RoadRunner

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

Contents

1

Get Started with RoadRunner

RoadRunner Fundamentals

2

3

4

<div style="text-align: right">Export Scenes</div>

5

7

RoadRunner Asset Library Product Overview

Get Started with RoadRunner

RoadRunner Product Description

Design 3D scenes for automated driving simulation

RoadRunner is an interactive editor that lets you design 3D scenes for simulating and testing automated driving systems. You can customize roadway scenes by creating region-specific road signs and markings. You can insert signs, signals, guardrails, and road damage, as well as foliage, buildings, and other 3D models. RoadRunner provides tools for setting and configuring traffic signal timing, phases, and vehicle paths at intersections.

RoadRunner supports the visualization of lidar point cloud, aerial imagery, and GIS data. You can import and export road networks using ASAM OpenDRIVE®. 3D scenes built with RoadRunner can be exported in FBX®, glTF™, OpenFlight, OpenSceneGraph, OBJ, and USD formats. The exported scenes can be used in automated driving simulators and game engines, including CARLA, VIRES VTD, NVIDIA DRIVE Sim®, SVL, Baidu Apollo®, Unity®, and Unreal Engine®.

RoadRunner Asset Library lets you quickly populate your 3D scenes with a large set of realistic and visually consistent 3D models. RoadRunner Scene Builder lets you automatically generate 3D road models from HD maps.

RoadRunner System Requirements

RoadRunner is an interactive editor that lets you design 3D scenes for simulating and testing automated driving systems. Before you install RoadRunner, check that your system meets these required specifications.

Specification	Recommended	Minimum Requirement
Operating System	Windows®: Windows 10 x64 Linux®: Ubuntu® 16.04 and 18.04.	Windows: Windows 10 x64 Linux: Ubuntu 16.04
CPU	Intel® or AMD® x86-64 processor with four logical cores operating at 3.5 GHz or higher	Intel or AMD x86-64 processor operating at 2.5 GHz or higher
Memory	16 GB	8 GB
Video Card	NVIDIA® GTX 1060 3 GB	OpenGL® 3.2-compatible with 1 GB VRAM
Disk	SSD hard drive	2 GB available disk space

For more details on graphics card requirements and support with graphics issues, see "Graphics and Startup Issues" on page 2-94.

See Also

More About

- "Install and Activate RoadRunner" on page 1-4

Install and Activate RoadRunner

RoadRunner is an interactive editor that lets you design 3D scenes for simulating and testing automated driving systems.

The procedures in this topic are for a single RoadRunner computer installation or update/upgrade on Windows or Linux. These procedures can be performed by an individual license holder or by an end user or administrator with a network license.

Follow these procedures to install RoadRunner for the first time, update an installed release of RoadRunner, or upgrade an installed release of RoadRunner to a new release.

Network License Administrators Before you or your end users install RoadRunner, perform the following tasks:

- Install the network license manager. See "Install Network License Manager for RoadRunner" on page 1-11.
- Download the network license file and the platform-specific product installer and save them to removable media or a network location. You will need to give these items to your end users for them to install on their own computers.

Verify System Requirements

Before you install RoadRunner, check that your system meets these required specifications.

Specification	Recommended	Minimum Requirement
Operating System	Windows: Windows 10 x64 Linux: Ubuntu 16.04 and 18.04.	Windows: Windows 10 x64 Linux: Ubuntu 16.04
CPU	Intel or AMD x86-64 processor with four logical cores operating at 3.5 GHz or higher	Intel or AMD x86-64 processor operating at 2.5 GHz or higher
Memory	16 GB	8 GB
Video Card	NVIDIA GTX 1060 3 GB	OpenGL 3.2-compatible with 1 GB VRAM
Disk	SSD hard drive	2 GB available disk space

Get License and Product Installer

Use an Individual License

Get your RoadRunner license by following these steps.

1 Go to the License Center on the MathWorks® website.

If prompted, sign in to your MathWorks Account.

2 Click the RoadRunner license in your account.

3   On the Install and Activate tab, click **Activate a Computer**.

4   In the form displayed, enter all requested information. When finished, click **Submit**.

5   Download or email the license file and save it to a folder on the computer where you will install RoadRunner.

Next, download the product installer by following these steps.

1   Sign in to your MathWorks Account on the MathWorks website.

2   Under **My Software**, click the down arrow next to your RoadRunner license.

   This action takes you to the Downloads page where you can find the download RoadRunner link. If you see **Get Latest Release**, click **Download R2022b** to get to the right page..

3   Under **Additional Product Downloads**, click **Get R2022b RoadRunner**.

4   Download the platform-specific installer to the computer on which you want to install RoadRunner.

Use a Network License

- **End Users** — Get the license file and platform-specific installer from your license administrator and copy them to the computer where you are installing RoadRunner.

- **Network/License Administrators** — Give end user the platform-specific installer (or provide access on a network share) and the modified network license. See "Step 3. Configure Network License" on page 1-12 for instructions on configuring the network license for end user access to RoadRunner software.

Install RoadRunner

For RoadRunner installation, GUI installation is available for Windows and GUI and command line methods are available on Linux platforms. Installation instructions vary based on the platform and method you choose.

Platform and Installation Method	Instructions
Windows GUI	**1** Double-click the downloaded product installer. **2** Follow all prompts to complete installation.
Linux GUI	**1** Double-click the `.deb` file that you just downloaded. **2** Click **Install** and follow all prompts.

Linux command line	**1** Open a command prompt.
	2 Enter the following command, replacing *release_number* with the current release number:
	
	`sudo dpkg -i RoadRunner/ `*`release_number`*`.deb`
	
	For example, for the R2022b release, use this command:
	
	`sudo dpkg -i RoadRunner_R2022b_glnxa64.deb`
	3 Follow all prompts to complete installation.

Activate License

Before you can use RoadRunner, you must activate the license. Follow these steps to install an individual or network RoadRunner license on your computer.

Note If you do not have a license, see "Get License and Product Installer" on page 1-4.

1. On your desktop, launch RoadRunner and follow all prompts.

   - **Windows** — Launch RoadRunner from the shortcut on the desktop or Start Menu.
   - **Linux** — Launch RoadRunner from the installed shortcut. To find this shortcut, click Home and type "roadrunner" in the text field or go to "/usr/share/applications".

2. When prompted for a license file, enter the path to the license file from "Get License and Product Installer" on page 1-4.

3. Follow any additional prompts to complete activation.

Create a New Project and Scene

After you install and activate RoadRunner, you can now create a project to get started creating scenes.

RoadRunner uses a project folder to store the assets (materials, models, and so on) that you can use within the application. If you do not already have an existing project, you need to create one. See "RoadRunner Project and Scene System" on page 2-2.

After following the previous instruction, RoadRunner opens to a new blank scene.
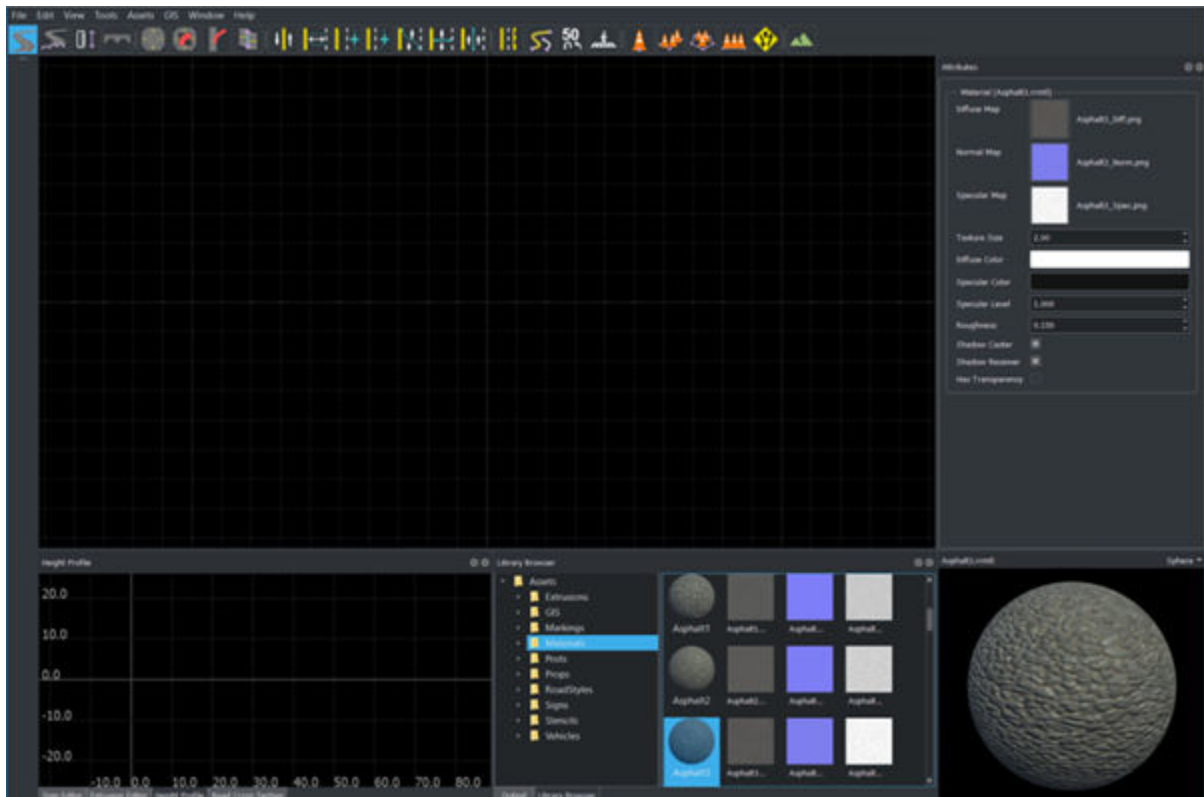
To get started, you can use the **Road Plan Tool** to create your first road. Alternatively, use one of the other tools to start creating your scene. For more on getting started, see the Getting Started with RoadRunner video series or the "Create Simple RoadRunner Scene" on page 1-19 example.

