included with RoadRunner projects.

```
CmdRoadRunnerApi "LoadScene(file_path='SanAntonio')" --serverAddress=localhost:54321
```

Load the `TrajectoryCutIn` scenario into this scene.

```
CmdRoadRunnerApi "LoadScenario(file_path='TrajectoryCutIn' keep_current_scene='true')" ^
--serverAddress=localhost:54321
```


> **Note** When you load a scenario into a new scene, that scene must contain a road anchor with the same name as the parent anchor of the scenario actors. For more details on working with road anchors, see "Scenario Anchoring System" on page 3-108.

The `ScenarioBasic` and `SanAntonio` scene both have a road anchor named `ScenarioStart`. In both scenes, the scenario is located relative to this anchor. If you load this scenario into a scene that does not have this anchor, then RoadRunner positions the scenario relative to the scene origin, and

the scenario might not look as expected. To add a road anchor to a scene, use the **Road Anchor Tool** in RoadRunner.

Simulate the scenario. The cut-in takes place in the new scene.

```
CmdRoadRunnerApi "SimulateScenario(pacing.value='0.5')" --serverAddress=localhost:54321
```


Export Scenario from Multiple Scenes

To export a scenario from all scenes in the current project, you can write a script that calls the `CmdRoadRunnerApi` command in a loop.

Copy the script for your platform to a file named `reuse_scenario.bat` (Windows) or `reuse_scenario.bash` (Linux) and modify these variables:

- `RRPATH` — Update to the `bin/platform` folder path of your local RoadRunner installation (`win64` for Windows or `glnxa64` for Linux).
- `PROJECT` — Update to the path of your RoadRunner project.
- `SCENARIONAME` — Update to the path for your scenario.
- `PORT` — Update to the IP network port that you want to connect to.

Note To use this script, all scenes in the project must include a road anchor with the same name and the scenario must be attached to that road anchor.

reuse_scenario.bat (Windows)

```
@echo off
SetLocal EnableDelayedExpansion
REM Export scenario from all scenes in project.

set RRPATH=C:\Program Files\RoadRunner R2022b\bin\win64&
set PROJECT=C:\RR\MyProject&
set SCENARIONAME=TrajectoryCutIn&
set PORT=54321&

REM Open RoadRunner.
cd %RRPATH%
Start "" AppRoadRunner --projectPath="%PROJECT%" --apiPort=%PORT%
timeout /t 2 /nobreak>nul& REM Wait for API server to start.

REM Load scenes from "Scenes" folder and export to "Exports" folder.
for %%F in (%PROJECT%\Scenes\*.rrscene) do (
  set SCENENAME=%%~nF
  CmdRoadRunnerApi "LoadScene(file_path='%%F')" --serverAddress=localhost:!PORT!
  CmdRoadRunnerApi "LoadScenario(file_path='!SCENARIONAME!' keep_current_scene='true')" --serverAddress=localhost:!PORT!
  set EXPORTPATH=!PROJECT!\Exports\!SCENENAME!_!SCENARIONAME!.xosc
  CmdRoadRunnerApi "Export(file_path='!EXPORTPATH!' format_name='OpenSCENARIO')" --serverAddress=localhost:!PORT!
)

REM Exit RoadRunner
CmdRoadRunnerApi "Exit()" --serverAddress=localhost:!PORT!
```

reuse_scenario.bash (Linux)

```
#!/bin/bash
# Export scenario from all scenes in project.

RRPATH="/usr/local/RoadRunner_R2022b/bin/glnxa64"
PROJECT="C:/RR/MyProject"
SCENARIONAME=TrajectoryCutIn
PORT=54321

# Open RoadRunner.
cd "$RRPATH"
./AppRoadRunner --projectPath="$PROJECT" --apiPort="$PORT" &
sleep 2 # Wait for API server to start

# Load scenes from "Scenes" folder and export to "Exports" folder.
for SCENE in $PROJECT/Scenes/*.rrscene
do
  SCENENAME=$(basename $SCENE .rrscene)
  ./CmdRoadRunnerApi "LoadScene(file_path='$SCENENAME')" --serverAddress=localhost:$PORT
  ./CmdRoadRunnerApi "LoadScenario(file_path='$SCENARIONAME' keep_current_scene='true')" --serverAddress=localhost:$PORT
  EXPORTPATH="$PROJECT"/Exports/"$SCENENAME"_"$SCENARIONAME".xosc
  ./CmdRoadRunnerApi "Export(file_path='$EXPORTPATH' format_name='OpenSCENARIO')" --serverAddress=localhost:$PORT
done

# Exit RoadRunner.
./CmdRoadRunnerApi "Exit()" --serverAddress=localhost:$PORT
```

This script performs these actions:

1 Opens RoadRunner to a project and starts the RoadRunner API server.
2 Loads each scene from the Scenes folder of the project.
3 Loads the scenario into each scene by setting the keep_current_scene option to true.
4 Exports each scene and scenario combination to ASAM OpenSCENARIO 1.0.
5 Exits RoadRunner and shuts down the RoadRunner server.

Call this script from the command line. For example, if you used the .bat script, then call this command from a Windows command prompt.

Note If you receive `Permission Denied` errors when running this script on Windows, you might need to run this script as an administrator. Right-click the script in File Explorer and select **Run as administrator**.

```
reuse_scenario
```

To verify that the script exported the scenarios, navigate to the `Exports` folder of the specified project and list the contents of the folder. For example, if you specified a project at path `"C:\RR\MyProject"`, run these commands.

```
cd "C:\RR\MyProject\Exports"
dir
```

The folder lists all the exported scenarios and associated files. For example, if you exported the `TrajectoryCutIn` scenario with the `SanAntonio` and `ScenarioBasic` scenes, then your `Exports` folder contains these files.

```
SanAntonio_TrajectoryCutIn.geojson
SanAntonio_TrajectoryCutIn.osgb
SanAntonio_TrajectoryCutIn.xodr
SanAntonio_TrajectoryCutIn.xosc
ScenarioBasic_TrajectoryCutIn.geojson
ScenarioBasic_TrajectoryCutIn.osgb
ScenarioBasic_TrajectoryCutIn.xodr
ScenarioBasic_TrajectoryCutIn.xosc
```

Extend Scenario Reuse Options

To customize the script further, you can specify nondefault export settings. For more details on these settings, see the `Export` RPC method. You can also relocate a scenario within a scene programmatically by creating a variable for the **Distance** attribute of the road anchor and varying it programmatically. For more details, see "Relocate Scenarios" on page 3-117.

For additional flexibility in using the API, you can also compile the protocol buffer (protobuf) files that define the API into your desired programming language. You can then write client applications in those languages. For more details on compiling protobuf files and writing clients, see "Compile Protocol Buffers for RoadRunner gRPC API".

See Also
AppRoadRunner | LoadScene | LoadScenario | SimulateScenario | Export

Related Examples
- "Control RoadRunner Programmatically Using gRPC API"
- "Generate Scenario Variations Using gRPC API" on page 4-2
- "Export Multiple Scenarios Using gRPC API" on page 4-20

Export Multiple Scenarios Using gRPC API

This example shows how to bulk-export scenarios in a RoadRunner project to one of the file formats supported by RoadRunner. In this example, you export scenes to the ASAM OpenSCENARIO 1.0 file format using command-line operations.

How the RoadRunner gRPC API Works

RoadRunner provides an API service for importing and exporting scenes and scenarios programmatically. This API is built using the open-source, language-neutral gRPC framework, which enables you to make remote procedure calls (RPCs) to the RoadRunner server to control the application programmatically. For more background, see "Control RoadRunner Programmatically Using gRPC API".

You can compile the RoadRunner API service into any programming language supported by gRPC and write clients to remotely control RoadRunner in that language. RoadRunner also provides a precompiled version of this API as a command-line tool. This example uses this precompiled helper command to perform these operations.

Note This example primarily uses Windows commands and file paths to call the API, but this API also works on Linux.

Open RoadRunner and Start API Server

To use the command-line API to control RoadRunner remotely, you must first start the API server on an IP network port. Start the API server by opening a RoadRunner project, which you can do programmatically by using the `AppRoadRunner` executable file.

From the command line, navigate to the folder that contains the `AppRoadRunner` executable file. For example, this code shows the default installation location of the executable file on Windows:

```
cd "C:\Program Files\RoadRunner R2022b\bin\win64"
```

This code shows the default installation location of the executable file on Linux:

```
cd /usr/local/RoadRunner_R2022b/bin/glnxa64
```

Open RoadRunner to an existing project and set the port over which to run the server. Make these updates to the code:

- Replace the `projectPath` option value, `C:\RR\MyProject`, with a path to a valid RoadRunner project on your system. If you do not have an existing project, open RoadRunner and create one interactively. See "RoadRunner Project and Scene System".

- (Optional) Replace the `apiPort` option value, `54321`, with an IP network port number of your choice, between `1024` and `65535`, inclusive. If you omit the `apiPort` option, then RoadRunner opens to its default port of `35707`.

```
AppRoadRunner --projectPath="C:\RR\MyProject" --apiPort=54321
```

RoadRunner opens to a new scene in the specified project.

The **Output** pane displays the port on which the RoadRunner API server is running.

> Started RoadRunner API server on port 54321.

Switch to scenario editing mode. In the top-right corner of RoadRunner, select **Scene Editing**, then **Scenario Editing**. The **Output** pane displays an additional message indicating that the Scenario API server is running on its default port. This server is for cosimulating scenarios with MATLAB and Simulink, or with external simulators such as CARLA. The commands used in this example do not communicate with this server.

> Started RoadRunner API server on port 54321.
> Started Scenario API server on port 35706.

Export Single Scenario

Preview the programmatic operations by exporting one scene from the current project to ASAM OpenSCENARIO 1.0. To perform these operations, use the `CmdRoadRunnerApi` helper command. This helper command is located in the same folder as the `AppRoadRunner` executable file.

Load the `TrajectoryCutIn` scenario by using the `LoadScenario` RPC method. This scene is included by default in the `Scenarios` folder of RoadRunner projects. In the `serverAddress` option, replace the port number with the `apiPort` value you specified when opening RoadRunner. If you used the default port, omit the `serverAddress` option.

```
CmdRoadRunnerApi "LoadScenario(file_path='TrajectoryCutIn')" --serverAddress=localhost:54321
```

Export Multiple Scenarios

To export all scenarios in the current project, you can write a script that calls the `CmdRoadRunnerApi` command in a loop.

Copy the script for your platform to a file named `bulk_openscenario_export.bat` (Windows) or `bulk_openscenario_export.bash` (Linux) and modify these variables:

- `RRPATH` — Update to the `bin/`*`platform`* folder path of your local RoadRunner installation (`win64` for Windows or `glnxa64` for Linux).
- `PROJECT` — Update to the path of your RoadRunner project.

- PORT — Update to the IP network port that you want to connect to.

---

bulk_openscenario_export.bat (Windows)

```
@echo off
SetLocal EnableDelayedExpansion
REM Export all RoadRunner scenes in project to ASAM OpenDRIVE.

set RRPATH=C:\Program Files\RoadRunner R2022b\bin\win64&
set PROJECT=C:\RR\MyProject&
set PORT=54321&

REM Open RoadRunner.
cd %RRPATH%
Start "" AppRoadRunner --projectPath="%PROJECT%" --apiPort=%PORT%
timeout /t 1 /nobreak>nul& REM Wait for API server to start.

REM Load scenarios from "Scenarios" folder and export to "Exports" folder.
for %%F in (%PROJECT%\Scenarios\*.rrscenario) do (
  set SCENARIONAME=%%~nF
  set EXPORTPATH=!PROJECT!\Exports\!SCENARIONAME!.xodr
  CmdRoadRunnerApi "LoadScenario(file_path='%%F')" --serverAddress=localhost:!PORT!
  CmdRoadRunnerApi "Export(file_path='!EXPORTPATH!' format_name='OpenSCENARIO')" --serverAddress=localhost:!PORT!
)

REM Exit RoadRunner
CmdRoadRunnerApi "Exit()" --serverAddress=localhost:!PORT!
```

---

bulk_openscenario_export.bash (Linux)

```
#!/bin/bash
# Export all RoadRunner scenes in project to ASAM OpenDRIVE.

RRPATH="/usr/local/RoadRunner_R2022b/bin/glnxa64"
PROJECT="/local/RR/MyProject"
PORT=54321

# Open RoadRunner.
cd "$RRPATH"
./AppRoadRunner --projectPath="$PROJECT" --apiPort="$PORT" &
sleep 2 # Wait for API server to start

# Load scenes from "Scenarios" folder and export to "Exports" folder.
for SCENARIO in $PROJECT/Scenarios/*.rrscenario
do
  SCENARIONAME=$(basename $SCENARIO .rrscenario)
  EXPORTPATH=$PROJECT/Exports/$SCENARIONAME.xodr
  ./CmdRoadRunnerApi "LoadScenario(file_path='$SCENARIONAME')" --serverAddress=localhost:$PORT
  ./CmdRoadRunnerApi "Export(file_path='$EXPORTPATH' format_name='OpenSCENARIO')" --serverAddress=localhost:$PORT
done

# Exit RoadRunner.
./CmdRoadRunnerApi "Exit()" --serverAddress=localhost:$PORT
```

This script performs these actions:

1 Opens RoadRunner to a project and starts the RoadRunner API server.

2 Loads each scenario from the Scenarios folder of the project, along with the scene that the scenario was last saved with.

3 Exports each scenario to ASAM OpenSCENARIO using the default settings. RoadRunner exports each scenario file to the Exports folder of the project and gives it the same name as its associated scene, but with the extension .xosc. The associated OpenSceneGraph, ASAM OpenDRIVE, and GeoJSON files are exported for each scenario as well.

4 Exits RoadRunner and shuts down the RoadRunner server.

Call this script from the command line. For example, if you used the `.bat` script, then call this command from a Windows command prompt.

```
bulk_openscenario_export
```

To verify that the script exported the scenarios, navigate to the `Exports` folder of the specified project and list the contents of the folder. For example, if you specified a project at path `"C:\RR\MyProject"`, run these commands.

```
cd "C:\RR\MyProject\Exports"
dir
```

The folder lists all the exported scenarios and associated files. For example, if your project includes the `TrajectoryCutIn` scenario, then your `Exports` folder contains these files.

```
...
TrajectoryCutIn.geojson
TrajectoryCutIn.osgb
TrajectoryCutIn.xodr
TrajectoryCutIn.xosc
...
```

Extend RoadRunner Export Options

To customize the script further, you can specify nondefault export settings. For more details on these settings, see the `Export` RPC method.