If you experience rendering issues or crashes on startup, see "Graphics and Startup Issues" on page 2-94 for troubleshooting help.

See Also

Related Examples

- "Get RoadRunner Updates and Upgrades" on page 1-8
- "Graphics and Startup Issues" on page 2-94
- "Import Scene Data"
- "Design Scenes"
- "Choose a RoadRunner Tool" on page 2-35
- "Install Network License Manager for RoadRunner" on page 1-11
- "Update Network License Manager for RoadRunner" on page 1-16

External Websites

- License Center
- Getting Started with RoadRunner

Get RoadRunner Updates and Upgrades

Update Installed Release

You can download and install a more recent version of the release of MATLAB® that you currently have installed. For example, if you have R2022b installed, you might be able to download a more recent version of R2022b. This version is called a MATLAB update. When you install a MATLAB update, you get not only the most recent update for your version but also all previous updates for your version as well.

When an updated version of RoadRunner for your installed release is available, download the installer again, and then rerun the installation, following the instructions in "Install and Activate RoadRunner" on page 1-4.

Upgrade RoadRunner Release

Upgrading involves installing a newer general release than the current installation on your computer, for example, from R2020b to R2022b. The Software Maintenance Service on your license must be current and it determines which releases you can upgrade to. For more information, contact Sales.

To upgrade to a new release of RoadRunner, rerun the steps in "Install and Activate RoadRunner" on page 1-4.

Install RoadRunner Add-On Products

To install RoadRunner add-on products (RoadRunner Asset Library, RoadRunner Scene Builder, or RoadRunner Scenario) complete these steps:

1 Contact Sales to put the add-on product on your RoadRunner license.
2 Follow all instructions in "Install and Activate RoadRunner" on page 1-4. You must get an updated license, download and rerun the installer to get the added products, and activate the updated license.

See Also

Related Examples
- "Install and Activate RoadRunner" on page 1-4
- "Update RoadRunner Licenses" on page 1-9

Update RoadRunner Licenses

If you are an individual RoadRunner user, you can download and install an updated RoadRunner license on your computer. If you are a network license administrator, you can update the network license for add-on products or to add users or seats to the license.

Update RoadRunner Individual License

When you get an updated license for RoadRunner, you must update the license on your computer.

To update your license, use the following procedure.

1   Go to the License Center on the MathWorks website. Sign in to your MathWorks account if prompted.
2   Click the RoadRunner license in your account.
3   On the Install and Activate tab, under **Get License File**, click the down arrow next to your RoadRunner license.
4   Enter the release for the license, and then click **Continue**.
5   Download or email the license file and save it to a folder on the computer where the previous RoadRunner license was located. When you have completed this step, click **Done**.
6   Launch RoadRunner, and then click **License** to update the license on your computer.
7   When prompted for the license file, enter the path to the license file you just downloaded.
8   Follow any additional prompts to complete the updated license activation.
9   Restart RoadRunner.

Update RoadRunner Network Licenses

When you receive a new license from MathWorks because the products or seat counts have changed, plan to update the license file on the network server at a time when users are least likely to be accessing a RoadRunner license.

To update the license:

1   Make a copy of the existing RoadRunner network license on the license server.
2   Go to License Center on the MathWorks website. Sign in to your MathWorks Account, if prompted.
3   On the Install and Activate tab, under **Get License File**, click the down arrow next to RoadRunner Server.
4   Download or email the license file and save it to a folder on the computer where the previous RoadRunner network license is located. When you have completed this step, click **Done**.
5   Stop the network license manager.
6   Open the existing license on the server and the new license in an editor. You are going to copy most of the new license into the old license with these instructions:

    a   In the existing license, delete all content except the `SERVER` and `DAEMON` lines at the top of the file.
    b   In the new license, copy all content starting from below the `SERVER` and `DAEMON` lines to the end.

    c   Paste the copied content into the existing license below the SERVER and DAEMON lines.

    d   Save the existing license. You can store the new license as a backup, remembering that the SERVER and DAEMON lines must be replaced with those specific to your organization.

7   If you have an options file, depending on how it is configured, you might have to update it. If you do not have an options file, skip this step.

8   Start the network license manager.

9   Have end users restart RoadRunner.

See Also

Related Examples

- "Update Network License Manager for RoadRunner" on page 1-16

Install Network License Manager for RoadRunner

Overview

> **Note** You must be a **network license administrator** to perform these procedures. If you are an end user on a network license and have been asked to install and/or activate RoadRunner yourself, follow the instructions in "Install and Activate RoadRunner" on page 1-4.

Before you or your end users can install and run RoadRunner software, you must first install a network license manager from MathWorks to manage the RoadRunner network license. The instructions in this topic apply to both Network Named User and Concurrent licenses.

Select one of these workflows:

- If you want to install a network license manager for RoadRunner only, see "Install New Network License Manager for RoadRunner Only" on page 1-11.
- If you want to install a network license manager for RoadRunner and other MathWorks products, see "Install New Network License Manager for All Products" on page 1-13.

If you already have the MathWorks network license manager and you want to add RoadRunner, see "Update Existing Network License Manager for New RoadRunner Installation" on page 1-16.

Install New Network License Manager for RoadRunner Only

Follow this workflow if you are installing the network license manager from MathWorks for the first time and you are planning to install only RoadRunner at this time.

Step 1. Download RoadRunner Network License

1   Go to License Center on the MathWorks website and select the RoadRunner network license.

2   On the Install and Activate tab, click **Activate to Retrieve License File** and follow all prompts. For application label, enter "RoadRunner Server."

3   Download or email the license file and save it on the server that will be hosting the network license manager.

Step 2. Install Network License Manager

1   Download the MathWorks-specific license manager daemons from License Manager Files on the MathWorks website.

The license manager consists of four binaries:

- lmgrd, the core license manager binary
- MLM, the MATLAB vendor daemon
- lmutil, a suite of tools for administering the license manager
- lmtools.exe, a graphical front end for the license manager (Windows only).

2   Manually install the network license manager:

    a   Extract the downloaded folder and place the folder in the desired destination

  b Place the file license.lic (downloaded in the previous step) somewhere you can remember, optimally with lmgrd and MLM.

When you have completed installing the network license manager, go to the next step, Configure Network License. Do not start the network license manager at this time.

Step 3. Configure Network License

For all network license types, configure network and end user license. If using a Network Named User License, set up named users.

1 Open the network license file for editing.

2 The SERVER line identifies the server (host and port number). Add a SERVER line to the top of the license file, as follows:

```
SERVER host hostid port
```

An example of this syntax is:

```
SERVER Server1 0123abcd0123 12345
```

3 The DAEMON line identifies the name of the network license manager daemon. Add the DAEMON line with the name of the network license manager daemon to the next line, as follows:

```
DAEMON MLM <path to MLM.exe>
```

An example of this syntax is:

```
DAEMON MLM $lmroot/etc/glnxa64/MLM
```

`$lmroot` is where you installed the network license manager.

4 Create end user license.

  a Create a license file that points to the license server for each end user installation of RoadRunner.

    i Create a text file, and name it `network.lic`.

    ii From the network license file, copy the `SERVER` line and paste it into the new license file as the first line. This server line specifies the host, hostID, and port of the license server and has the following format:

      SERVER *host hostid port*

      An example of this syntax is:

      ```
SERVER Server1 0123abcd0123 1711
```

      By copying the line rather than recreating it, you are less likely to mistype the information.

    iii For the second line, add:

      USE_SERVER

    iv Save the license file.

  b Put the `network.lic` file on each end user computer, using one of two options.

- Option 1: If the end user will be performing the activation, give the license file to them.
- Option 2: On the end user's computer, put the license file in an accessible folder and set the "MLM_LICENSE_FILE" environment variable to specify the path to the file (or the folder containing it).

5 If you are the administrator on a Network Named User License, complete the following steps:

a Follow the procedure in "Set Up Named User Licensing", in which you will create an options file. This options file is used by the network license manager to identify the specific named users to whom you have assigned right-to-use privileges.

b Make sure the DAEMON line in the license file you downloaded during network license installation contains the correct path to the options file. See "Check the Options File".

Step 4. Start Network License Manager

Start the license manager.

Step 5. Ready to Install RoadRunner

When you have completed updating the network license manager, you can then install RoadRunner software on individual computers.

You have the following options for installing RoadRunner:

- You can install and activate each installation yourself. See "Install and Activate RoadRunner" on page 1-4 and follow all steps in the procedure.
- You can have each end user perform their own installation and activation. Give each user a copy of the network license file (network.lic) and the platform-specific product installer and have them follow all steps in "Install and Activate RoadRunner" on page 1-4. You must set up the network license first; see Set up Network License
- You can install RoadRunner on each computer but have the end user activate it.

  1 "Install and Activate RoadRunner" on page 1-4 (but don't activate the software)

  2 Give the end user a copy of the network license (network.lic) and have them put it on the end user computer where RoadRunner was installed.

  3 Instruct the end user to follow the procedures in "Activate License" on page 1-6.

Note If you are performing a RoadRunner installation for the first time, consider first installing and activating RoadRunner on a test computer. If your test is successful, you can start the individual installations with confidence.

Install New Network License Manager for All Products

Follow this procedure if you are planning to install RoadRunner and other MathWorks products for the first time, but you have not yet installed the network license manager.

Step 1. Install Network License Manager and Other MathWorks Products

For this procedure, before you add the RoadRunner license, you will use the MathWorks product installer to install the network license manager and other MathWorks products.

1. Select one of the following options to install the network license manager:

   - "Install on Server Connected to Internet"
   - "Install on Offline Server"

2. Go to "Install Products" and complete your MATLAB installation. When you have finished, return here.

Step 2. Download and Configure Network Licenses

1. If the network license manager is running, Stop the license manager before continuing.
2. Go to License Center on the MathWorks website and select the RoadRunner network license.
3. On the Install and Activate tab, click **Activate to Retrieve License File** and follow all prompts. For application label, enter "RoadRunner Server."
4. Download or email the license file and save it on the server that is hosting the network license manager.
5. Combine your RoadRunner network license with the MATLAB network license. See How do I serve multiple MATLAB licenses from the same network license manager? in MATLAB Answers.