If the scenarios in your project all use the same scene, you can speed up export by exporting the scene with the first scenario export only. After you export the first scene, set the `path_to_existing_scene_graph` option in the `open_scenario_settings` field of the `Export` method. RoadRunner reuses this scene for all subsequent exports. For an example of how to use the `path_to_existing_scene_graph` option, see "Generate Scenario Variations Using gRPC API" on page 4-2.

For additional flexibility in exporting scenarios, consider compiling the protocol buffer (protobuf) files that define the API into your desired programming language. For more details, see "Compile Protocol Buffers for RoadRunner gRPC API". You can then write client applications in those languages. For sample clients, see these examples:

- "Create gRPC Python Client for Controlling RoadRunner Programmatically"
- "Create gRPC C++ Client for Controlling RoadRunner Programmatically"

See Also
AppRoadRunner | Export | Import | export_settings.proto | import_settings.proto

Related Examples
- "Control RoadRunner Programmatically Using gRPC API"
- "Export to ASAM OpenSCENARIO" on page 5-2

Simulate a RoadRunner Scenario Using MATLAB Functions

This example shows how to run and visualize scenarios in RoadRunner Scenario using MATLAB functions. You can use MATLAB functions to control RoadRunner Scenario programmatically. Common programmatic tasks that you can perform include:

- Open and close the RoadRunner Scenario application.
- Open, close, and save scenes, scenarios, and projects.
- Import and export scenarios.

RoadRunner Scenario enables you to interactively design and simulate agents in scenarios. To verify the behavior of these agents, it is often helpful to automate the process of running and analyzing the results of scenario simulations. In this example, you learn how to use Automated Driving Toolbox® to launch RoadRunner Scenario, configure and run a simulation, and then plot simulation results.

To run this example, you must:

- Have an Automated Driving Toolbox® license.
- Have a RoadRunner license and the product is installed.
- Have a RoadRunner Scenario license and the product is installed.
- Have created a RoadRunner project folder.

Set Up Environment to Launch RoadRunner Scenario

To use MATLAB functions to control RoadRunner Scenario programmatically, use the `roadrunner` object. By default, `roadrunner` opens RoadRunner from the default installation folder for the platform you are using (either Windows® or Linux®). These are the default installation locations by platform:

- Windows – C:\Program Files\RoadRunner *R20NNx*\bin\win64
- Linux, Ubuntu® – /usr/local/RoadRunner_*R20NN*x/bin/glnxa64

R20NNx is the MATLAB release you are using.

If your RoadRunner Scenario installation is at a different location than the default location, use the MATLAB `settings` API to customize the default value of the RoadRunner Scenario installation folder.

Open RoadRunner Scenario Session

You can use the `roadrunner` function to create a `roadrunner` object and launch a RoadRunner Scenario session. The `roadrunner` function requires an argument that specifies the location of a RoadRunner project. A RoadRunner project folder typically contains these subfolders: `Assets`, `Exports`, `Project`, `Scenarios`, and `Scenes`.

Open a project in RoadRunner using the `roadrunner` function by specifying the location in which to create a project. This example assumes that RoadRunner is installed in its default location in Windows.

Specify the path to an existing project. For example, this code shows the path to a project located on `C:\RR\MyProject`. The function returns a `roadrunner` object, `rrApp,` that provides functions for performing basic workflow tasks such as opening, closing, and saving scenes and projects.

```
rrProj = "C:\RR\MyProject";
rrApp = roadrunner(rrProj,InstallationFolder="C:\Program Files\RoadRunner R2022b\bin\win64");
```

Open an existing scenario in RoadRunner Scenario by using the `openScenario` function and specifying the `rrApp` object and the specific scenario `filename` that you want to open. For example, open the `TrajectoryCutIn` scenario file, which is a scenario included by default in RoadRunner projects. This function opens the desired scenario in the RoadRunner Scenario application through MATLAB.

```
openScenario(rrApp,"TrajectoryCutIn.rrscenario");
```

Simulate Scenario

Once the scenario is loaded into RoadRunner Scenario, automate the simulation tasks by using the `createSimulation` function to create a simulation object. The simulation object enables you to programatically interact with the scenario simulation.

Specify the `rrApp` object as an input argument to the `createSimulation` function. The function creates a simulation object, `rrSim`.

```
rrSim = createSimulation(rrApp);
```

```
Connection status: 1
Connected to RoadRunner Scenario server on localhost:59151, with client id {1c6fd5fc-736d-4cd9-9c
```

Set a maximum simulation time of 10 seconds. Use the `set` function and specify the `rrSim` object, name of the variable to set, and the value for that variable as input arguments.

```
maxSimulationTimeSec = 10;
set(rrSim,'MaxSimulationTime',maxSimulationTimeSec);
```

Enable simulation result logging so that you can plot the results later.

```
set(rrSim,"Logging","on");
```

Start the simulation. Use a `while` loop to monitor the status of the simulation, and wait for the simulation to complete.

```
set(rrSim,"SimulationCommand","Start");
while strcmp(get(rrSim,"SimulationStatus"),"Running")
    pause(1);
end
```

Plot Agent Velocities

In this section, you retrieve the logged velocities of the actors from the simulation and plot their magnitudes against simulation time.

Get the logged results from the scenario. Use the `get` function and specify the `rrSim` object and `"SimulationLog"` as input arguments. The function returns the simulation logs in `rrLog`, which contains information about the simulation of the scenario.

```
rrLog = get(rrSim,"SimulationLog");
```

The `TrajectoryCutIn scenario` contains two actors. The red sedan has `Actor ID` set to 1, and the white sedan has `Actor ID` set to 2. Get the logged velocities of these actors from simulation log. Also, get the corresponding simulation times from the simulation logs.

```
velocityAgent1 = get(rrLog,'Velocity','ActorID',1);
velocityAgent2 = get(rrLog,'Velocity','ActorID',2);
time = [velocityAgent1.Time];
```

The function returns the velocities of the red sedan and the white sedan as vectors and stores them in the `velMagAgent1` and `velMagAgent2` variables, respectively. Calculate the magnitude of the velocity for each actor by using the `norm` function.

```
velMagAgent1 = arrayfun(@(x) norm(x.Velocity,2),velocityAgent1);
velMagAgent2 = arrayfun(@(x) norm(x.Velocity,2),velocityAgent2);
```

Plot the agent velocities with respect to simulation time using the `plot` function. Label the graph and the *x* and *y* axes.

```
figure
hold on
plot(time,velMagAgent1,"r")
plot(time,velMagAgent2,"b")
grid on
title("Agent Velocities from RoadRunner Scenario")
ylabel("Velocity (m/sec)")
xlabel("Time (sec)")
legend("Actor ID = 1","Actor ID = 2")
```
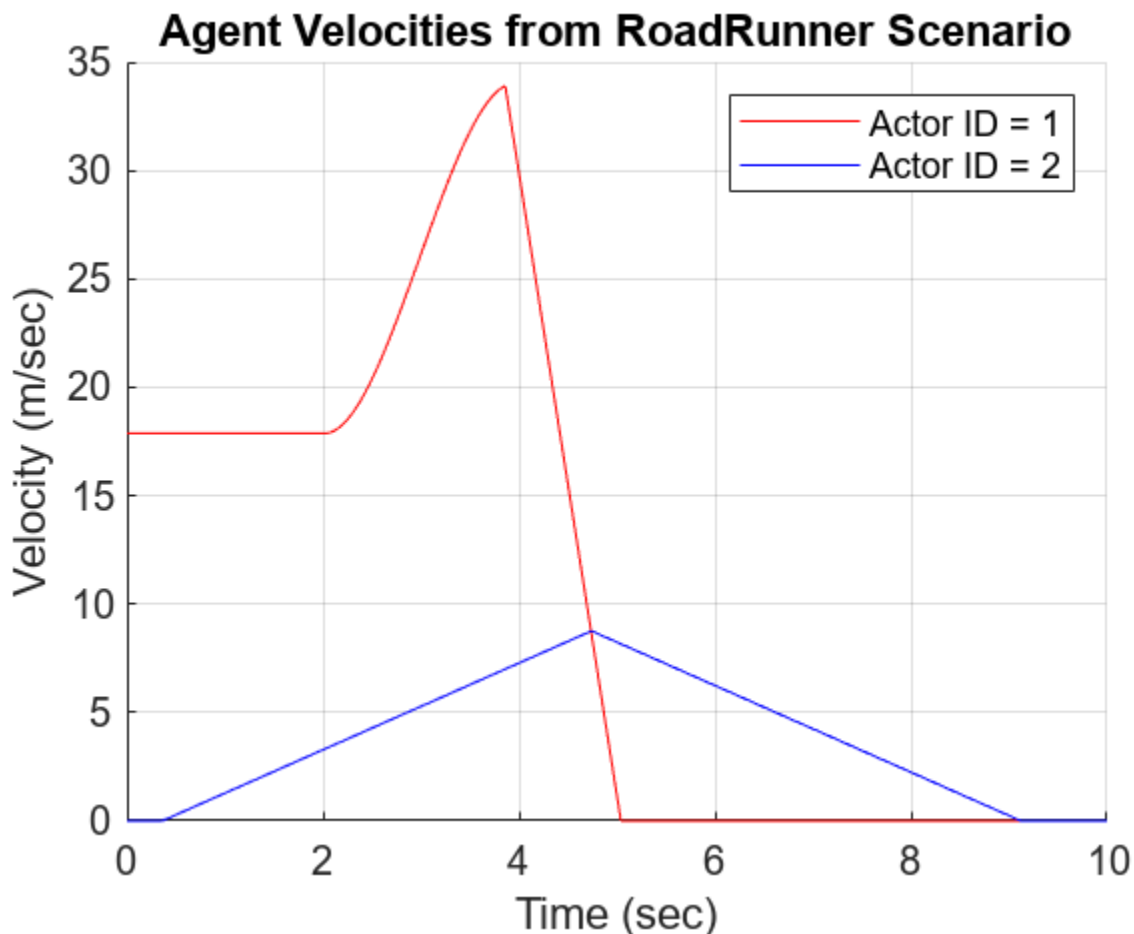
Notice that the velocities of the actors correspond to their specifications in the **Logic Editor** of RoadRunner Scenario.

Plot Agent Velocities

Plot the lanes from the RoadRunner scene and overlay the positions of vehicles on the map.

Get the HD Map specification from RoadRunner by using the `getMap` function. Notice that the function returns a structure and one of the fields contains information about the lanes.

```
hdMap = getMap(rrSim);
lanes = hdMap.map.lanes;
```

Loop through each of the lane specifications using a `for` loop and plot the lane coordinates.

```
figure
hold on
for i = 1:numel(lanes)
    control_points = lanes(i).geometry.values;
    x_coordinates = arrayfun(@(cp) cp.x,control_points);
    y_coordinates = arrayfun(@(cp) cp.y,control_points);
    plot(x_coordinates, y_coordinates, 'black');
end
axis equal
```
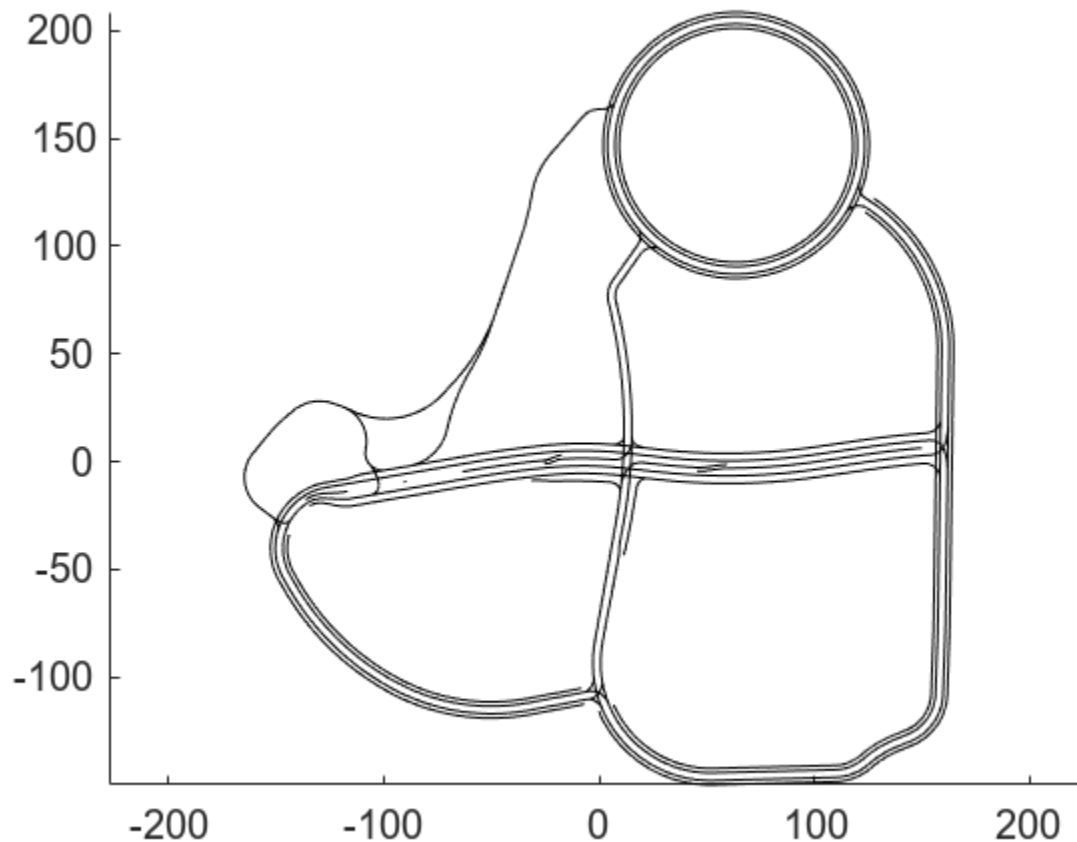
Extract the positions of the vehicles and plot them on the lanes.

```
poseActor1 = rrLog.get('Pose','ActorID',1);
positionActor1_x = arrayfun(@(x) x.Pose(1,4),poseActor1);
positionActor1_y = arrayfun(@(x) x.Pose(2,4),poseActor1);
plot(positionActor1_x,positionActor1_y,"r","LineWidth",2)

poseActor2 = rrLog.get('Pose','ActorID',2);
positionActor2_x = arrayfun(@(x) x.Pose(1,4),poseActor2);
positionActor2_y = arrayfun(@(x) x.Pose(2,4),poseActor2);
plot(positionActor2_x,positionActor2_y,"b","LineWidth",2)

title("Agent Positions from RoadRunner Scenario")
ylabel("Y (m)")
xlabel("X (m)")
```
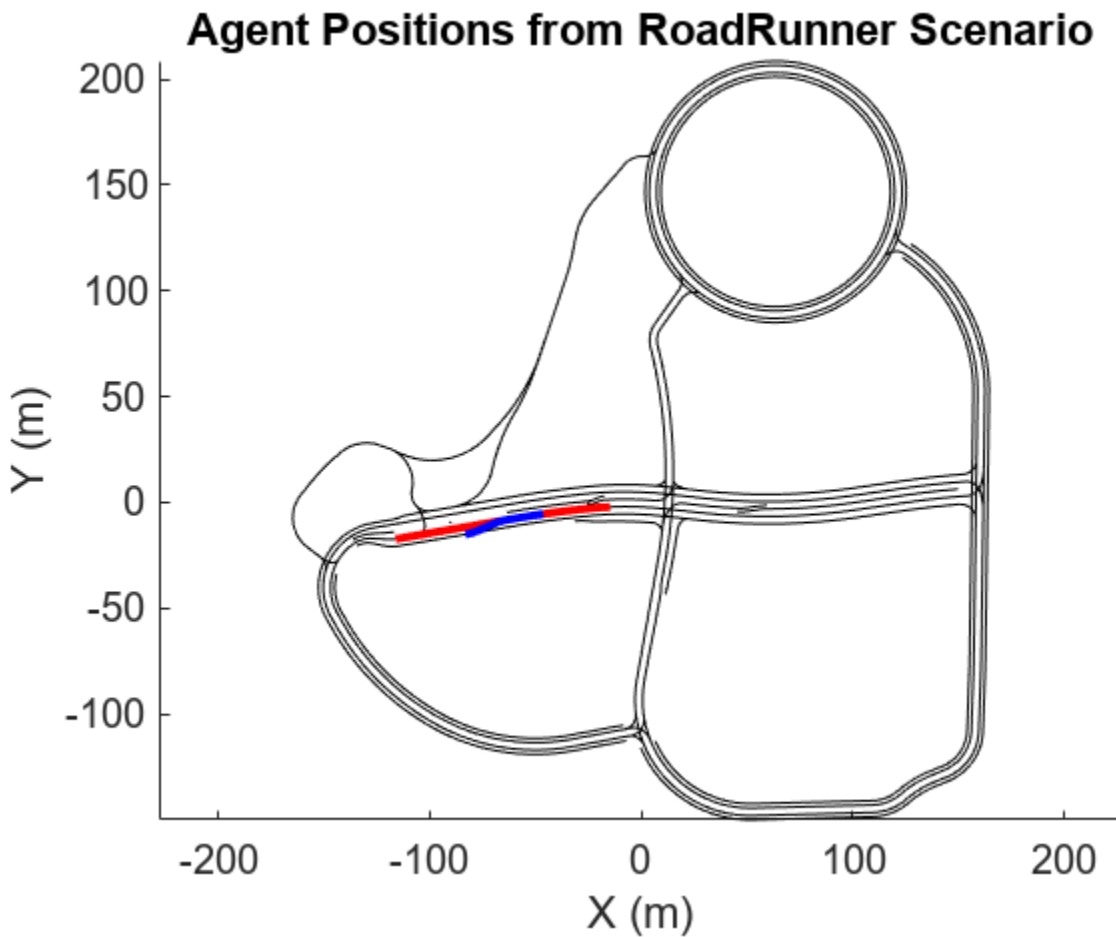


```
figure(gcf)
```

Close Scenario Session

To stop interacting with RoadRunner Scenario, close the simulation. Then close the application.

```
close(rrApp)
```

Close the open figure.

```
close all
```

Further Exploration

In this example you learned about the basic capabilities of connecting to RoadRunner Scenario programmatically using MATLAB. To extend this script further, you can:

- Vary the scenario and vehicle actors in the scenario.
- Develop MATLAB and Simulink behaviors, publish actor behavior, simulate behaviors in RoadRunner Scenario simulation, and control simulations and access simulation parameters.

See Also

roadrunner | openScenario | createSimulation | Simulink.ScenarioLog | set | close | settings

Related Examples

- "Design and Simulate Scenarios" on page 1-4
- "Overview of Simulating RoadRunner Scenarios with MATLAB and Simulink" (Automated Driving Toolbox)

5

Export Scenarios

Export to ASAM OpenSCENARIO

ASAM OpenSCENARIO is an open file format that describes the dynamic content for an automated driving simulation. Using RoadRunner Scenario, you can export scenarios to ASAM OpenSCENARIO file versions 1.0, 1.1, and 2.0.

Export Interactively

To export scenarios to ASAM OpenSCENARIO 1.x by using the RoadRunner Scenario user interface, follow these steps:

1 From the **Tools** menu, select **Scenario Editing** to switch to scenario editing mode.

2 From the **File** menu, select **Export**, then **ASAM OpenSCENARIO 1.x (.xosc + .xodr + .osgb)**.

3 In the Export ASAM OpenSCENARIO dialog box, set the **File path** option to the path where you want to export the generated `.xosc` file.

4 On the **OpenSCENARIO** tab, specify the **Version** for the output file as `OpenSCENARIO 1.0` or `OpenSCENARIO 1.1`.

   Default: `OpenSCENARIO 1.1`

5 (Optional) On the **ASAM OpenDRIVE** tab, specify options to customize the exported ASAM OpenDRIVE (`.xodr`) file. This file and the associated `.geojson` file describe the road network used in the scenario. For details on these options, see "Export to ASAM OpenDRIVE".

6 (Optional) On the **OpenSceneGraph** tab, specify options to customize the OpenSceneGraph (`.osgb`) file. This file describes the scene used in the scenario. For details on these options, see "Export to OpenSceneGraph".

7 Click **Export**.

To export scenarios to ASAM OpenSCENARIO 2.0 by using the RoadRunner Scenario user interface, follow these steps:

1 From the **Tools** menu, select **Scenario Editing** to switch to scenario editing mode.

2 From the **File** menu, select **Export**, then **ASAM OpenSCENARIO 2.0 (.osc + .xodr)**.

3 In the Export ASAM OpenSCENARIO 2.0 dialog box, set the **File path** option to the path where you want to export the generated `.osc` file.

4 (Optional) Specify options to customize the exported ASAM OpenDRIVE (`.xodr`) file. This file describe the road network used in the scenario. For details on these options, see "Export to ASAM OpenDRIVE".

5 Click **Export**.

RoadRunner Scenario exports these files for the scenario:

ASAM OpenSCENARIO File Version	Exported Files
ASAM OpenSCENARIO 1.x	• ASAM OpenSCENARIO file (`.xosc`) — XML data that describes the scenario logic and actors • ASAM OpenDRIVE file (`.xodr`) — XML data that describes the road network • OpenSceneGraph file (`.osgb`) — Binary 3D format file that describes the scene • GeoJSON file (`.geojson`) — JSON data that describes the geographic location of the ASAM OpenDRIVE data
ASAM OpenSCENARIO 2.0	• ASAM OpenSCENARIO file (`.osc`) — Domain specific language (DSL) format that describes the scenario logic and actors • ASAM OpenDRIVE file (`.xodr`) — XML data that describes the road network • GeoJSON file (`.geojson`) — JSON data that describes the geographic location of the ASAM OpenDRIVE data

Export Programmatically

To export an ASAM OpenSCENARIO file programmatically, use the `Export` remote procedure call (RPC) method. Using this method and other RPC methods, you can, for example, export all scenarios in a project to ASAM OpenSCENARIO 1.x or 2.0 file formats. If all scenarios use the same scene, you can also specify an option to export the OpenSceneGraph file for the scene a single time rather than export this large file with each scenario. This option can significantly reduce export times.

For an example that shows how to export scenarios to ASAM OpenSCENARIO programmatically, see "Generate Scenario Variations Using gRPC API" on page 4-2.

Visualize Exported Scenario

Using exported ASAM OpenSCENARIO 1.x files, you can view the scenario in an external simulator, such as esmini (version 2.4 and higher are supported).

This code shows a sample call to view an exported scenario on a local Windows installation of esmini.

```
cd "C:\Program Files\esmini\bin"
esmini --window 60 60 800 400 --osc "C:\RR\MyProject\Exports\CutIn.xosc"
```
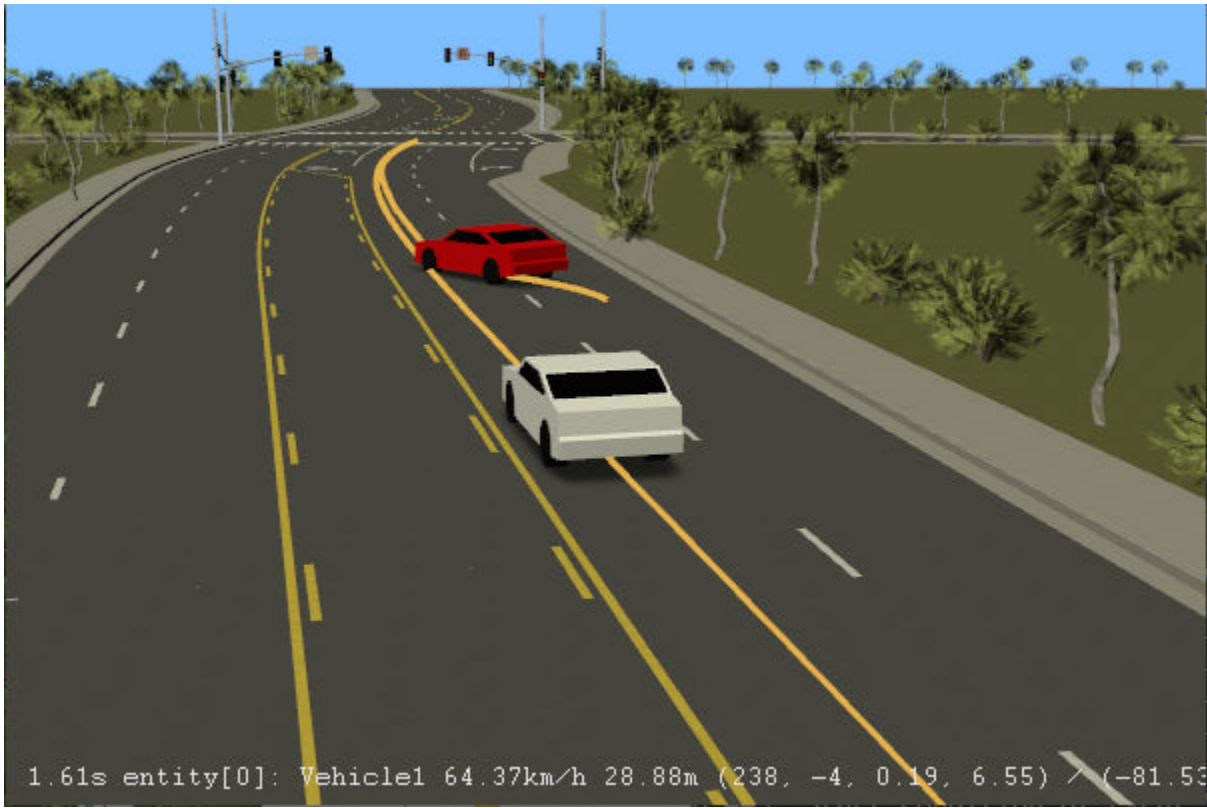
ASAM OpenSCENARIO 1.x Representations

This section describes how vehicle properties and dynamic logic specified using various actions and conditions in RoadRunner Scenario are represented in the ASAM OpenSCENARIO 1.x format.

Properties of Actors

RoadRunner Scenario exports the properties of vehicles in the scenario using a `Vehicle` element within a `ScenarioObject` element. Vehicle properties are derived from the corresponding vehicle asset in the **Library Browser**. For more information on vehicle assets and their parameters, see Vehicle Assets.

Example

```
<ScenarioObject name="CompactCar">
    <Vehicle name="CompactCar" vehicleCategory="car" mass="1500" model3d="Vehicles/CompactCar.fbx">
        <ParameterDeclarations/>
        <BoundingBox>
            <Center x="0" y="0" z="0.800383"/>
            <Dimensions height="1.22452" length="3.65698" width="1.86693"/>
        </BoundingBox>
        <Performance maxAcceleration="5" maxDeceleration="5" maxSpeed="65"/>
        <Axles>
            <FrontAxle maxSteering="0.698132" positionX="1.17035" positionZ="0.315215" trackWidth="1.19184" wheelDiamete
            <RearAxle maxSteering="0" positionX="-1.2491" positionZ="0.315215" trackWidth="1.19184" wheelDiameter="0.630
        </Axles>
        <Properties/>
    </Vehicle>
</ScenarioObject>
```

RoadRunner Scenario exports the properties of pedestrians using a `Pedestrian` element within a `ScenarioObject` element. Pedestrian properties are derived from the corresponding character asset in the **Library Browser**. For more information on character assets and their parameters, see Character Assets.

Example

```
<ScenarioObject name="Citizen_Male">
    <Pedestrian name="Citizen_Male" mass="6.2000000000000000e+1" model3d="Characters/Citizen_Male.rrchar" pedestrianCate
        <ParameterDeclarations/>
        <BoundingBox>
            <Center x="0.0027168" y="-0.0681267" z="0.872004"/>
            <Dimensions height="1.74811" length="0.391623" width="0.626842"/>
        </BoundingBox>
        <Properties/>
    </Pedestrian>
</ScenarioObject>
```

Vehicle Trajectory

RoadRunner Scenario exports the trajectory of a vehicle to the `Trajectory` element of the `FollowTrajectoryAction` element in the `Init` section of the output ASAM OpenSCENARIO file. The `Trajectory` element defines the motion of the associated vehicle in the world position format using a polyline. Setting `TrajectoryFollowingMode` to `position` forces the vehicle to strictly adhere to the specified trajectory.

If you specify **Timing Data** for waypoints in the scenario, RoadRunner Scenario exports absolute time domain values of trajectory waypoints, without any scaling or offset, by using the `Timing` property of the `TimeReference` element. Otherwise, RoadRunner Scenario exports the trajectory using an empty `TimeReference` element, and you should ignore the time information contained in the `Trajectory` element.