Step 3. Start Network License Manager

Start the license manager.

Step 4. Ready to Install RoadRunner

When you have completed updating the network license manager, you can then install RoadRunner software on individual computers.

You have the following options for installing RoadRunner:

- You can install and activate each installation yourself. See "Install and Activate RoadRunner" on page 1-4 and follow all steps in the procedure.
- You can have each end user perform their own installation and activation. Give each user a copy of the network license file (`network.lic`) and the platform-specific product installer and have them follow all steps in "Install and Activate RoadRunner" on page 1-4. You must set up the network license first; see Set up Network License
- You can install RoadRunner on each computer but have the end user activate it.

   1. "Install and Activate RoadRunner" on page 1-4 (but don't activate the software)
   2. Give the end user a copy of the network license (`network.lic`) and have them put it on the end user computer where RoadRunner was installed.
   3. Instruct the end user to follow the procedures in "Activate License" on page 1-6.

Note If you are performing a RoadRunner installation for the first time, consider first installing and activating RoadRunner on a test computer. If your test is successful, you can start the individual installations with confidence.

See Also

Related Examples

- "Install and Activate RoadRunner" on page 1-4
- "Update RoadRunner Network Licenses" on page 1-9
- "Get RoadRunner Updates and Upgrades" on page 1-8
- "Update Network License Manager for RoadRunner" on page 1-16

More About

- "Network Named User License Administration"
- "Concurrent License Administration"

External Websites

- License Center

Update Network License Manager for RoadRunner

Overview

Follow the procedures in this topic if you are adding or upgrading RoadRunner software and require an updated network license manager from the same release.

- If you are adding RoadRunner, update the network license manager first.
- If you are upgrading RoadRunner to a newer release, you can perform the network license manager update and the RoadRunner upgrade in either order.

Either way, you must have the new or updated license in place before you restart the network license manager. Instructions are in the procedures described in this topic.

Note You must be a network license administrator to perform these procedures. Updating the network license manager software requires you to stop and restart the license manager.

Select one of these workflows:

- To install RoadRunner with an existing network license manager from MathWorks, see "Update Existing Network License Manager for New RoadRunner Installation" on page 1-16.
- To upgrade RoadRunner to a newer release, and you need to update the network license manager to the same release, see "Update Existing Network License Manager to Upgrade RoadRunner Software" on page 1-17.

Update Existing Network License Manager for New RoadRunner Installation

To add RoadRunner to an existing MATLAB installation, the network license manager version must be the same or higher as the version of RoadRunner you plan to install. For example, if your MathWorks products and network license manager are from R2020a and you want to download and install RoadRunner R2020b, you must update your existing network license manager to the R2020b release.

Step 1. Download RoadRunner Network License

1. Go to License Center on the MathWorks website and select the RoadRunner network license.
2. On the Install and Activate tab, click **Activate to Retrieve License File** and follow all prompts. For application label, enter "RoadRunner Server."
3. Download or email the license file and save it on the server that is hosting the network license manager.

Step 2. Configure Network License

1. Combine your new RoadRunner network license with the existing MATLAB network license. See How do I serve multiple MATLAB licenses from the same network license manager? in MATLAB Answers.
2. If you have a Network Named User license and are adding users for RoadRunner, modify the options file to include the additional users.

3   If you have a Concurrent license and have added new seats for RoadRunner, create a license file for each new machine that will be running RoadRunner by copying the license file you already created for your other RoadRunner installations..

Step 3. Update Network License Manager

Follow the instructions in "Update Network License Manager Software". This procedure requires you to stop and restart the network license manager.

Step 4. Ready to Install RoadRunner

When you have completed updating the network license manager, you can then install RoadRunner software on individual computers.

You have the following options for installing RoadRunner:

- You can install and activate each installation yourself. See "Install and Activate RoadRunner" on page 1-4 and follow all steps in the procedure.
- You can have each end user perform their own installation and activation. Give each user a copy of the network license file (`network.lic`) and the platform-specific product installer and have them follow all steps in "Install and Activate RoadRunner" on page 1-4. You must set up the network license first; see Set up Network License
- You can install RoadRunner on each computer but have the end user activate it.

  1   "Install and Activate RoadRunner" on page 1-4 (but don't activate the software)

  2   Give the end user a copy of the network license (`network.lic`) and have them put it on the end user computer where RoadRunner was installed.

  3   Instruct the end user to follow the procedures in "Activate License" on page 1-6.

Note If you are performing a RoadRunner installation for the first time, consider first installing and activating RoadRunner on a test computer. If your test is successful, you can start the individual installations with confidence.

Update Existing Network License Manager to Upgrade RoadRunner Software

Follow these procedures if you are updating the network license manager so that you can upgrade your RoadRunner release.

1   If you have received an updated RoadRunner network license, complete the steps in "Update RoadRunner Licenses" on page 1-9 first.

2   Stop the license manager.

3   Download the MathWorks-specific license manager daemons from License Manager Files on the MathWorks website.

4   Manually install the network license manager by extracting the downloaded folder and placing the folder where you had the previous network manager files.

5   Start the license manager.

You can now upgrade the software. See "Upgrade RoadRunner Release" on page 1-8.

See Also

Related Examples
- "Install and Activate RoadRunner" on page 1-4
- "Update RoadRunner Network Licenses" on page 1-9
- "Get RoadRunner Updates and Upgrades" on page 1-8

More About
- "Network Named User License Administration"
- "Concurrent License Administration"

External Websites
- License Center

Create Simple RoadRunner Scene

RoadRunner is an interactive editor that lets you design 3D scenes for simulating and testing automated driving systems. This example shows how to create a simple scene containing an intersection, bridges, and trees in the surrounding terrain, similar to the scene shown here:

Prerequisites

Before beginning this example, make sure that your system meets these prerequisites:

- You have downloaded, installed, and activated RoadRunner by following the instructions described in "Install and Activate RoadRunner" on page 1-4.

- You have a license for the "RoadRunner Asset Library Add-On". This example uses assets that are available only in this library.

Although this example covers some basic camera operations, for a more complete understanding of how the RoadRunner camera works, consider reviewing the "Camera Control in RoadRunner" on page 1-44 example first.