Example

```
<RoutingAction>
  <FollowTrajectoryAction>
    <Trajectory name="Path_CompactCar" closed="false">
      <Shape>
        <Polyline>
          <Vertex time="0.0000000000000000e+0">
            <Position>
              <WorldPosition x="-1.3816883511982547e+2" y="-5.2470924367182683e+1" z="0.0000000000000000e+0" h="0.000000
            </Position>
          </Vertex>
          <Vertex time="1.0000000000000000e+0">
            <Position>
              <WorldPosition x="-1.3246930791258063e+2" y="-5.0256077000744838e+1" z="0.0000000000000000e+0" h="0.000000
            </Position>
          </Vertex>
        </Polyline>
      </Shape>
    </Trajectory>
      <TimeReference/>
      <TrajectoryFollowingMode followingMode="position"/>
  </FollowTrajectoryAction>
</RoutingAction>
```

Dynamic Logic

RoadRunner Scenario exports all action phases of all vehicles using a single `Story` element and one or more `Act` elements, such that the exported file preserves the original intent and simulation behavior of the scenario.

If you specify end condition for an action phase, RoadRunner Scenario exports the action phase using a distinct `Act` element. Otherwise, RoadRunner Scenario attempts to export action phases of a scenario using the minimum number of `Act` elements.

ASAM OpenSCENARIO v1.x file format specifies that an `Act` can end when all nested `ManeuverGroup` instances are completed. This is true even if the stop trigger of the `Act` is not yet satisfied. RoadRunner Scenario treats end conditions differently, keeping the action running until the end condition is satisfied. Therefore, when you specify an end condition for an action phase, RoadRunner Scenario exports the action using a distinct `Act` instance, which has a dummy `ManeuverGroup` instance to force the act to continue running until the specified end condition is satisfied. This `ManeuverGroup` contains `ForceActToExecuteUntilStopTrigger` event that never triggers to prevent the act from ending before the stop trigger is satisfied.

Example

```
<ManeuverGroup name="Act_GlobalGroup" maximumExecutionCount="1">
    <Actors selectTriggeringEntities="false"/>
    <Maneuver name="Act_GlobalGroup_Maneuver">
        <Event name="ForceActToExecuteUntilStopTrigger" priority="parallel">
            <Action name="NeverExecutedAction">
                <UserDefinedAction>
                    <CustomCommandAction type="MW_WaitAction"></CustomCommandAction>
                </UserDefinedAction>
            </Action>
            <StartTrigger>
                <ConditionGroup>
                    <Condition name="NeverSatisfiedCondition" conditionEdge="none" delay="0.0000000000000000e+0">
                        <ByValueCondition>
                            <StoryboardElementStateCondition storyboardElementType="story" storyboardElementRef="Story"
                        </ByValueCondition>
                    </Condition>
                </ConditionGroup>
            </StartTrigger>
        </Event>
    </Maneuver>
</ManeuverGroup>
```

To translate sequential logic (serial phases) of the RoadRunner scenario to ASAM OpenSCENARIO v1.x file format, RoadRunner uses `StoryboardElementStateCondition` elements for triggering the events in the specified sequential order.

Tip To understand the structure of the exported data, run the transformation for ASAM OpenSCENARIO v1.x export. To run the transformation, select **Tools > Debug > Run OSCv1 Transform**.

Change Speed Action

RoadRunner Scenario exports `Change Speed` actions using a `SpeedAction` element.

When you set the **Relative to** parameter to `Actor`, the specified **Speed Offset** value is exported as a `RelativeTargetSpeed` element with a `speedTargetValueType` value of `delta`. The **Direction** parameter specifies the sign of the speed target value in the exported file.

Direction Parameter	Speed Target Value in Exported File
`Faster than`	Positive value specified using **Speed Offset** parameter

Direction Parameter	Speed Target Value in Exported File
`Slower than`	Negative value specified using **Speed Offset** parameter
`Same speed as`	`0`

In the exported file, the `continuous` property of the `RelativeTargetSpeed` element is set to `true` if you set the **Speed Sampling** parameter to `Continuous`. Otherwise the `continuous` property is set to `false`.

Example

```
<SpeedAction>
  <SpeedActionDynamics dynamicsShape="cubic" dynamicsDimension="rate" value="1.0000000000000000e+1"/>
  <SpeedActionTarget>
    <RelativeTargetSpeed entityRef="Ambulance" value="-20" speedTargetValueType="delta" continuous="true"/>
  </SpeedActionTarget>
</SpeedAction>
```

Change Lane Action

RoadRunner Scenario exports `Change Lane` actions using a `LaneChangeAction` element. RoadRunner Scenario supports only the `RelativeTargetLane` type of `LaneChangeTarget` element. `AbsoluteTargetLane` is not supported.

RoadRunner Scenario exports the sign of the specified **Lane Offset** parameter as shown in this table.

Direction Parameter	Sign of Lane Offset Value in Exported File
`To the left`	Positive
`To the right`	Negative

Note The `targetLaneOffset` attribute of ASAM OpenSCENARIO 1.x is not supported. RoadRunner Scenario assumes that a vehicle travels along the center line of a lane after performing a `Change Lane` action.

Example

```
<LaneChangeAction>
  <LaneChangeActionDynamics dynamicsShape="cubic" dynamicsDimension="distance" value="10"/>
  <LaneChangeTarget>
      <RelativeTargetLane entityRef="CompactCar" value="-2"/>
  </LaneChangeTarget>
</LaneChangeAction>
```

Change Lateral Offset Action

RoadRunner Scenario exports `Change Lateral Offset` actions using a `LaneOffsetAction` element. RoadRunner Scenario supports only the `AbsoluteTargetLaneOffset` type of `LaneOffsetTarget` element. `RelativeTargetLaneOffset` is not supported.

Note ASAM OpenSCENARIO 1.x does not consider the time or distance over which a `LaneOffsetAction` occurs. RoadRunner Scenario supports a time dimension to control `LaneOffsetAction`. The `maxLateralAcc` element of ASAM OpenSCENARIO 1.x is not supported.

Example

```
<LaneOffsetAction continuous="false">
  <LaneOffsetActionDynamics dynamicsShape="cubic"/>
  <LaneOffsetTarget>
      <AbsoluteTargetLaneOffset value="0.2"/>
  </LaneOffsetTarget>
</LaneOffsetAction>
```

Change Longitudinal Distance Action

RoadRunner Scenario exports `Change Longitudinal Distance` actions using a `LongitudinalDistanceAction` element. RoadRunner Scenario supports only the `entity` type of `coordinateSystem` element.

This table shows the relation between attributes of RoadRunner Scenario and the elements in the exported file.

Attribute	Element in Exported File
Relative Position	`displacement`
Reference Actor	`entityRef`
Space Distance Offset	`distance` RoadRunner Scenario exports the `distance` property with the value you specify for the **Space Distance Offset** attribute. The `timeGap` property is exported with a value of 0.
Time Distance Offset	`timeGap` RoadRunner Scenario exports the `timeGap` property with the value you specify for the **Time Distance Offset** attribute. The `distance` property is exported with a value of 0.
Measure From	`freespace` If you set the **Measure From** attribute to `Bounding boxes`, RoadRunner Scenario sets the `freespace` property to `true` in the exported file. Otherwise, it sets the `freespace` property to `false`.
Sampling Mode	`continuous` If you set the **Sampling Mode** attribute to `continuous`, RoadRunner Scenario sets the `continuous` property to `true` in the exported file. Otherwise, it sets the `continuous` property to `false`.
Dynamic Constraints	`DynamicConstraints`

Example

```
<LongitudinalDistanceAction continuous="true" coordinateSystem="entity"
  displacement="leadingReferencedEntity" distance="5.0000000000000000e+1"
  entityRef="CompactCar2" freespace="true" timeGap="0.0000000000000000e+0">
  <DynamicConstraints maxAcceleration="5.0000000000000000e+0"
    maxDeceleration="5.0000000000000000e+0" maxSpeed="6.5000000000000000e+1"/>
</LongitudinalDistanceAction>
```

Change Behavior Parameter Action

RoadRunner Scenario exports `Change Behavior Parameter` actions by using a
`ParameterAction` element to set the specified parameter to a new value.

Example

```
<GlobalAction>
    <ParameterAction parameterRef="Yellow_Headlights">
        <SetAction value="ON"/>
    </ParameterAction>
</GlobalAction>
```

Change Global Parameter Action

RoadRunner Scenario exports `Change Global Parameter` actions by using a `ParameterAction`
element to set the specified parameter to a new value.

Example

```
<GlobalAction>
    <ParameterAction parameterRef="PrecipitationDensity">
        <SetAction value=".75"/>
    </ParameterAction>
</GlobalAction>
```

User Defined Action

RoadRunner Scenario exports `User Defined` actions by using a `UserDefinedAction` element. This
example shows how RoadRunner Scenario exports the details of the `User Defined` action and the
name of the referenced actor that executes this action. In this example, the user-defined action
`CustomDrive` controls the `ThrottleLevel` and `SteeringAngle` parameters of an ambulance. For
more information on how to design a scenario with user-defined actions, see "Model Vehicle Behavior
Using User-Defined Actions in MATLAB" on page 3-50.

Example

```
<UserDefinedAction>
  <CustomCommandAction type="CustomDrive">Actor:Ambulance, ThrottleLevel:30, SteeringAngle:20</CustomCommandAction>
</UserDefinedAction>
```

Wait Action

RoadRunner Scenario exports `Wait` actions as a `UserDefinedAction` element, because ASAM
OpenSCENARIO does not specify this action. This action does not do anything until the specified end
condition is satisfied.

<table>
<tr><td>

Example

```
<UserDefinedAction>
    <CustomCommandAction type="MW_WaitAction"></CustomCommandAction>
</UserDefinedAction>
```
</td></tr>
</table>

Add Actors After Scenario Initialization

In RoadRunner Scenario, you can configure an actor to appear after scenario initialization during simulation. To do so, follow these steps:

1 In the **Logic** editor, right-click the `Initialize Speed` action phase of the desired actor, and select **Add an Action Phase After**. By default, RoadRunner Scenario adds a `Change Speed` action phase.

2 Right-click this newly added `Change Speed` action phase, and select **Set as Initial Phase**.

3 Select the `Change Speed` action phase. From the **Attributes** pane, set the **Action Type** attribute to `Wait`. By default, the `Wait` action has a `Duration` condition for 5 seconds. If you want to change the wait time, modify this value from the **Attributes** pane.

When you add an actor to the scenario after initialization, RoadRunner Scenario exports the position information of the actor using the `AddEntityAction` element of the ASAM OpenSCENARIO 1.x file format. RoadRunner Scenario supports only the `LanePosition` type for the `Position` property of the `AddEntityAction` element.

<table>
<tr><td>

Example

```
<GlobalAction>
    <EntityAction entityRef="Sedan">
        <AddEntityAction>
            <Position>
                <LanePosition roadId="215" laneId="-1" offset="-5.25206e-15" s="12.7">
                    <Orientation type="absolute" h="0.170215" p="0" r="0"/>
                </LanePosition>
            </Position>
        </AddEntityAction>
    </EntityAction>
</GlobalAction>
```
</td></tr>
</table>

Simulation Time Condition

By default, RoadRunner Scenario exports a `Simulation Time` condition to start an `Act` at a simulation time greater than `0` and end a `Story` at a simulation time greater than `60` seconds. You can also specify this condition as an end condition for any action phase in your scenario, in which case RoadRunner Scenario exports your specified `Simulation Time` condition data to trigger the next action.

<table>
<tr><td>

Example

```
<StartTrigger>
  <ConditionGroup>
    <Condition name="Start Condition of Act" conditionEdge="rising" delay="0.0000000000000000e+0">
      <ByValueCondition>
          <SimulationTimeCondition value="0.0000000000000000e+0" rule="greaterThan"/>
      </ByValueCondition>
    </Condition>
  </ConditionGroup>
</StartTrigger>
```
</td></tr>
</table>

Actor Speed Condition

Using the `Actor Speed` condition, you can check either the current absolute speed of the referenced actor or relative speed between two actors. Depending on how you set the **Relative to** attribute of the `Actor Speed` condition, RoadRunner Scenario exports the `Actor Speed` condition to one of two elements of an ASAM OpenSCENARIO 1.x file.

Value of Relative to Attribute of Actor Speed Condition	Element in Exported File
Absolute	SpeedCondition
Actor	RelativeSpeedCondition

Example

```
<Condition name="Start Condition of Event_CompactCar_1" conditionEdge="none" delay="0.0000000000000000e+0">
  <ByEntityCondition>
    <TriggeringEntities triggeringEntitiesRule="any">
      <EntityRef entityRef="CompactCar"/>
    </TriggeringEntities>
    <EntityCondition>
      <SpeedCondition value="1.5000000000000000e+1" rule="equalTo"/>
    </EntityCondition>
  </ByEntityCondition>
</Condition>
```

Distance To Point Condition

RoadRunner Scenario exports the `Distance To Point` condition using the `ReachPositionCondition` element of ASAM OpenSCENARIO 1.x. The **Threshold** attribute of the `Distance To Point` condition specifies the `tolerance` field of the `ReachPositionCondition` element. The **Rule** attribute of the `Distance To Point` condition is not exported because ASAM OpenSCENARIO does not specify a rule for `ReachPositionCondition`. RoadRunner Scenario supports the `WorldPosition` type of coordinates for specifying the `Position` attribute.

Example

```
<Condition name="Start Condition of Event_CompactCar_1" conditionEdge="none" delay="0.0000000000000000e+0">
  <ByEntityCondition>
    <TriggeringEntities triggeringEntitiesRule="any">
      <EntityRef entityRef="CompactCar"/>
    </TriggeringEntities>
    <EntityCondition>
      <ReachPositionCondition tolerance="3.2500000000000000e+0">
        <Position>
          <WorldPosition x="2.5061806803169961e-1" y="7.3375634784816427e+1" z="0.0000000000000000e+0" h="0.000000000000
        </Position>
      </ReachPositionCondition>
    </EntityCondition>
  </ByEntityCondition>
</Condition>
```

Distance To Actor Condition

RoadRunner Scenario exports the `Distance To Actor` condition using the `RelativeDistanceCondition` element of ASAM OpenSCENARIO 1.x. The **Threshold** attribute of the `Distance To Actor` condition specifies the `value` field of the `RelativeDistanceCondition` element. RoadRunner Scenario exports `cartesianDistance` value for the `relativeDistanceType` property. The `freespace` value of the `RelativeDistanceCondition` element is set to `false`.

Example
```
<Condition name="Start Condition of Event_CompactCar_1" conditionEdge="none" delay="0.0000000000000000e+0">
  <ByEntityCondition>
    <TriggeringEntities triggeringEntitiesRule="any">
      <EntityRef entityRef="CompactCar"/>
    </TriggeringEntities>
    <EntityCondition>
      <RelativeDistanceCondition entityRef="PickupTruck" relativeDistanceType="cartesianDistance" value="30" freespace="
    </EntityCondition>
  </ByEntityCondition>
</Condition>
```

Behavior Parameter Condition

RoadRunner Scenario exports the `Behavior Parameter` condition using the `ParameterCondition` element of ASAM OpenSCENARIO 1.x.