Create New Scene and Project

In RoadRunner, each scene you create is part of a project, which is a folder of assets (scene components) that can be shared across all scenes in that project. Create a new scene and a new project in which to put that scene.

1   Open RoadRunner, and from the start page, click **New Scene**.

2   On the Select a Project window, click **New Project**.

3   In your file system, browse for an empty folder in which to create the project. If an empty folder does not exist, create one and name it `My Project`. The folder name becomes the name of the project.

4   When prompted, click **Yes** to install the RoadRunner Asset Library in your project.

RoadRunner opens to a new scene with an empty scene editing canvas.

The name of the project that you specified appears in the title bar. The name of the scene also appears in the title bar, but it is displayed as **New Scene** until you save the scene and give it a name.
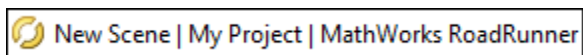
You can create a new scene, change scenes, or change projects at any time from the **File** menu. When you reopen RoadRunner, you can select recent scenes that you worked on from the start page, in the **Recent Scenes** list.

Add Roads

When you open a new scene, RoadRunner opens with the **Road Plan Tool** selected. Instructions on using this tool appear in the bottom status bar. By right-clicking in the scene editing canvas with this tool selected, you can add control points that shape the geometry of a road.

1   At the bottom center of the scene editing canvas, right-click to add the first control point of a new road.
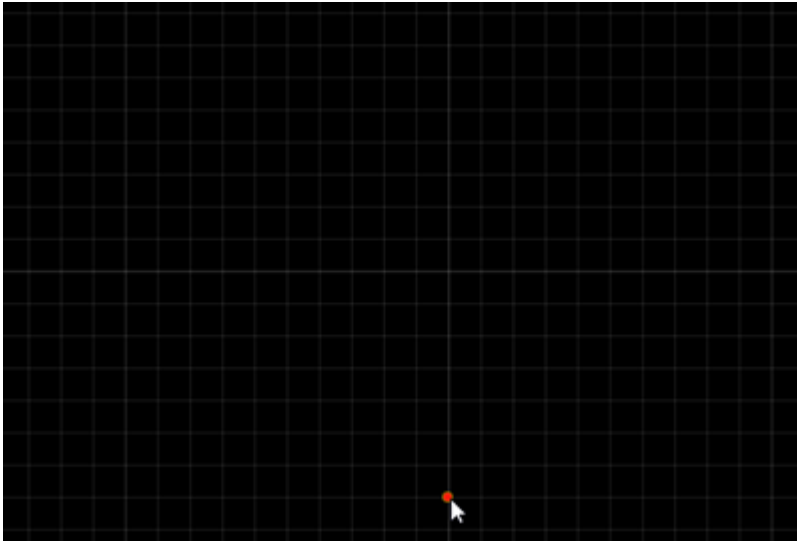
2 At the top center of the canvas, right-click to add a second control point and form your first road segment.
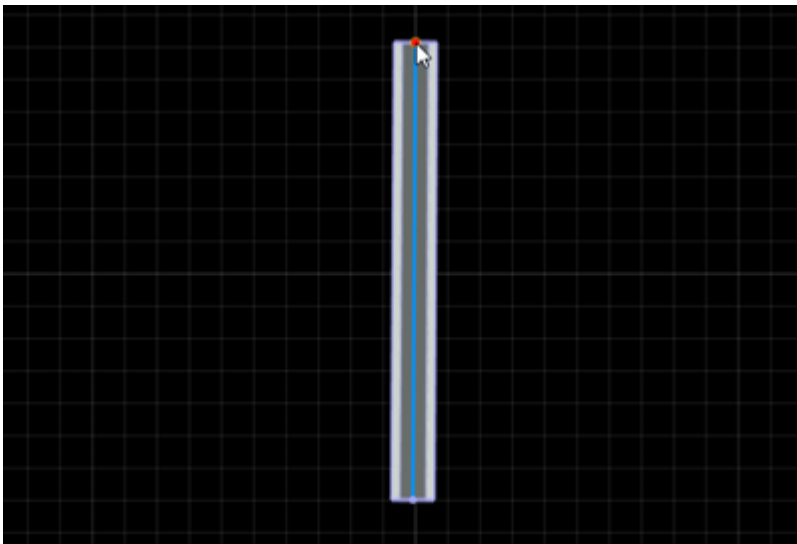
3 Click away from the road to deselect the road and finish creating it.
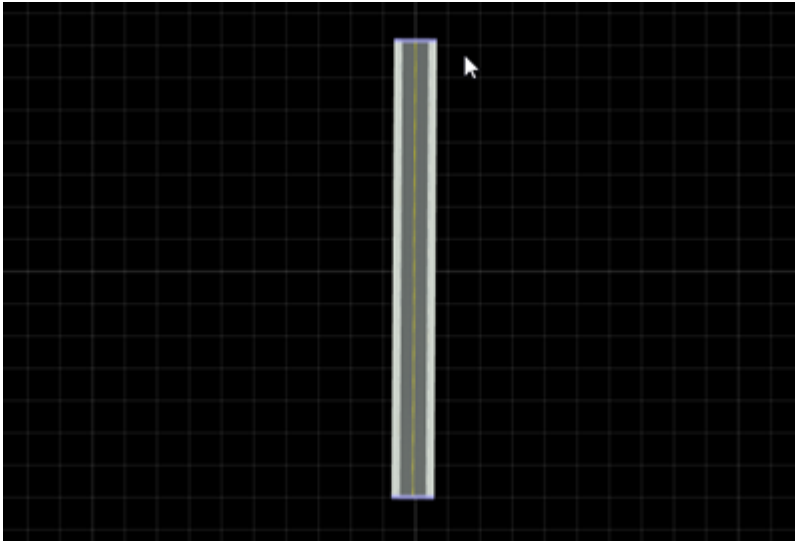
4   Create a new straight road that intersects the first road by right-clicking to its left, right-clicking to its right, and then clicking away from the road. The two roads form a junction.
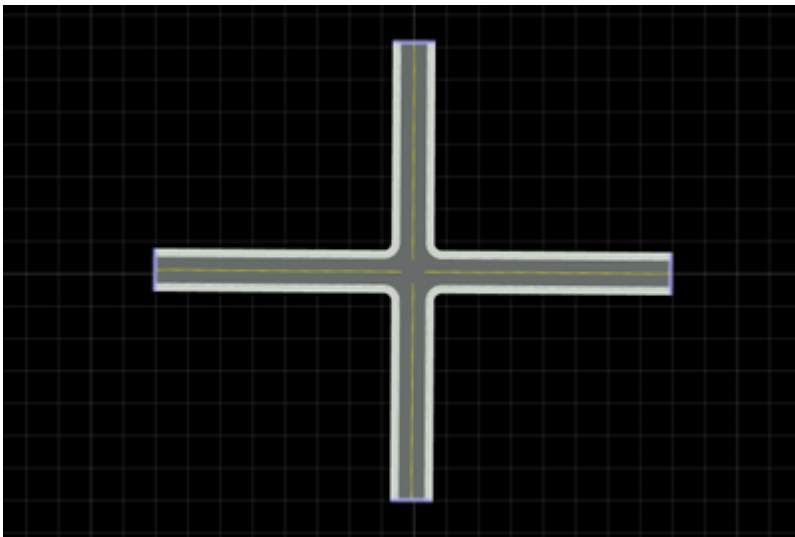
So far you have created straight roads. To form curved roads, right-click multiple times to add additional control points to a road. Create a curved road that overlaps the intersection.

1   Right-click within the top-left quadrant of the intersection.
2   Right-click within the top-right quadrant of the intersection. The first created road segment is straight.
3   Right-click in the bottom-right quadrant of the intersection. The area enclosed within the intersection and the curved road forms a ground surface.
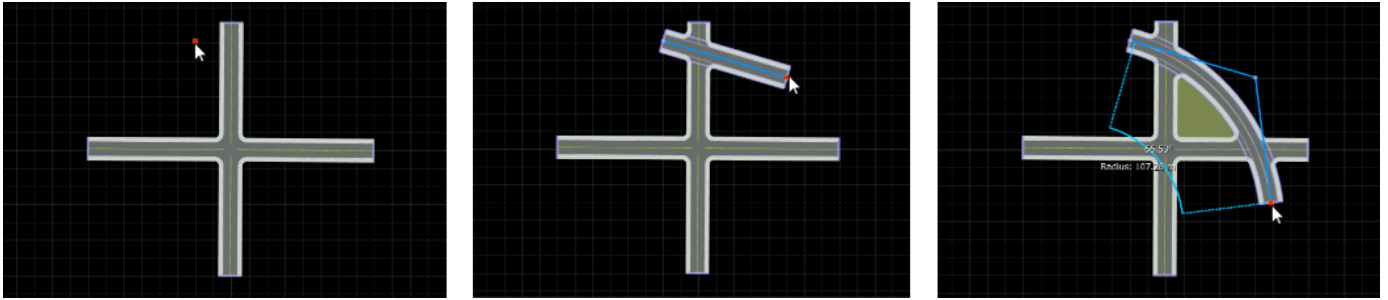
You can extend existing roads by selecting the endpoint of a road and right-clicking to add more control points.

1 In the curved road you created, click to select the end near the top of the canvas.

2 Right-click the left end of the intersection. RoadRunner creates a road that meets the necessary geometric constraints. The enclosed area again forms a ground surface.
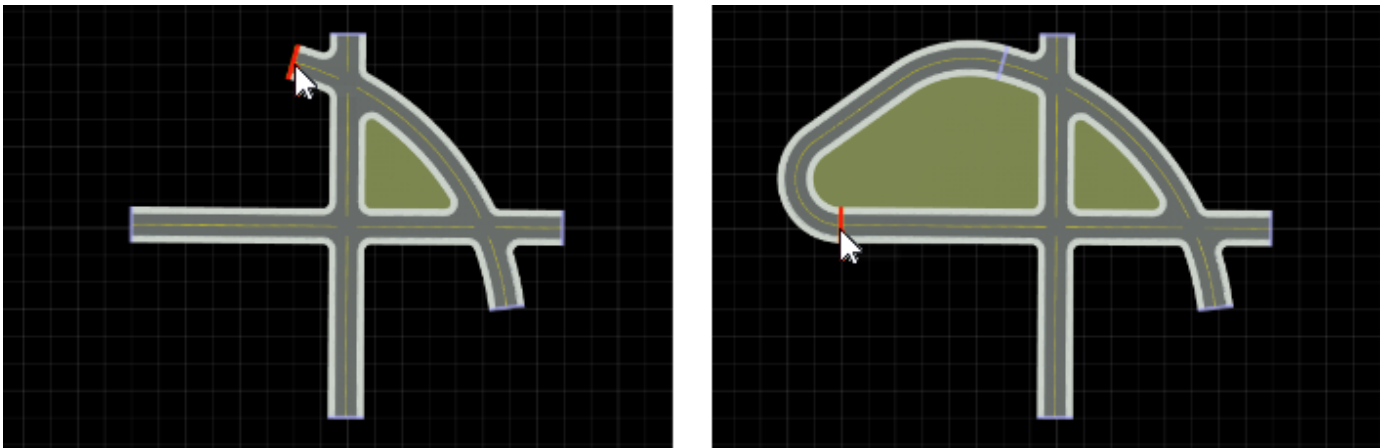
To modify any road, click to select it and try dragging its control points or moving the entire road. You can also right-click a road to add additional control points. For example, in this road network, you can add control points to smooth out the curve on the left side of the intersection.

Add Surface Terrain

So far, only the areas enclosed by roads contain surface terrain. To add surface terrain around the

entire road network, you can use the **Surface Tool** .

1 In the toolbar, click the **Surface Tool** button. Selecting a new tool puts RoadRunner in a different mode that enables new interactions and makes different scene objects selectable. With the **Surface Tool** selected, the roads are no longer selectable but the road surface nodes become selectable.
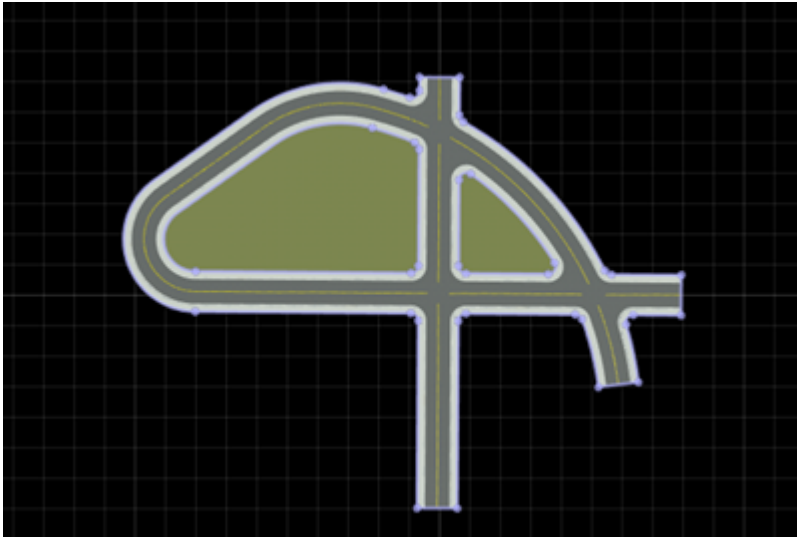
2   Zoom out of the scene, either by using the scroll wheel or by holding **Alt** and right-click and then dragging down or left.

3   Right-click above the road network to add a new surface node. Then, keep right-clicking at points around the road to form a circle. When you reach the top node again, right-click it to connect the surface graph and commit the surface to the canvas.
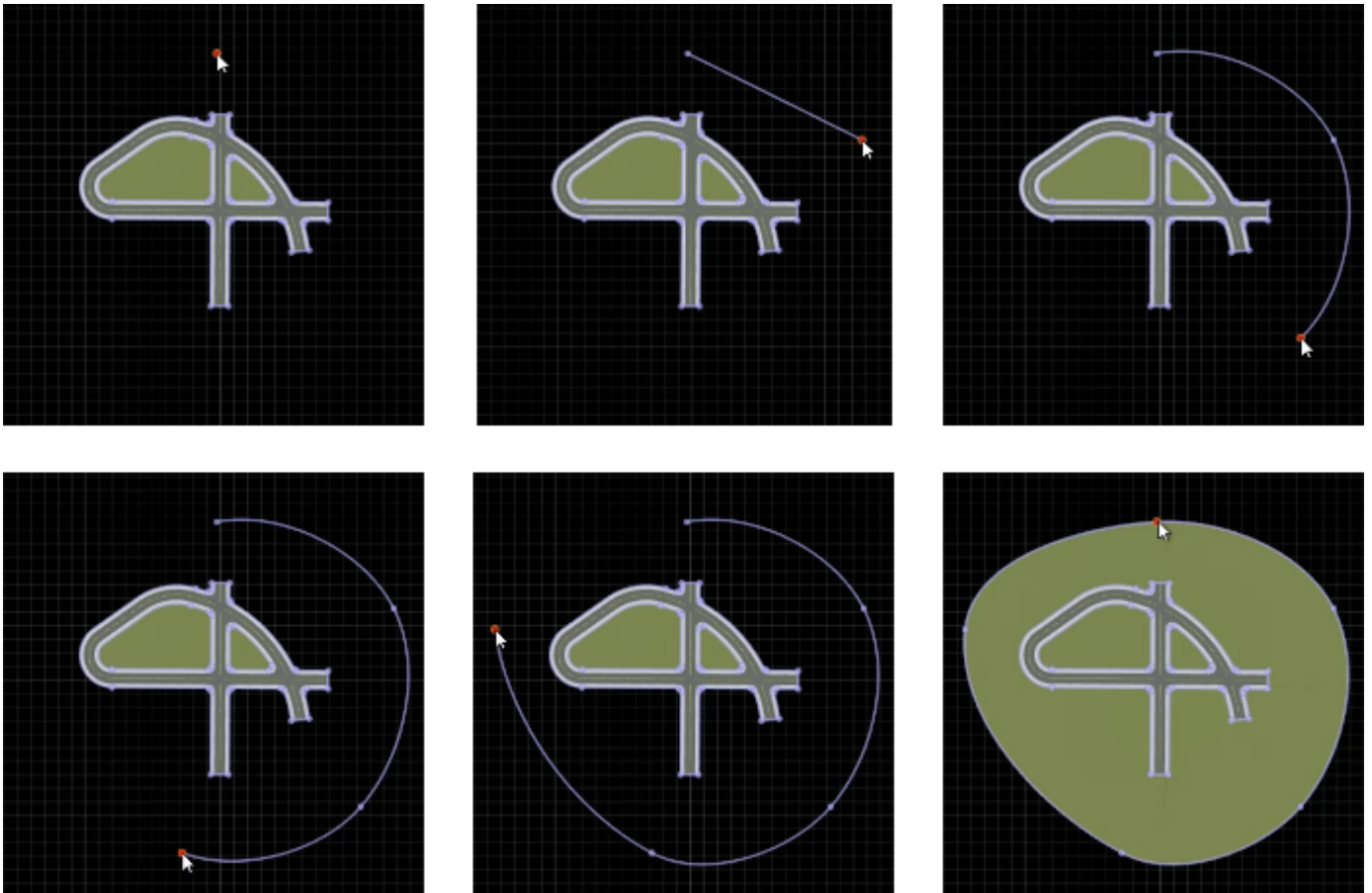
To modify the surface size, click and drag the surface nodes. To modify the curve of the surface, click the segments between nodes, and then click and drag the tangent lines.

Add Elevation and Bridges

Up to this point, the scene has been flat. Modify elevation in the scene by changing the height of one of the roads.

1   Hold **Alt** and then click and drag the camera to view the scene at an angle.
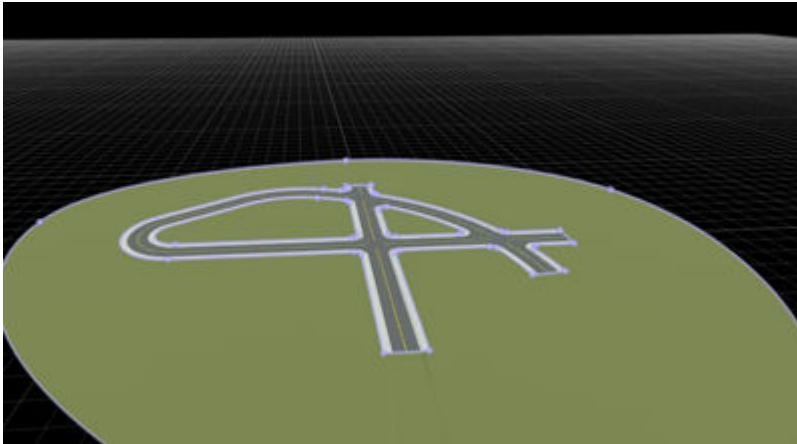
2   Click the **Road Plan Tool** button to make roads selectable again. Then, click to select the first curved road that you created.
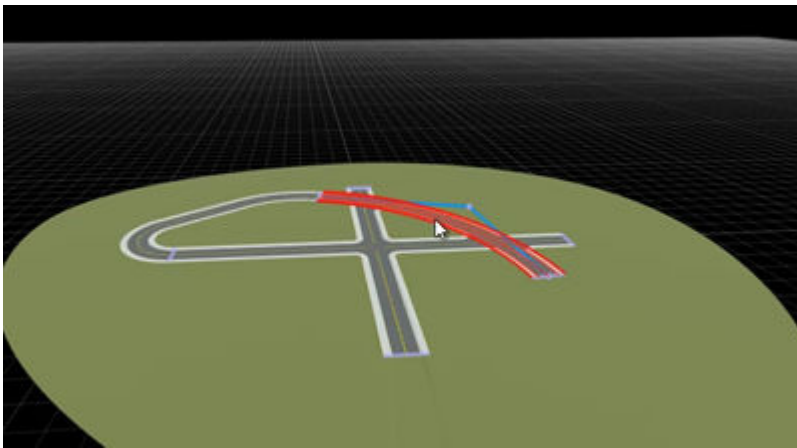
3   To elevate the road, use the **2D Editor**, which enables you to view scene aspects such as road profiles and road cross-sections. In the **2D Editor**, select the road profile and raise it approximately 10 meters.
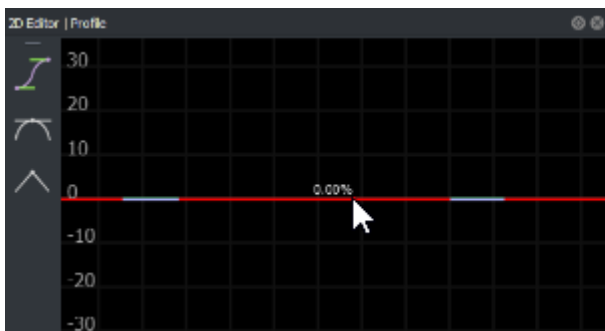
The road is now elevated in the scene canvas above the intersection. Instead of forming junctions, the elevated road forms overpasses.
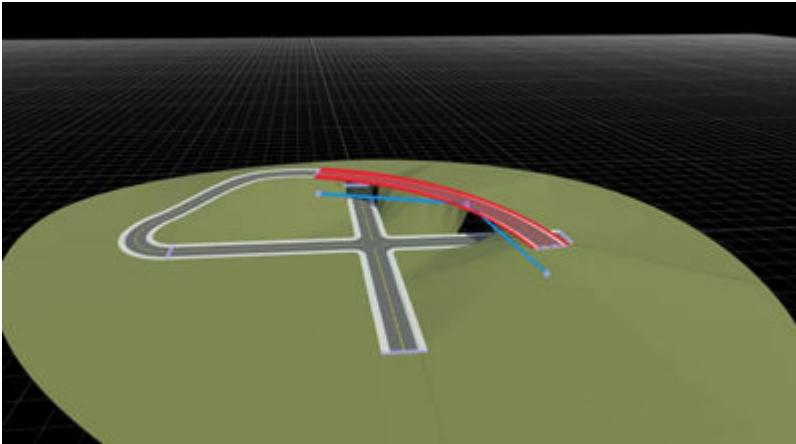
Roads attach to the surface terrain. When you elevate a road, the terrain elevates with it. Increasing elevation can lead to visual artifacts below the overpasses. To resolve this issue, you can create

bridge spans by using the **Road Construction Tool** .

1 Rotate the camera and zoom in to see the visual artifacts at the overpasses.
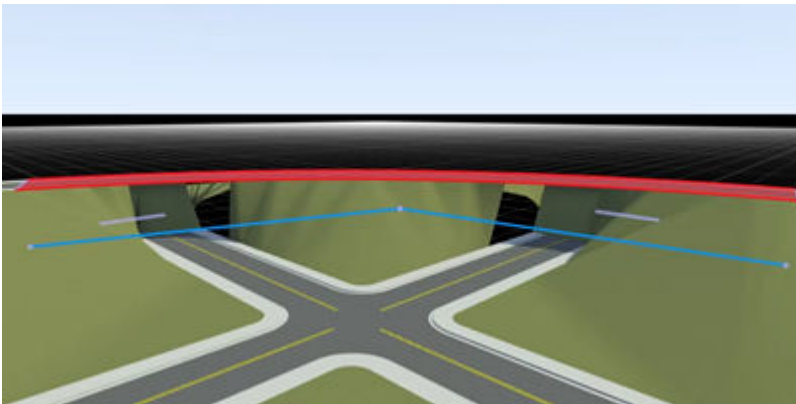
2 Click the **Road Construction Tool** button.

3
On the left toolbar, click the **Auto Assign Bridges** button . This operation, which is available only when you are using the **Road Construction Tool**, converts only those road sections that are directly above a region to bridge spans. Use the default bridge span inflation and click **OK**. The roads spans are converted to bridges and the visual artifacts are removed.
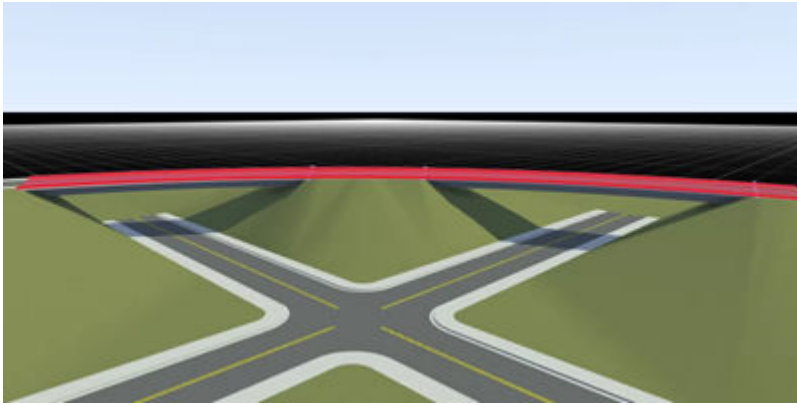
If the bridges do not form correctly, try adjusting the road elevation or the bridge span inflation and rerun the **Auto Assign Bridges** operation.

Modify Junction

Some tools enable you to select and modify properties at junctions. Modify the corner radius of the four-way intersection.

1. Click the **Corner Tool** button , and then click to select the four-way intersection.
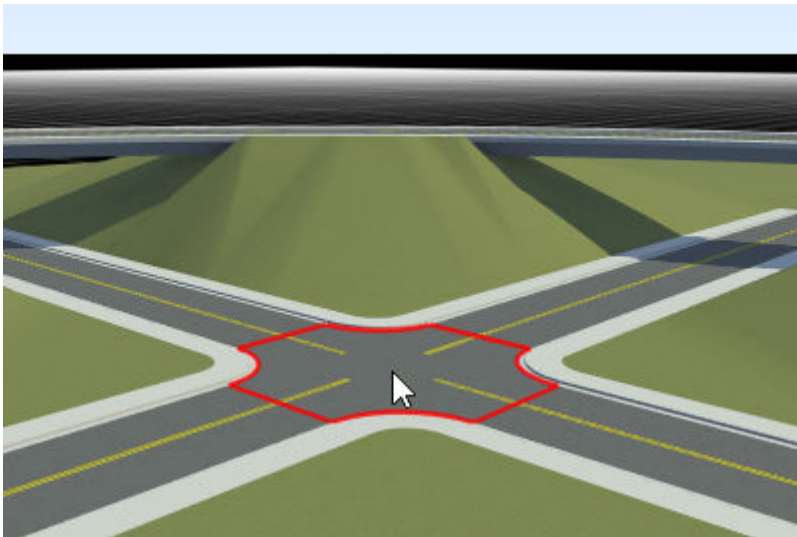
2. By default the junction has a corner radius of 5 meters. Increase this value by using the **Attributes** pane. This pane contains information and editable attributes about currently selected items. In the **Corner Tool**, selecting the junction selects all four corners of the junction, so you can modify the attributes of all four corners at the same time.

   In the **Attributes** pane, set the **Corner Radius** attribute of all four corners to 10.
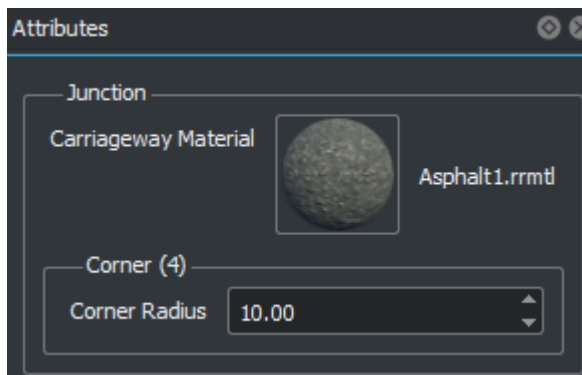
The junction corners expand in the scene editing canvas.
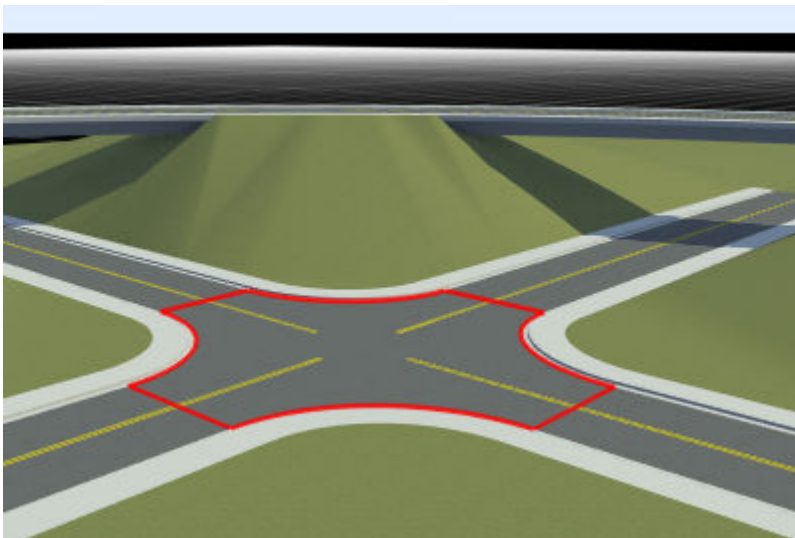
Alternatively, you can modify the **Corner Radius** attribute value by clicking on the attribute name
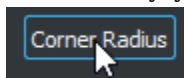
 and dragging up or down.

Add Crosswalk

Add a crosswalk to the intersection.

1   Rotate the camera to view the intersection from the top down. To focus the camera on the selected intersection, press the **F** key.
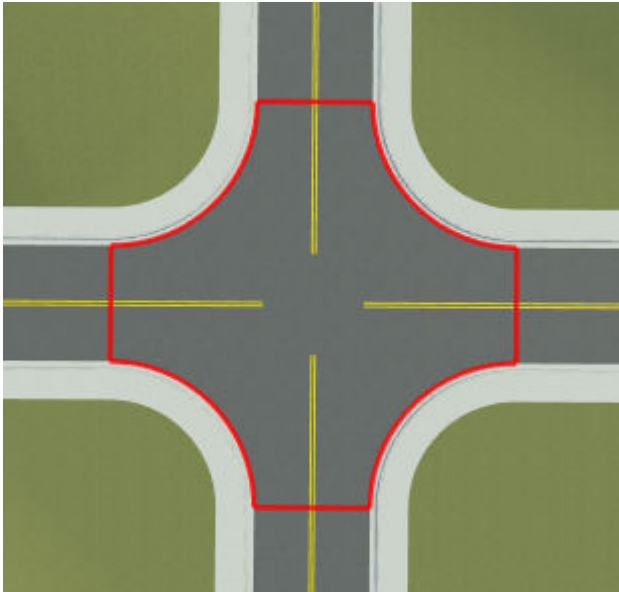
2

Click the **Crosswalk and Stop Line Tool** button . The intersection displays blue chevrons for adding stop lines to the intersection.