Example
```
<Condition name="Behavior_Parameter" conditionEdge="none" delay="0.0000000000000000e+0">
    <ByValueCondition>
        <ParameterCondition parameterRef="Yellow_Headlights" value="ON" rule="equalTo"/>
    </ByValueCondition>
</Condition>
```

Global Parameter Condition

RoadRunner Scenario exports the `Global Parameter` condition using the `ParameterCondition` element of ASAM OpenSCENARIO 1.x.

Example
```
<Condition name="Global_Parameter" conditionEdge="none" delay="0.0000000000000000e+0">
    <ByValueCondition>
        <ParameterCondition parameterRef="SpeedLimit" value="5" rule="lessOrEqual"/>
    </ByValueCondition>
</Condition>
```

Longitudinal Distance To Actor Condition

RoadRunner Scenario exports the `Longitudinal Distance To Actor` condition using the `RelativeDistanceCondition` element of ASAM OpenSCENARIO 1.x. The `relativeDistanceType` property is set to a value of `longitudinal` in the exported file.

The **Threshold** attribute of the `Longitudinal Distance To Actor` condition specifies the `value` property of the `RelativeDistanceCondition` element. The **Measure From** attribute specifies the `freespace` property. When you set the **Measure From** attribute to `Bounding boxes`, RoadRunner Scenario sets the `freespace` property to `true` in the exported file. Otherwise, it sets the `freespace` property to `false`.

Note The **Relative Position** and the **Measure Distance** attributes of the `Longitudinal Distance To Actor` condition are not applicable to the ASAM OpenSCENARIO 1.x file format. Your specifications for these attributes do not affect the exported file.

Example

```
<Condition name="" conditionEdge="none" delay="0.0000000000000000e+0">
  <ByEntityCondition>
    <TriggeringEntities triggeringEntitiesRule="any">
      <EntityRef entityRef="Sedan"/>
    </TriggeringEntities>
    <EntityCondition>
      <RelativeDistanceCondition entityRef="CompactCar" relativeDistanceType="longitudinal" value="30" freespace="true"
    </EntityCondition>
  </ByEntityCondition>
</Condition>
```

Collision Condition

The `Collision` condition checks whether the referenced entities are involved in a collision. RoadRunner Scenario exports this condition using a `collisionCondition` element.

Example

```
<Condition name="" conditionEdge="none" delay="0.0000000000000000e+0">
  <ByEntityCondition>
      <TriggeringEntities triggeringEntitiesRule="any">
        <EntityRef entityRef="CompactCar"/>
      </TriggeringEntities>
      <EntityCondition>
        <CollisionCondition entityRef="Sedan"/>
      </EntityCondition>
  </ByEntityCondition>
</Condition>
```

Phase State Condition

RoadRunner Scenario exports the `Phase State` condition using the `StoryboardElementStateCondition` element of ASAM OpenSCENARIO 1.x.

Example

```
<Condition name="Phase_State3" conditionEdge="none" delay="0.0000000000000000e+0">
    <ByValueCondition>
        <StoryboardElementStateCondition storyboardElementType="act" storyboardElementRef="Act" state="runningState"/>
    </ByValueCondition>
</Condition>
```

Duration Condition

The `Duration` condition specifies the duration of an action phase. RoadRunner Scenario exports this condition using a `StoryBoardElementStateCondition` element that references the action phase for which you specify the `Duration` condition. The specified duration is exported as a `delay` attribute of `StoryBoardElementStateCondition`. The `state` attribute of the `StoryBoardElementStateCondition` is set to `startTransition`.

Example

```
<Condition name="Start Condition durations of Event_Speed_Action_CompactCar_1" conditionEdge="none" delay="5.00000000000
  <ByValueCondition>
    <StoryboardElementStateCondition storyboardElementType="action" storyboardElementRef="Speed_Action_CompactCar_1" sta
  </ByValueCondition>
</Condition>
```

Limitations of ASAM OpenSCENARIO 1.x Export

- Export of a separate file for the actor catalog is not supported.

- Scenarios with composite conditions are not supported for esmini validation. In the **Logic** editor, composite conditions are indicated by this icon:

- Not all actions and conditions are supported for ASAM OpenSCENARIO 1.x export. This table shows the complete list of supported elements and attributes for ASAM OpenSCENARIO 1.x export.

Supported Element or Attribute
AbsoluteTargetLaneOffset
AbsoluteTargetSpeed
Act
Action
Actors
AddEntityAction
Axle
Axles
BoundingBox
ByEntityCondition
ByValueCondition
CatalogLocations
Center
CollisionCondition
Condition
ConditionGroup
Dimensions
DistanceCondition
Entities
EntityCondition
EntityObject
EntityRef
EntitySelection
Event
File
FileHeader
FollowTrajectoryAction
GlobalAction

Supported Element or Attribute
`Init`
`LaneChangeAction`
`LaneChangeTarget`
`LaneOffsetAction`
`LaneOffsetActionDynamics`
`LaneOffsetTarget`
`LanePosition`
`LateralAction`
`LongitudinalAction`
`LongitudinalDistanceAction`
`Maneuver`
`ManeuverGroup`
`MiscObject`
`OpenScenario`
`Orientation`
`ParameterAction`
`ParameterCondition`
`ParameterDeclaration`
`Performance`
`Polyline`
`Position`
`Private`
`PrivateAction`
`ReachPositionCondition`
`RelativeDistanceCondition`
`RelativeLanePosition`
`RelativeSpeedCondition`
`RelativeTargetLane`
`RelativeTargetSpeed`
`RoadNetwork`
`RoutingAction`
`ScenarioObject`
`SelectedEntities`
`Shape`
`SimulationTimeCondition`
`SpeedAction`

Supported Element or Attribute
SpeedActionTarget
SpeedCondition
Story
Storyboard
StoryboardElementStateCondition
TeleportAction
TimeReference
Trajectory
TransitionDynamics
Trigger
TriggeringEntities
UserDefinedAction
Vehicle
Vertex
Waypoint
WorldPosition

ASAM OpenSCENARIO 2.0 Representations

This section describes how vehicle properties and dynamic logic specified using various actions and conditions in RoadRunner Scenario are represented in the ASAM OpenSCENARIO 2.0 format.

Properties of Actors

RoadRunner Scenario exports the properties of vehicles in the scenario using a `vehicle` actor. Vehicle properties are derived from the corresponding vehicle asset in the **Library Browser**. For more information on vehicle assets and their parameters, see Vehicle Assets.

Example

```
compact_car: vehicle with:
    keep(it.color == white)
    keep(it.geometry_reference == "Vehicles/CompactCar.fbx")
    keep(it.bounding_box.center.x == 0m)
    keep(it.bounding_box.center.y == 0m)
    keep(it.bounding_box.center.z == 0.800383m)
    keep(it.bounding_box.length == 3.65698m)
    keep(it.bounding_box.width == 1.86693m)
    keep(it.bounding_box.height == 1.22452m)
    keep(it.center_of_gravity.x == 0m)
    keep(it.center_of_gravity.y == 0m)
    keep(it.center_of_gravity.z == 0.800383m)
    keep(it.vehicle_category == car)
    keep(it.axles.size == 2)
    keep(it.axles[1].max_steering == 0.698132rad)
    keep(it.axles[1].wheel_diameter == 0.63043m)
    keep(it.axles[1].track_width == 1.19184m)
    keep(it.axles[1].position_x == 1.17035m)
    keep(it.axles[1].position_z == 0.315215m)
    keep(it.axles[1].number_of_wheels == 2)
    keep(it.axles[2].max_steering == 0.698132rad)
    keep(it.axles[2].wheel_diameter == 0.63043m)
    keep(it.axles[2].track_width == 1.19184m)
    keep(it.axles[2].position_x == -1.2491m)
    keep(it.axles[2].position_z == 0.315215m)
    keep(it.axles[2].number_of_wheels == 2)
    keep(it.rear_overhang == 0.579387m)
```

RoadRunner Scenario exports the properties of pedestrians in the scenario using a `person` actor. Pedestrian properties are derived from the corresponding character asset in the **Library Browser**. For more information on character assets and their parameters, see Character Assets.

Example

```
citizen_male: person with:
    keep(it.color == white)
    keep(it.geometry_reference == "Characters/Citizen_Male.rrchar")
    keep(it.bounding_box.center.x == 0.0681267m)
    keep(it.bounding_box.center.y == 0.0027168m)
    keep(it.bounding_box.center.z == 0.872004m)
    keep(it.bounding_box.length == 0.391623m)
    keep(it.bounding_box.width == 0.626842m)
    keep(it.bounding_box.height == 1.74811m)
    keep(it.center_of_gravity.x == 0.0681267m)
    keep(it.center_of_gravity.y == 0.0027168m)
    keep(it.center_of_gravity.z == 0.872004m)
```

Lane-Following Motion

When a vehicle performs a lane-following motion, RoadRunner Scenario defines the route information of the vehicle using a `create_route` method in the exported file. For lane-following motions, RoadRunner Scenario supports ASAM OpenDRIVE coordinates for roads and lanes.

Example

```
sedan_route_start_point: route_point = map.odr_to_route_point(road_id: 239, lane_id: -1, s: 24.46m, t: -3.50137e-15m)
sedan_route_lane: lane with:
        keep('net.asam.opendrive: roadId:239, laneId:-1' in it.anchors)
sedan_route: route = map.create_route([sedan_route_start_point, sedan_route_lane], connect_points_by: lane)
```

After defining the route of the vehicle, RoadRunner Scenario uses the `along` modifier in the first action phase of the vehicle to specify its movement along the route.

> **Example**
> ```
> do root_phase: parallel:
>     sedan.drive() with:
>         along(sedan_route)
> ```

Path-Following Motion

When a vehicle performs a path-following motion without **Timing Data** for waypoints, RoadRunner Scenario defines the list of waypoints using a `create_path` method in the exported file.

> **Example**
> ```
> sedan_route_points: list of pose_3d = [sedan_route_point_1, sedan_route_point_2]
> sedan_route_path: path = map.create_path(points: sedan_route_points, interpolation: smooth)
> sedan_route: route = map.create_route(sedan_route_path)
> ```

When a vehicle performs a path-following motion with **Timing Data** for waypoints in the scenario, RoadRunner Scenario exports the list of polyline points and time information using a `create_trajectory` method.

> **Example**
> ```
> sedan_route_time_stamps: list of time = [0s, 3s]
> sedan_route_points: list of pose_3d = [sedan_route_point_1, sedan_route_point_2]
> sedan_route: trajectory = map.create_trajectory(points: sedan_route_points, time_stamps: sedan_route_time_stamps, interp
> ```

For path-following motions, RoadRunner Scenario supports world *XYZ*-coordinates.

When you define a path-following motion without waypoint timing data, RoadRunner Scenario exports the `along` modifier in the first action phase of the actor. However, if you define a path-following motion with waypoint timing data, RoadRunner Scenario exports the `along_trajectory` modifier in the first action phase of the actor to specify its movement along the defined trajectory.

> **Example**
> ```
> do root_phase: parallel:
>     sedan.drive() with:
>         along_trajectory(sedan_route)
> ```

Scenario Logic

RoadRunner Scenario exports the scenario logic specified in the **Logic** editor using the `do` block. For exporting serial and parallel action phases, RoadRunner Scenario uses the `serial` and `parallel` composition operators, respectively.

This image shows a scenario with one vehicle. In this scenario, there is one serial phase, which contains these phases:

- The first phase is an action phase, which initializes the vehicle speed of 15 m/s.
- The second phase is a parallel phase consisting of two action phases, where the vehicle changes its speed and lane simultaneously.
- The third phase is an action phase specifying that the vehicle accelerates to achieve the target speed of 30 m/s.
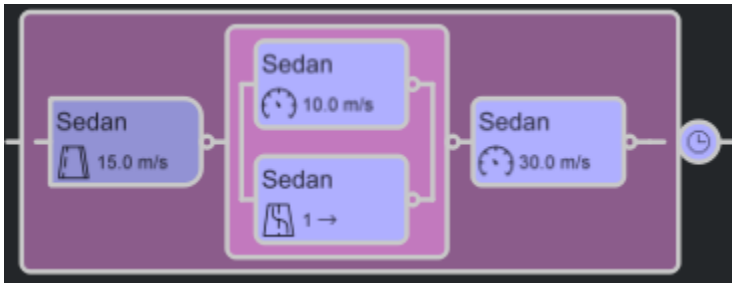
This example shows how RoadRunner Scenario exports the scenario logic of this scenario to an ASAM OpenSCENARIO 2.0 file.

Example

```
do root_phase: parallel:
    sedan.drive() with:
        along(sedan_route)
    phase_1: parallel:
        phase_2: serial:
            phase_3: sedan.assign_speed(15mps)
            phase_4: parallel:
                phase_5: sedan.change_speed(10mps, rate_profile: constant, rate_peak: 4mpss)
                phase_6: sedan.change_lane(1, side: right, rate_profile: smooth)
            phase_7: sedan.change_speed(30mps, rate_profile: constant, rate_peak: 4mpss)
    with:
        until (environment.datetime - sample(environment.datetime, @root_phase.start) >= 60s)
```

Initialize Speed Action

When you set the **Relative to** attribute of an `Initialize Speed` action to `Absolute`, RoadRunner Scenario exports the `Initialize Speed` action using an `assign_speed` action.

Example

```
phase_3: compact_car.assign_speed(17.8816mps)
```

When you set the **Relative to** attribute of an `Initialize Speed` action to `Actor`, RoadRunner Scenario exports the `Initialize Speed` action using a `speed` modifier with the `at` parameter set to `start`.

Example

```
phase_4: compact_car.drive() with:
    speed(10mps, slower_than: sedan, at: start)
```

When you set the **Relative to** attribute of an `Initialize Speed` action to `Waypoint Time Data`, RoadRunner Scenario exports the `Initialize Speed` action using a custom action, `mw_assign_time_data_based_speed`.

Example

```
compact_car.mw_assign_time_data_based_speed()
```

Change Speed Action

When you set the **Relative to** attribute of a `Change Speed` action to `Absolute`, RoadRunner Scenario exports the `Change Speed` action using a `change_speed` action.

Example

```
compact_car.change_speed(35mps, rate_profile: smooth, duration: 5s)
```

When you set the **Relative to** attribute of a `Change Speed` action to `Actor`, RoadRunner Scenario exports the `Change Speed` action using a `speed` modifier with the `at` parameter set to `end`. If you also set the **Speed Sampling** attribute of the `Change Speed` action to `At start of action`, then RoadRunner Scenario sets the `track` parameter of the `speed` modifier to `projected`.

Example

```
phase_4: compact_car.drive() with:
    speed(10mps, slower_than: sedan, at: end, track: projected, shape: speed_shape_1)
```

When you set the **Speed Sampling** attribute of the `Change Speed` action to `Continuous`, RoadRunner Scenario exports two `speed` modifiers, as shown in this example.

Example

```
phase_7: serial:
    compact_car.drive() with:
        speed(5mps, faster_than: sedan, at: end, track: actual, shape: speed_shape_1)
    compact_car.drive() with:
        speed(5mps, faster_than: sedan, at: all, track: actual)
```

The `shape` parameter of the first `speed` modifier specifies the dynamics type and profile by which the vehicle achieves the target speed. RoadRunner Scenario exports the `shape` parameter as shown in this example.

Example

```
speed_shape_1: common_speed_shape with:
    keep(it.rate_profile == constant)
    keep(it.rate_peak ==  4mpss)
```

This table shows how RoadRunner Scenario maps the **Dynamics Profile** attribute to the `rate_profile` parameter of the ASAM OpenSCENARIO 2.0 file format.

Dynamics Profile Attribute of RoadRunner Scenario	rate_profile Parameter in Exported File
Cubic	smooth
Linear	constant
Step	asap

This table shows how RoadRunner Scenario maps the **Dynamics Type** attribute to the ASAM OpenSCENARIO 2.0 file format.

Dynamics Type Attribute of Change Speed action in RoadRunner Scenario	Parameter of speed modifier in Exported File
With acceleration	rate_peak
Over time	duration
Over distance	Not supported for export

When you set the **Relative to** attribute of a `Change Speed` action to `Waypoint Time Data`, RoadRunner Scenario exports the `Change Speed` action using a custom action, `mw_change_time_data_based_speed`.

Example
`compact_car.mw_change_time_data_based_speed()`

Change Lane Action

RoadRunner Scenario exports `Change Lane` actions using the `change_lane` action of the ASAM OpenSCENARIO 2.0 specifications. This table shows how RoadRunner Scenario exports the **Dynamics Type** attribute to an ASAM OpenSCENARIO 2.0 file.

Dynamics Type Attribute of Change Lane action in RoadRunner Scenario	Parameter of change_lane action in Exported File
`With lateral velocity`	`rate_peak`
`Over time`	`duration`
`Over distance`	Not supported for export

Example
`phase_4: compact_car.change_lane(2, side: right, rate_profile: smooth, rate_peak: 5mps)`

Change Lateral Offset Action

RoadRunner Scenario exports `Change Lateral Offset` actions using the `follow_lane` action of the ASAM OpenSCENARIO 2.0 specifications. The **Direction** attribute specifies the sign of the offset parameter value in the exported file.

Direction Attribute	Offset Parameter Value in Exported File
`To the left`	Positive value specified using the **Lateral Offset** attribute
`To the right`	Negative value specified using the **Lateral Offset** attribute
`To center`	`0`

Example
`phase_4: compact_car.follow_lane(-1m, rate_profile: smooth, duration: 5s)`

Change Longitudinal Distance Action

When you set the **Distance Type** attribute of the `Change Longitudinal Distance` action to `Space`, RoadRunner Scenario exports `Change Longitudinal Distance` actions using a `change_space_gap` action of the ASAM OpenSCENARIO 2.0 specifications. Otherwise, it exports `Change Longitudinal Distance` actions using a `change_time_headway` action.

Example
`phase_4: compact_car.change_space_gap(20m, behind, sedan)`

When you set the **Sampling Mode** attribute of the `Change Longitudinal Distance` action to `Continuous`, RoadRunner Scenario exports an additional action entity for the specified space gap or time headway using a `keep_space_gap` action or `keep_time_headway` action, respectively.

Example
```
phase_4: serial:
    compact_car.change_space_gap(20m, behind, sedan)
    compact_car.keep_space_gap(sedan, longitudinal)
```

Note

- RoadRunner Scenario does not support exporting the **Dynamic Constraints** attribute of a `Change Longitudinal Distance` action.

- ASAM OpenSCENARIO 2.0 specifications support only the bounding box type of measurement for specifying the space gap. If you set the **Distance Type** attribute to `Space` and the **Measure From** attribute to `Origins`, then RoadRunner Scenario exports the equivalent bounding box measurement value of the **Space Distance Offset** attribute.

- ASAM OpenSCENARIO 2.0 specifications support only reference points measurement for specifying time headway. If you set the **Distance Type** attribute to `Time`, then RoadRunner Scenario ignores the value of the **Measure From** attribute.

Change Behavior Parameter Action

RoadRunner Scenario exports `Change Behavior Parameter` actions using these steps:

1 Export the behavior parameters as parameters of a custom actor.

Example
```
actor mw_example_movable_object inherits vehicle:
    var headlights: string = "OFF" # Behavior Parameter
    var taillights: string = "OFF" # Behavior Parameter
```

2 Define a custom action because the ASAM OpenSCENARIO 2.0 file format does not specify an equivalent action.

Example
```
action mw_change_behavior_parameter inherits movable_object.action_for_movable_object:
    param_name: string
    param_value: string
```

3 Invoke the custom action.

Example
```
compact_car.mw_change_behavior_parameter(param_name: "headlights", param_value: "ON")
```

Change Global Parameter Action

RoadRunner Scenario exports `Change Global Parameter` actions using these steps:

1 Export the global parameters as scenario variables.

Example
```
# Scenario Variables
var precipitation_density: float = 0
```

2 Define a custom action because the ASAM OpenSCENARIO 2.0 file format does not specify an equivalent action.

Example

```
action mw_change_global_parameter inherits osc_action:
    var_name: string
    var_value: string
```

3 Invoke the custom action.

Example

```
mw_change_global_parameter(param_name: "precipitation_density", param_value: ".75")
```

User Defined Action

RoadRunner Scenario exports `User Defined` actions using these steps:

1 Define an action, which inherits from the standard action
`movable_object.action_for_movable_object`.

Example

```
action low_pressure_warning inherits movable_object.action_for_movable_object:
    pressure: string = "15psi"
    tire: string = "FrontRight"
```

2 Invoke the defined action.

Example

```
compact_car.low_pressure_warning(pressure: "15psi", tire: "FrontLeft")
```

Wait Action

RoadRunner Scenario exports `Wait` actions using the `wait` directive of the ASAM OpenSCENARIO 2.0 file format.

Example

```
wait elapsed(1s)
```

Actor Speed Condition

When you set the **Relative to** attribute of an `Actor Speed` condition to `Absolute`, RoadRunner Scenario exports the `Actor Speed` condition as shown in this example.

Example

```
until (sedan.speed == 10mps)
```

When you set the **Relative to** attribute of an `Actor Speed` condition to `Actor`, and the **Speed Sampling** attribute to `At start of action`, RoadRunner Scenario exports the `Actor Speed` condition using the `sample` expression of the ASAM OpenSCENARIO 2.0 file format.

Example

```
until (sedan.speed == sample(compact_car.speed, @start)+10mps)
```

When you set the **Relative to** attribute of an `Actor Speed` condition to `Actor`, and the **Speed Sampling** attribute to `Continuous`, RoadRunner Scenario exports the `Actor Speed` condition as shown in this example.

Example
`until (sedan.speed == compact_car.speed+10mps)`

Distance To Point Condition

When exporting `Distance To Point` conditions, RoadRunner Scenario first defines a custom method for the `physical_object` entity.

Example
`extend physical_object:` `    def mw_point_euclidean_dist(reference: position_3d) -> length is undefined`

Then, RoadRunner Scenario exports the `Distance To Point` conditions by calling the custom method using the `until` directive.

Example
`until (compact_car.mw_point_euclidean_dist(point_205) < 15m)`

Distance To Actor Condition

When exporting `Distance To Actor` conditions, RoadRunner Scenario first defines a custom method for the `physical_object` entity.

Example
`extend physical_object:` `    def mw_actor_euclidean_dist(reference: physical_object) -> length is undefined`

Then, RoadRunner Scenario exports the `Distance To Actor` conditions by calling the custom method using the `until` directive.

Example
`until (compact_car.mw_actor_euclidean_dist(sedan) < 10m)`

Behavior Parameter Condition

RoadRunner Scenario exports `Behavior Parameter` conditions using these steps:

1   Define the behavior parameters as the parameters of a custom actor.

Example

`actor mw_example_movable_object inherits vehicle:`
`    var headlights: string = "OFF" # Behavior Parameter`
`    var taillights: string = "OFF" # Behavior Parameter`

2   Use the defined behavior parameters with the `until` directive.

Example
`until (compact_car.headlights == "ON")`

Global Parameter Condition

RoadRunner Scenario exports `Global Parameter` conditions using these steps:

1   Define the global parameters as scenario variables.

> **Example**
> ```
> # Scenario Variables
> var precipitation_density: string = "0.0"
> ```

2   Use the defined global parameters with the `wait` directive.

> **Example**
> ```
> wait (precipitation_density > ".2")
> ```

Longitudinal Distance To Actor Condition

When you set the **Measure Distance** attribute of a `Longitudinal Distance To Actor` condition to `Along lane curvature`, RoadRunner Scenario exports the `Longitudinal Distance To Actor` condition using a `space_gap` method, which returns the space distance between the bounding boxes of actors. In this case, RoadRunner Scenario ignores the **Measure From** attribute.

> **Example**
> ```
> until (compact_car.space_gap(sedan, direction: longitudinal) < 5m)
> ```

When you set the **Measure Distance** attribute a `Longitudinal Distance To Actor` condition to `Longitudinal distance only`, RoadRunner Scenario exports the `Longitudinal Distance To Actor` condition using an `object_distance` method.

> **Example**
> ```
> until (compact_car.object_distance(sedan, direction: longitudinal, mode: reference_points) < 5m)
> ```

Collision Condition

RoadRunner Scenario exports a `Collision` condition between any two actors as shown in this example.

> **Example**
> ```
> # The scenario contains three vehicles, named white, red, and yellow
> until ((white.time_to_collision(red) == 0s) or (white.time_to_collision(yellow) == 0s) or (red.time_to_collision(yellow)
> ```

When you specify one actor, RoadRunner Scenario exports the collision conditions between the specified actor and any other actors as shown in this example.

> **Example**
> ```
> # The scenario contains three vehicles, named white, red, and yellow.
> # The specified actor of the collision condition is white
> until ((white.time_to_collision(red) == 0s) or (white.time_to_collision(yellow) == 0s)
> ```

When you specify two actors, RoadRunner Scenario exports the collision condition between the specified actors as shown in this example.

> **Example**
> ```
> until ((white.time_to_collision(red) == 0s)
> ```

Note If a scenario contains only one actor, then RoadRunner Scenario exports the `Collision` condition as a `false` condition. For example:

```
until (false)
```

Phase State Condition

RoadRunner Scenario exports `Phase State` conditions using predefined `start` and `end` events. When you set the **Phase State** attribute of the `Phase State` condition to `Running`, RoadRunner Scenario exports `Phase State` conditions using `start` events. Otherwise, it exports `Phase State` conditions using `end` events.

Example

```
phase_4: compact_car.change_speed(15mps) with:
    until @phase_4.end
```

Simulation Time Condition

RoadRunner Scenario exports `Simulation Time` conditions of action phases using these steps:

1 Instantiate the environment to extract its `datetime` property.

Example

```
environment: environment
```

2 Subtract the start time of the scenario from the current time.

Example

```
until (environment.datetime - sample(environment.datetime,@root_phase.start) >= 60s)
```

Note ASAM OpenSCENARIO recommends avoiding use of the `Simulation Time` condition for version 2.0 of the file specification.

Duration Condition

RoadRunner Scenario exports `Duration` conditions using an `elapsed` expression.

Example

```
until elapsed(60s)
```

Fail Condition

RoadRunner Scenario exports `Fail` conditions using a custom event and an `on` directive.

Example

```
# Event declaration to express scenario failure condition
event mw_scenario_fail_event is (compact_car.time_to_collision(sedan) == 0s)

on @mw_scenario_fail_event:
    emit fail(free_drive)
```

See Also

Related Examples

- "Generate Scenario Variations Using gRPC API" on page 4-2
- "Import Trajectories from ASAM OpenSCENARIO Files" on page 2-2
- "Export to ASAM OpenDRIVE"
- "Export to OpenSceneGraph"
- "Export to ASAM OpenCRG"
- "Importing ASAM OpenDRIVE Files"

External Websites

- ASAM OpenSCENARIO

Convert Asset Data Between RoadRunner Scenario and ASAM OpenSCENARIO

See Also

6

Simulate Scenarios with CARLA

# Overview of RoadRunner Scenario and CARLA Cosimulation

Cosimulation enables multiple simulation tools, such as RoadRunner Scenario, CARLA, and Automated Driving Toolbox, to coordinate so that each tool either managing specific aspects of the simulation or grouping objects within the simulation to produce a complete simulation of a driving scenario. RoadRunner Scenario and CARLA can cosimulate a large number of actors within a simulation. Cosimulation enables RoadRunner Scenario to control target vehicles in a scene, with easy state and behavior control, while the CARLA software engine or Automated Driving Toolbox manages the ego vehicles in the scene without obfuscating the ego vehicle simulation. Cosimulation combines the ease of development and design provided by RoadRunner Scenario with the scalability provided by CARLA.

Scenario Simulation Engine (SSE)

The scenario simulation engine is a background server that coordinates simulations between multiple clients. These clients can include:

- RoadRunner Scenario
- CARLA
- MATLAB and Simulink with Automated Driving Toolbox

Each of these simulation tools manage their own respective physics and environment simulation and a specified group of actors. The SSE coordinates information and synchronizes the physics, environments, and actors between each of the simulation clients, enabling all of the simulations to synchronize states and simulation times. This diagram illustrates the organization of the clients and the SSE server.
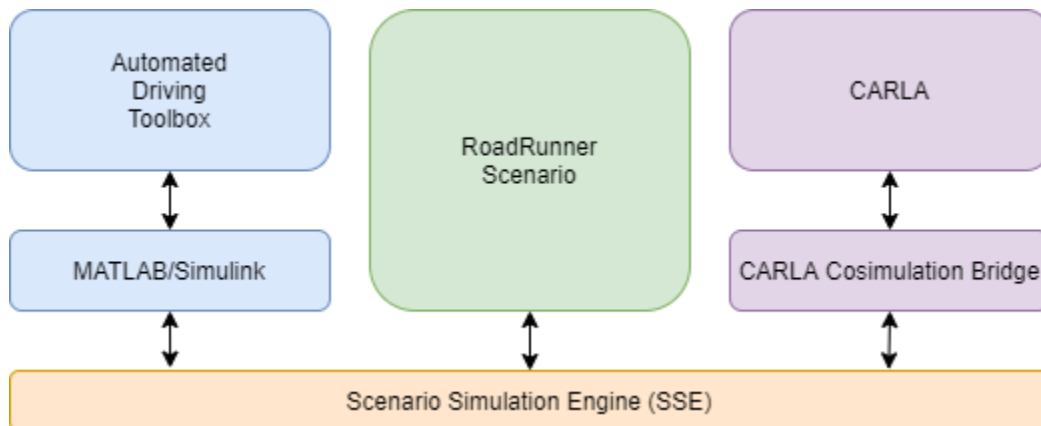
CARLA and Cosimulation Bridge

CARLA, by default, executes as a standalone simulation and simulation server. To cosimulate with RoadRunner Scenario, CARLA requires a cosimulation bridge. The cosimulation bridge acts as an intermediary and connects CARLA to the SSE as a client. The CARLA cosimulation bridge uses the open-source remote procedure call (RPC) library gRPC as its communication bridge. For more information on gRPC, see https://grpc.io/. For instructions on how to set up the cosimulation bridge between RoadRunner Scenario and CARLA, see "Set Up CARLA for Cosimulation" on page 6-5

See Also

"Overview of Simulating RoadRunner Scenarios with MATLAB and Simulink" (Automated Driving Toolbox) | "Set Up CARLA for Cosimulation" on page 6-5 | Simulation Configuration | "CARLA Cosimulation Workflow" on page 6-4

External Websites

- https://grpc.io/
- https://carla.readthedocs.io/en/latest/python_api/

CARLA Cosimulation Workflow

RoadRunner Scenario and CARLA can cosimulate a large number of actors within a simulation. Cosimulation enables RoadRunner Scenario to control an ego vehicle in a scene, with easy state and behavior control, while the CARLA software engine manages other vehicles in the scene without obfuscating the ego vehicle simulation.. Cosimulation combines the ease of development and design provided by RoadRunner Scenario with the scalability provided by CARLA.

This is the typical workflow for cosimulation between RoadRunner Scenario and CARLA:

1   "Set Up CARLA for Cosimulation" on page 6-5 — Setup and configure CARLA software and other required software to enable cosimulation between RoadRunner Scenario and CARLA.

> **Note** This setup only has to be performed once for the currently installed versions of RoadRunner Scenario and CARLA.

2   "Configure RoadRunner Scenario Model" on page 6-7 — Create or modify a new or existing RoadRunner Scenario scene to include vehicles, such as the ego vehicle, to be controlled by CARLA.

3   (Optionally) "Export Scenes and Visualizations to CARLA" on page 6-12 — Package and share a scene and visualization developed in RoadRunner Scenario with CARLA to allow simultaneous visualizations with both simulation tools.

4   "Run Cosimulations with CARLA" on page 6-14 — Run cosimulations with CARLA either as a background process or with a shared visualization.

See Also

More About

*   "Overview of RoadRunner Scenario and CARLA Cosimulation" on page 6-2

External Websites

*   https://carla.org/

Set Up CARLA for Cosimulation

Install CARLA

Download CARLA from the GitHub release site. https://github.com/carla-simulator/carla/releases. Unzip the release into a folder, which becomes the CARLA installation directory *CARLAInstall*.

Note

- RoadRunner Scenario has been tested with only CARLA version `0.9.12`.
- CARLA version `0.9.12` requires Python® `3.7`.

Generate CARLA Cosimulation Bridge

RoadRunner Scenario and CARLA cosimulation synchronize and communicate using the CARLA cosimulation bridge. For information on the cosimulation bridge, see "Overview of RoadRunner Scenario and CARLA Cosimulation" on page 6-2.

Follow these steps to generate the cosimulation bridge:

1 Install Python 3.7.

2 Add Python to your path. For information on how to do this in Microsoft Windows, see https://docs.python.org/3/using/windows.html#excursus-setting-environment-variables in the Python documentation.

3 Update `pip` and install the `psutil` package using these commands.

```
python -m pip install --upgrade pip
python -m pip install psutil
```

4 Run *CARLAInstall*`\bin\win64\Tools\CARLA\examples\setup.bat`, where *CARLAInstall* is the folder in which RoadRunner Scenario is installed.

Note If you install RoadRunner Scenario in the `Program Files` directory, you must edit the BAT file as an administrator..

Configure Cosimulation Properties

You can optionally modify the cosimulation properties between RoadRunner Scenario and CARLA. The cosimulation properties include timing step resolution and network location of the Scenario Simulation Engine (SSE). For information on how to configure the cosimulation properties, see Simulation Configuration.

See Also

"CARLA Cosimulation Workflow" on page 6-4 | "Overview of RoadRunner Scenario and CARLA Cosimulation" on page 6-2 | Simulation Configuration

External Websites

- https://github.com/carla-simulator/carla/releases
- Python 3.7
- https://docs.python.org/3/using/windows.html#excursus-setting-environment-variables

Configure RoadRunner Scenario Model

This section shows how to configure a RoadRunner Scenario model where CARLA controls the ego vehicle and RoadRunner Scenario controls the target vehicles.

Note Before performing this configuration, you must complete the setup steps defined in "Set Up CARLA for Cosimulation" on page 6-5.


Set Up RoadRunner Scenario Model with Vehicles

Start RoadRunner Scenario from the Windows Start menu or the app shortcut.

1 Create a new project in RoadRunner.

2 In the dialog box, click **New Scene**, then **New Project**, and create or select an empty folder, *ProjectFolder*, to save your project in.

3 Select **Yes** to include the asset library.

4 Click **File > Open Scene** and select *ProjectFolder*/Scenes/ScenarioBasic.rrscene.
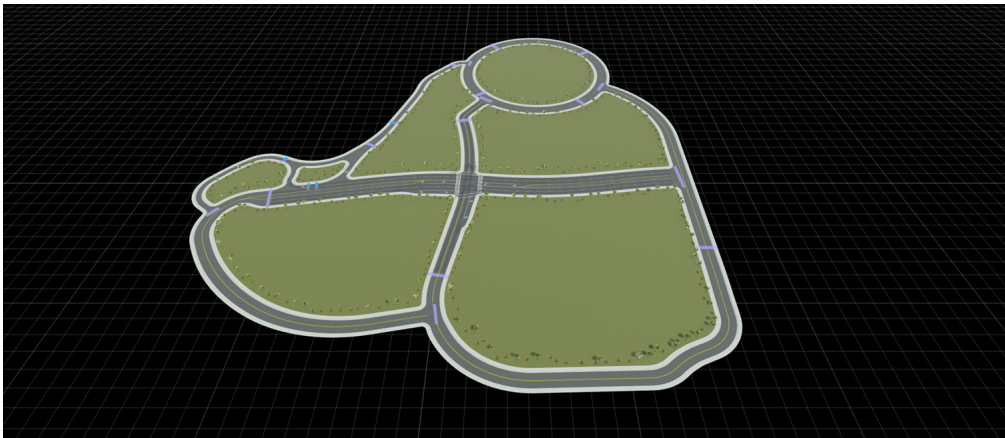
5 Switch to scenario editing mode. From the top-right corner of the RoadRunner application, select **Scene Editing**, then **Scenario Editing**.
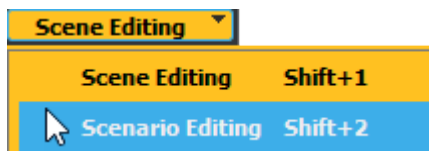
6 Create a new scenario. Select **File > New Scenario**.

7 Place a target vehicle for RoadRunner Scenario to control. Select the Sedan car from the **Library Browser**, and drag it onto the scene. With the vehicle selected in the scene, specify a path by right-clicking to create waypoints on the roads.

8   Place the ego vehicle for CARLA to control. Select the `Compact` car from the **Library Browser**, and drag it onto the scene. With the vehicle selected in the scene, specify a path by right-clicking to create waypoints on the roads. In this simulation, the compact car follows the same path as the sedan.

Add CARLA Behavior to Vehicle

Create a new folder named `Behavior` in the `Vehicles` folder. Add a new CARLA behavior by right-clicking in the folder and selecting **New > Behavior**. Set these `Behavior` attributes for the `CARLA Behavior.rrbehavior` asset:

- **Platform** — `External`
- **Platform Name** — `CARLA`
- **Model location** — `../examples/CarlaEgoVehicle1.py`

The `CarlaEgoVehicle1.py` file, included with RoadRunner Scenario, can be used as an example for creating ego vehicle behavior, with speed, path, and set parameters actions, script using the CARLA python API. For more information on the CARLA python APIs, see https://carla.readthedocs.io/en/latest/core_map/#navigating-through-waypoints.

Note Two additional scripts, `CarlaEgoVehicle2.py` and `CarlaTargetVehicle1.py`, provide sample code for an ego vehicle agent with path following and target vehicle agent behaviors, respectively.

Select the compact car. Assign the CARLA behavior by dragging the `CARLA Behavior.rrbehavior` asset to the **Attributes > Vehicle > Behavior** field. You can optionally rename the CARLA ego vehicle.
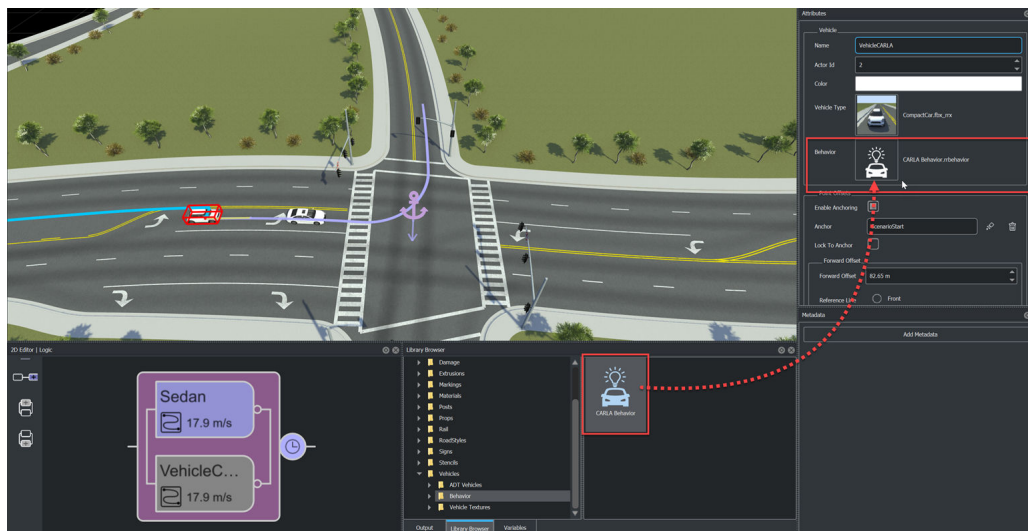
Check and update the actor mapping file *RoadrunnerInstall*\bin\win64\Tools\CARLA \examples\actors.json to include your vehicles. By default, the actors.json file includes the sedan, compact, and SUV cars. This file serves as a map between the vehicle visualization in RoadRunner Scenario and the simulation in CARLA.

```
{
    "Assets/Developer/Vehicles/CompactCar.fbx_rrx": "vehicle.mini.cooper_s",
    "Assets/Developer/Vehicles/CompactCar.fbx": "vehicle.mini.cooper_s",
    "Assets/Vehicles/CompactCar.fbx_rrx": "vehicle.mini.cooper_s",
    "Assets/Vehicles/CompactCar.fbx": "vehicle.mini.cooper_s",
    "Assets/Developer/Vehicles/Sedan.fbx_rrx": "vehicle.lincoln.mkz_2020",
    "Assets/Developer/Vehicles/Sedan.fbx": "vehicle.lincoln.mkz_2020",
    "Assets/Vehicles/Sedan.fbx_rrx": "vehicle.lincoln.mkz_2020",
    "Assets/Vehicles/Sedan.fbx": "vehicle.lincoln.mkz_2020",
    "Assets/Developer/Vehicles/Suv.fbx_rrx": "vehicle.nissan.patrol",
    "Assets/Developer/Vehicles/Suv.fbx": "vehicle.nissan.patrol",
    "Assets/Vehicles/Suv.fbx_rrx": "vehicle.nissan.patrol",
    "Assets/Vehicles/Suv.fbx": "vehicle.nissan.patrol"
}
```

Update the platform settings in C:/Users/*username*/AppData/Roaming/MathWorks/ RoadRunner/*R20NNa*/Scenario/Config/SimulationConfiguration.xml to point to the CARLA executable, for example *CARLAInstall*\WindowsNoEditor\CarlaUE4.exe, and restart RoadRunner to apply your changes. In the path *username* is your Windows user profile name and *CARLAInstall* is your CARLA installation directory. For example, if you install CARLA in the C:\CARLA_*n.n.n* directory, make these changes to the SimulationConfiguration.xml file:

```
<CoSimulationServer>
    <TimeOutValues>
        <Event name="SimulationStartEvent" value="30000"/>
        ...
    </TimeOutValues>
    ...
    <Platform name="CARLA">
        <ExecutablePath>C:\CARLA_N.N.N\WindowsNoEditor\CarlaUE4.exe</ExecutablePath>
        <StartTimeOut>60000</StartTimeOut>
    </Platform>
</CoSimulationServer>
```

For information on how to run these cosimulations, see "Run Cosimulations with CARLA" on page 6-14.

See Also
"CARLA Cosimulation Workflow" on page 6-4 | "Run Cosimulations with CARLA" on page 6-14 | "Set Up CARLA for Cosimulation" on page 6-5

Export Scenes and Visualizations to CARLA

The export workflow described in this topic extends the workflow described in "Export to CARLA" to show an equivalent visualization in RoadRunner Scenario and CARLA.

Export Scene from RoadRunner Scenario

1 Open your scene in RoadRunner.
2 Export the scene using the CARLA option. From the menu, select **File > Export > CARLA (.fbx + .xml + .xodr)**.
3 In the Export CARLA dialog box, set the mesh tiling on the **Filmbox** tab and the OpenDRIVE® options on the **OpenDRIVE** tab as needed.
4 Click **Export**.

Build and Add Plugins to CARLA

1 Build CARLA from its source. For more information, see the Windows build page of the Building CARLA instructions.

> **Note** The CARLA source code is cloned to `C:\tree\carla\`*N_N_NN*`\carla`, and Unreal® 4.24 to `C:\tree\UnrealEngine`

2 Copy the CARLA RoadRunner plugins to the folder `C:\tree\carla\`*N_N_NN*`\carla\Unreal\CarlaUE4\Plugins`.
3 Update the Microsoft Visual Studio® project. Right-click the project file `C:\tree\carla\`*N_N_NN*`\carla\Unreal\CarlaUE4\CarlaUE4.uproject` and click **Switch Unreal Engine Version**.

Add Maps and Rebuild CARLA

1 Open the Unreal project file, `CarlaUE4.uproject`.
2 Using the **Content Browser**, create new `RoadRunner\Maps` and `RoadRunner\Static` folders.
3 Drag all the files exported from RoadRunner to `RoadRunner\Static`, and follow the instructions in the import wizard.
4 Save `ScenarioBasic.umap` to the `RoadRunner\Maps` folder.
5 In the Unreal **Project Settings**, under **Project > Packaging**, add the map to the packaging list.
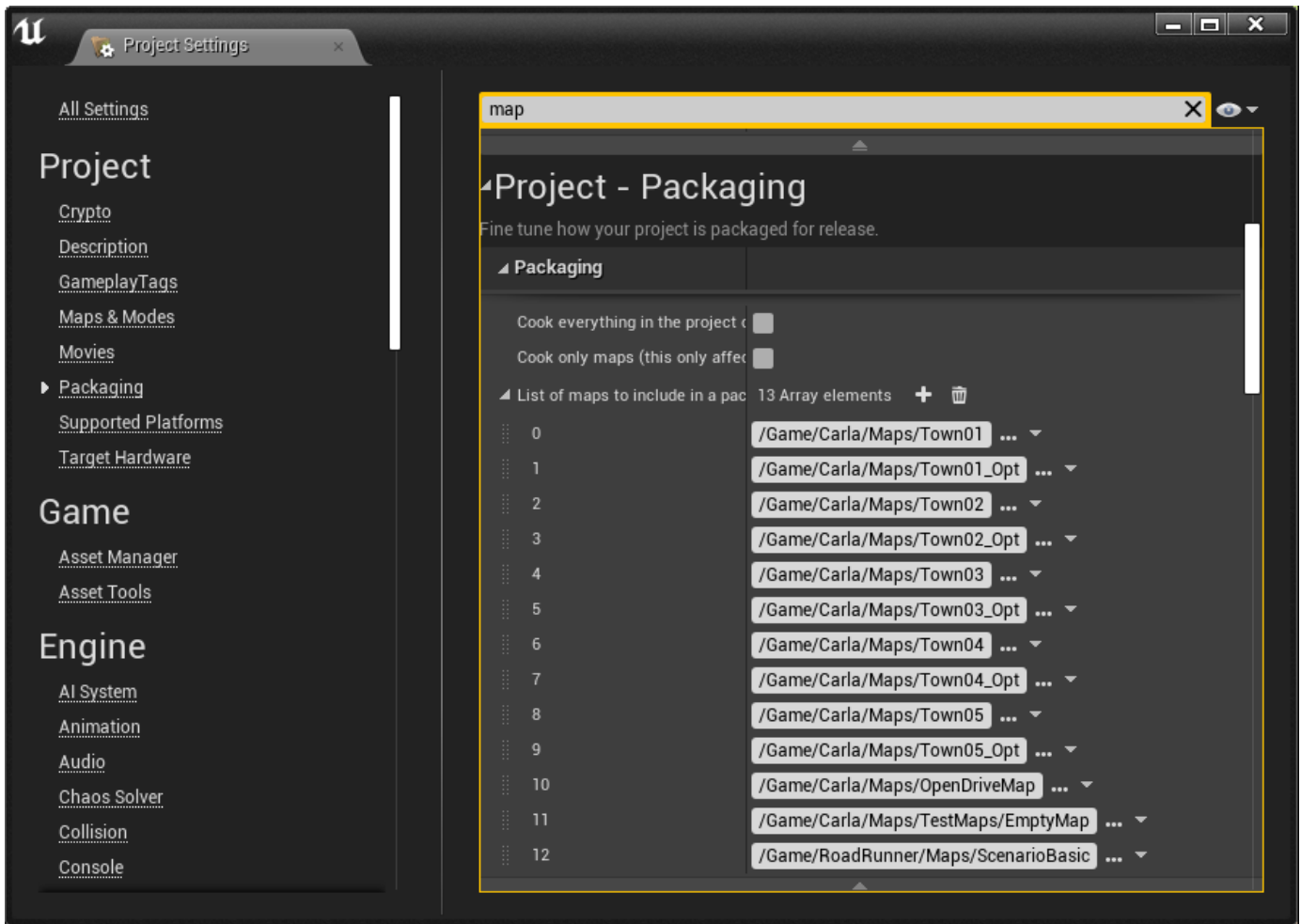
6   Close the Unreal editor and save the assets when prompted.

7   Build the CARLA executable and package the new map to `C:\tree\carla\`*`N_N_NN`*`\carla`
    `\Build\UE4Carla\`*`N_N_NN`*`-dirty\WindowsNoEditor`.

8   Set a Windows environment variable, `CARLA_ROOT`, to `C:\tree\carla\`*`N_N_NN`*`\carla\Build`
    `\UE4Carla\`*`N_N_NN`*`-dirty\WindowsNoEditor`.

9   Update the platform settings in `C:/Users/`*`username`*`/AppData/Roaming/MathWorks/`
    `RoadRunner/`*`R20NNa`*`/Scenario/Config/SimulationConfiguration.`xml to these values.

```
<TimeOutValues>
<Event name="SimulationStartEvent" value="30000"/>
...
</TimeOutValues>
...
<Platform name="CARLA">
 <ExecutablePath>C:\tree\carla\N_N_NN\carla\Build\UE4Carla\N_N_NN-dirty\WindowsNoEditor\CarlaUE4.exe</ExecutablePath>
 <StartTimeOut>60000</StartTimeOut>
</Platform>
```

The scene is now available in CARLA simulation engine.

See Also
"CARLA Cosimulation Workflow" on page 6-4 | "Export to CARLA"

Run Cosimulations with CARLA

When RoadRunner Scenario runs cosimulations, CARLA can operate either as a background process, without the user interface, or as a foreground process producing a real-time visualization of the scene.

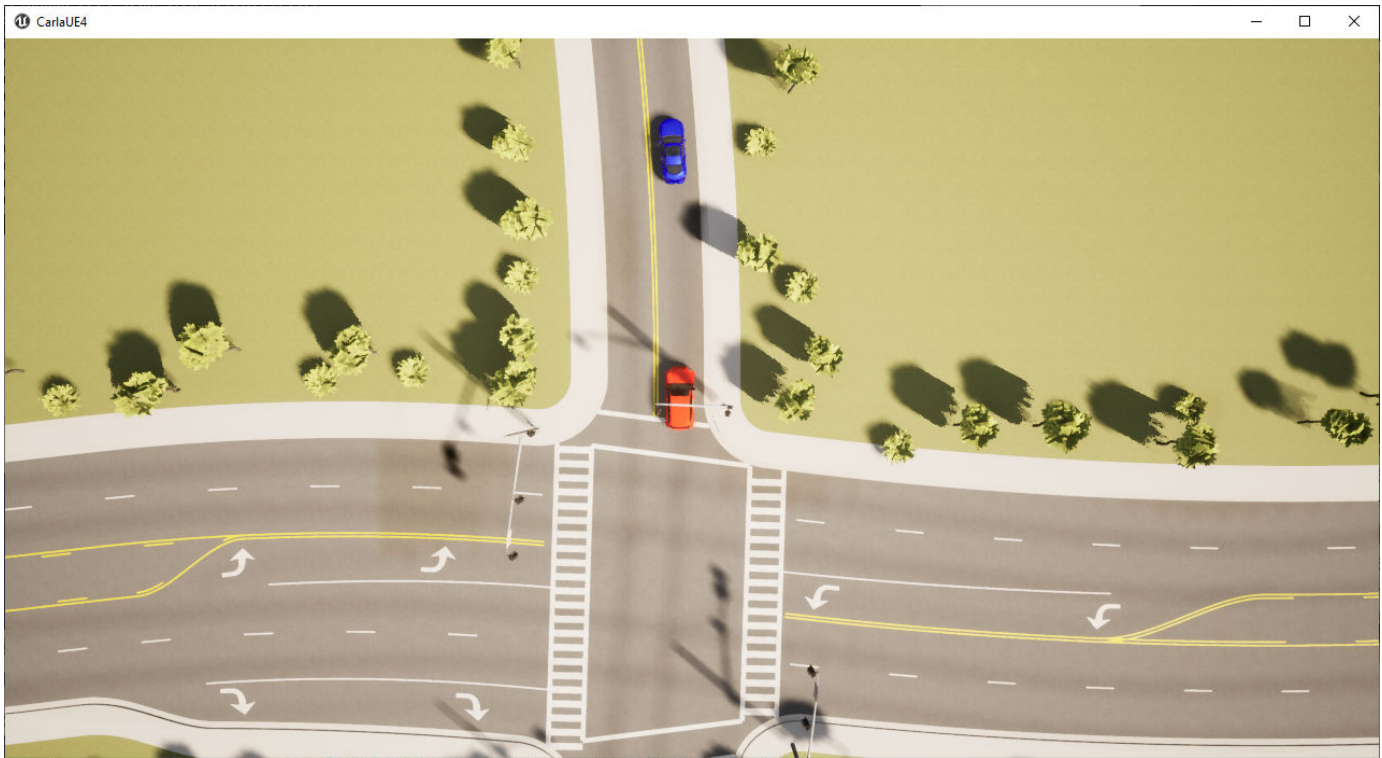### Run RoadRunner Scenario Simulation with CARLA in Background

Start RoadRunner and load your scenario file. Export your scene to an OpenDRIVE file, such as *ProjectFolder*/Scenes/ScenarioBasic.xodr. Switch to the **Simulation Tool** and click the **Play** button. The sample scene shown is a continuation of the scenario from "Configure RoadRunner Scenario Model" on page 6-7. Observe the cosimulation of vehicles in RoadRunner and CARLA. RoadRunner Scenario controls the blue sedan, and CARLA controls the red hatchback.
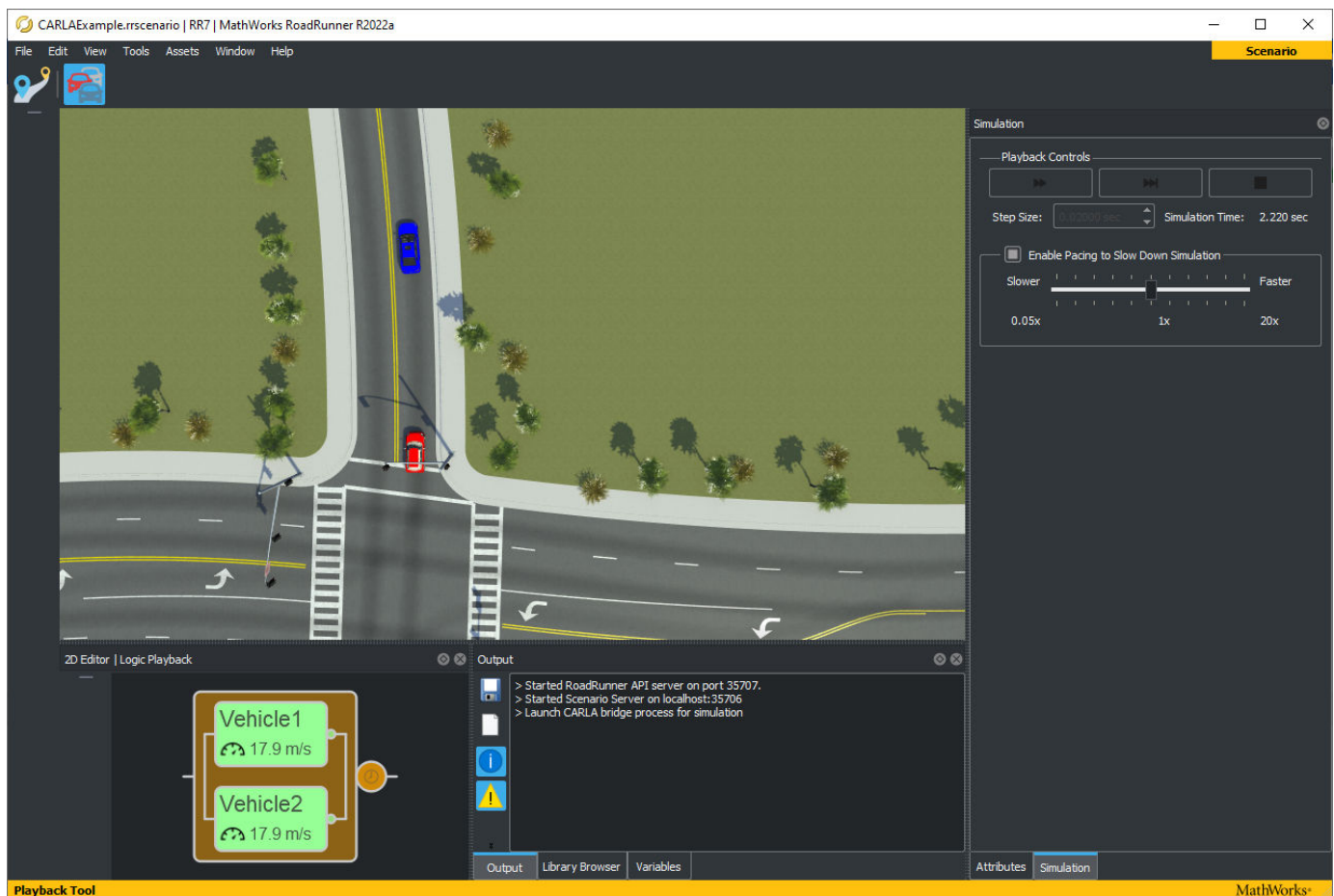
Note After clicking **Play**, CARLA may require some time to start executing, which can appear as a delayed start.

Run RoadRunner Scenario Cosimulation with CARLA Visualizations

Export your scene and visualization to CARLA using the steps shown in "Export Scenes and Visualizations to CARLA" on page 6-12. Restart RoadRunner, and reload your scenario file. Switch to the **Simulation Tool** and click the **Play** button. Observe the cosimulation of vehicles in RoadRunner Scenario and CARLA. RoadRunner Scenario controls the blue sedan, and CARLA controls the red hatchback.

See Also