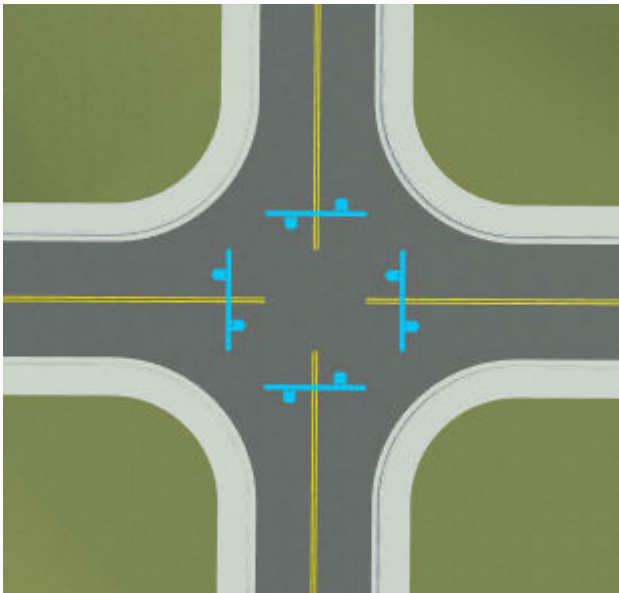
3 From the **Library Browser**, select a crosswalk to add to the intersection. The **Library Browser** stores all assets available to add to a scene. Assets include 3D objects, markings, textures, and materials.

In the **Library Browser**, select the Markings folder, and then select the ContinentalCrosswalk asset. A preview of the asset displays in the asset viewer.
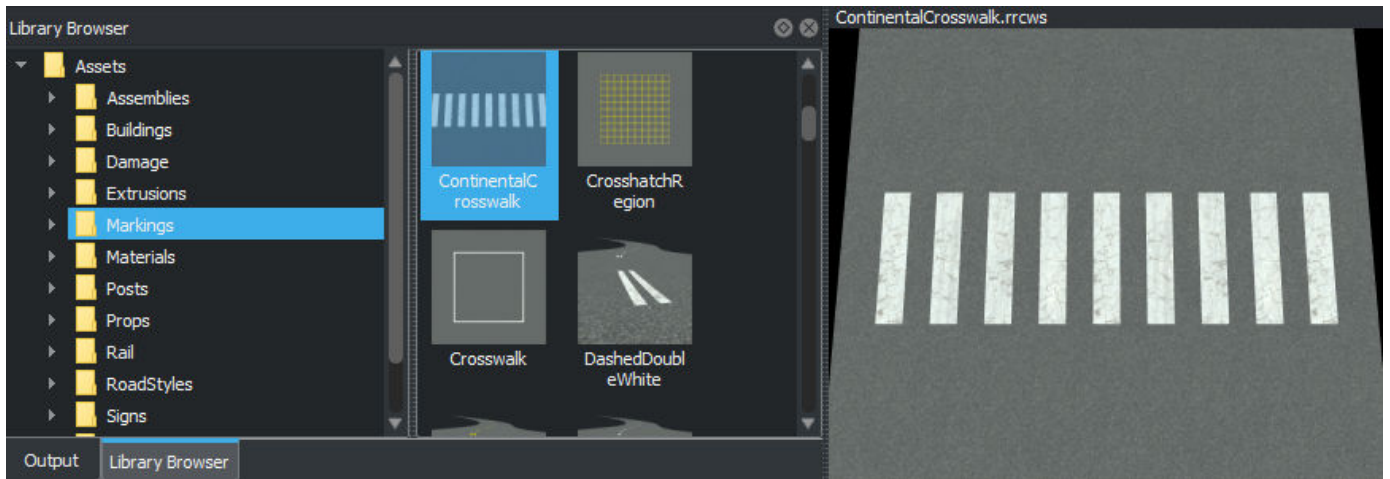
4 Click within the intersection to clear the blue chevrons. Then, right-click in the intersection to apply the selected crosswalk asset to the intersection.
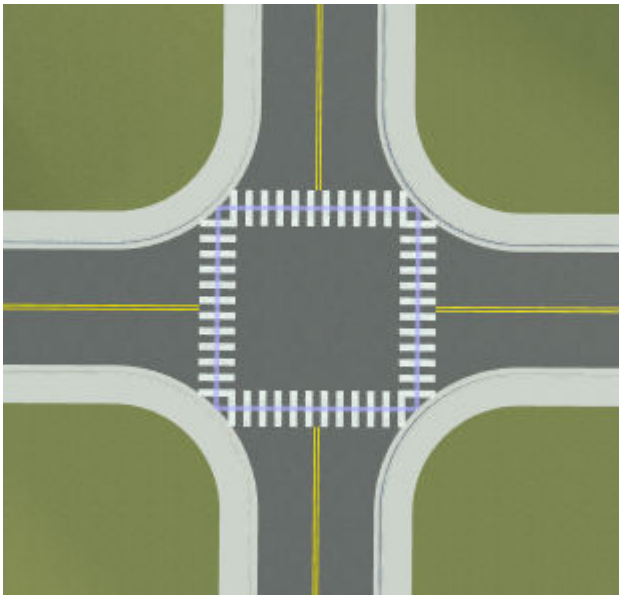
Add Turning Lanes

Convert one of the roads at the intersection into a more complex highway road that includes a turning lane with arrow markings.

Change Road Style

The existing roads all use the default road style, which is of a simple two-lane divided highway with sidewalks. Update one of the roads at the intersection to use a road style with additional lanes.

1 Zoom out and rotate the camera to view the scene at an angle similar to the one shown here.
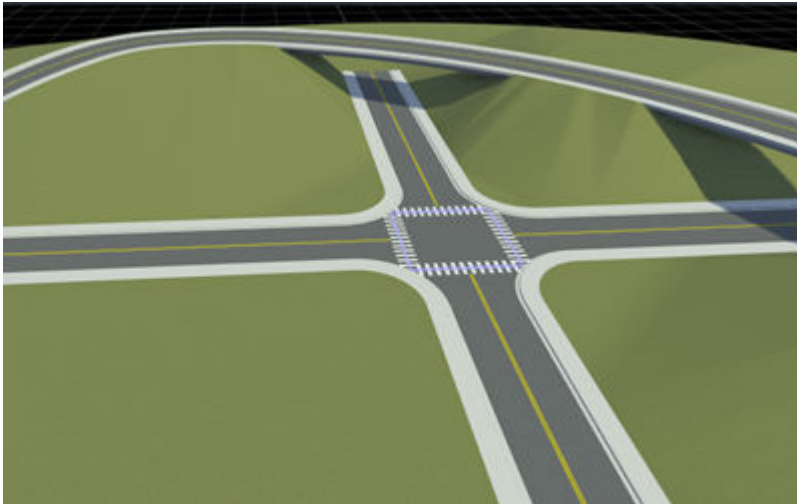
2   In the **Library Browser**, open the `RoadStyles` folder, and then select the `MainStreetCenterTurn` asset. This road style asset includes shoulder lanes, two passing lanes on each side, and a median lane. Optionally, rotate and move the camera in the asset viewer to inspect the road style.
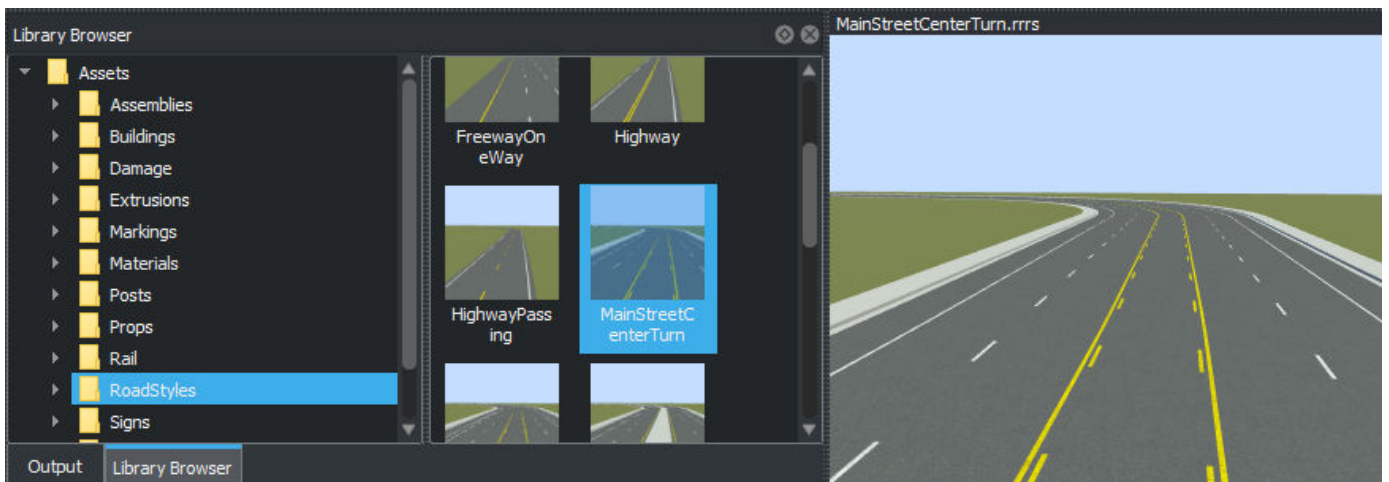
3   Drag the selected road style onto the road closest to the camera, as shown here. The road updates to the new style and switches back to the **Road Plan Tool**. The road maintains the corner radius and crosswalk style previously applied.
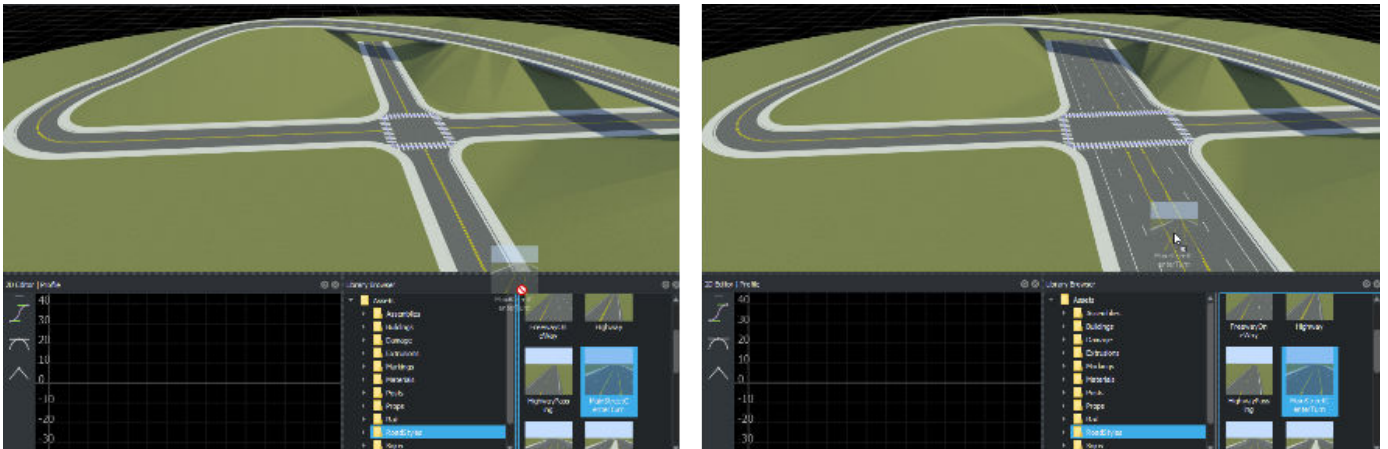
Create Turning Lane at Intersection

Create a short left-hand turn lane near the intersection.

1 Rotate the camera and zoom in near the crosswalk on one side of the road that has the new road style.
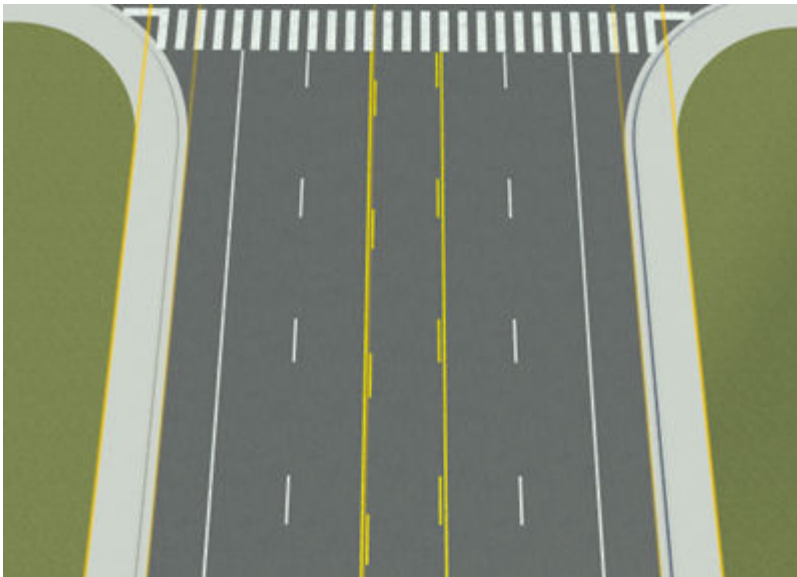
2 Click the **Lane Carve Tool** button . This tool enables you to create a tapering cut in an existing lane to form a turning lane.

3 Click to select the road. Then, right-click the right side of the median lane where you want the tapering cut to start. Drag the blue line diagonally to the left side of the median lane where you want the tapering cut to end and the turning lane to start.
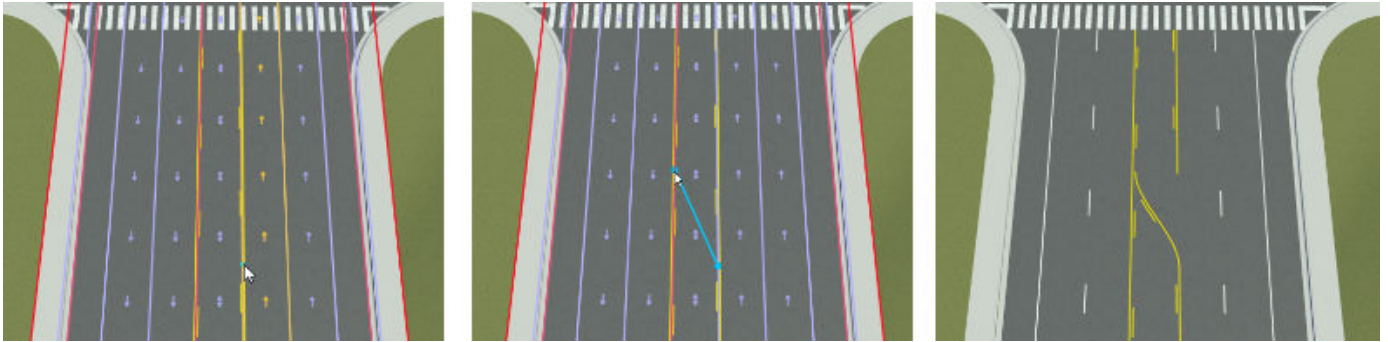
4   The newly formed turning lane still has the styles of the median lane. Update the lane markings to match the style of a standard turning lane.

    a   In the **Library Browser**, select the `SolidSingleWhite` asset and drag it onto the right side of the turning lane. The lane marking changes to a solid single white line.
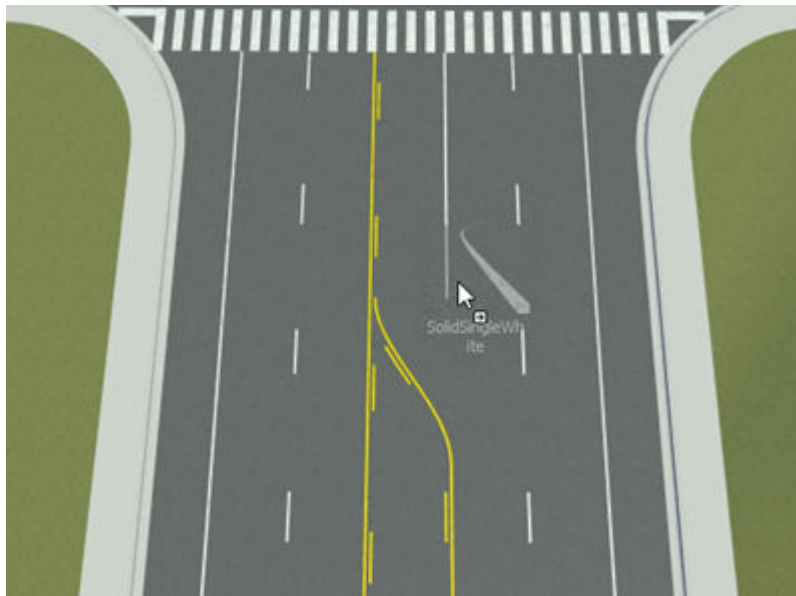
    b   Select the `SolidDoubleYellow` asset and drag it onto the two marking segments that form the left side of the turning lane. The lane marking segments change to solid double yellow lines.
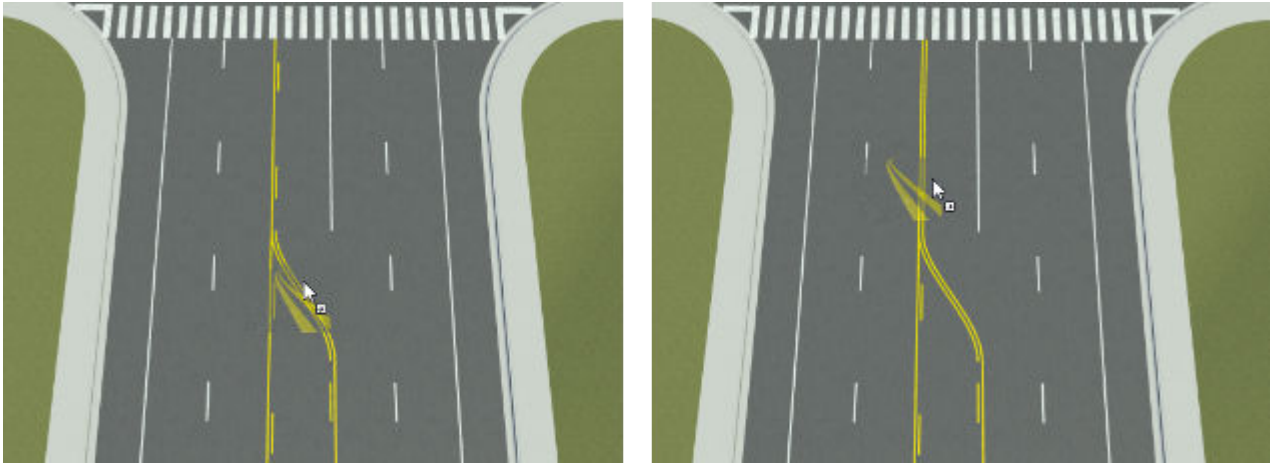
5 Add a turning arrow to the lane. In the `Stencils` folder of the **Library Browser**, select the `Stencil_ArrowType4L` asset. Drag this asset into the turning lane at the point where you want to add the arrow stencil.
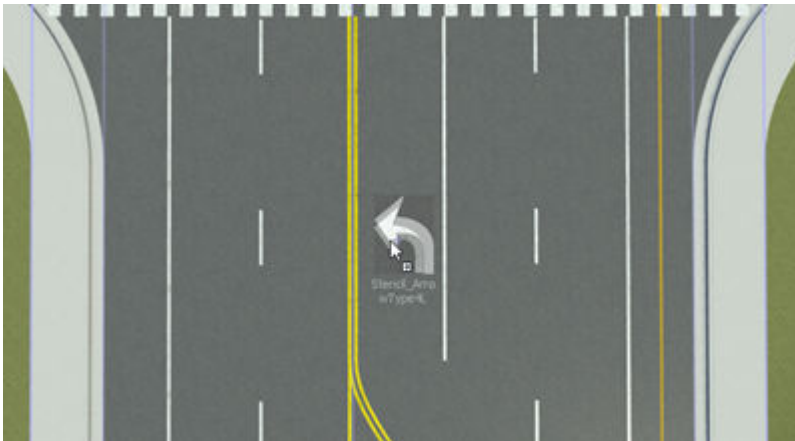
6

By adding the arrow stencil, RoadRunner selects the **Marking Point Tool** to make it the active tool. You can now add the second arrow by right-clicking at the point where you want to add it.
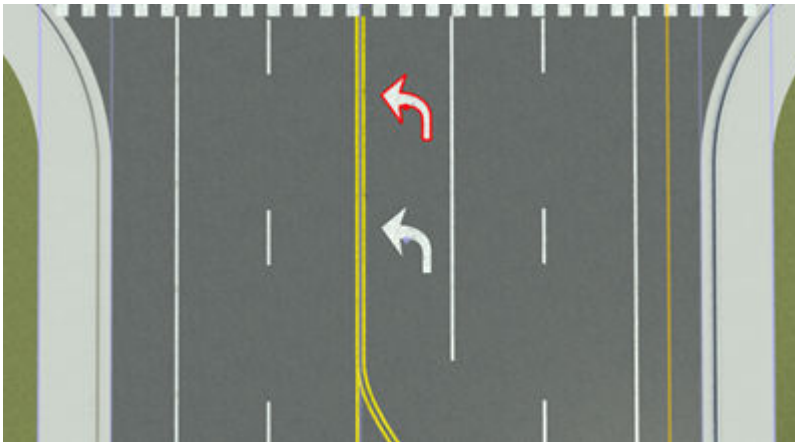
7 Modify the marking material of the arrows to make them appear more worn. First, select the two arrows. In the `Markings` folder of the **Library Browser**, select the `LaneMarking2` material asset. Then, drag this asset into the **Attributes** pane for the selected arrows and over the existing `LaneMarking1` material asset.
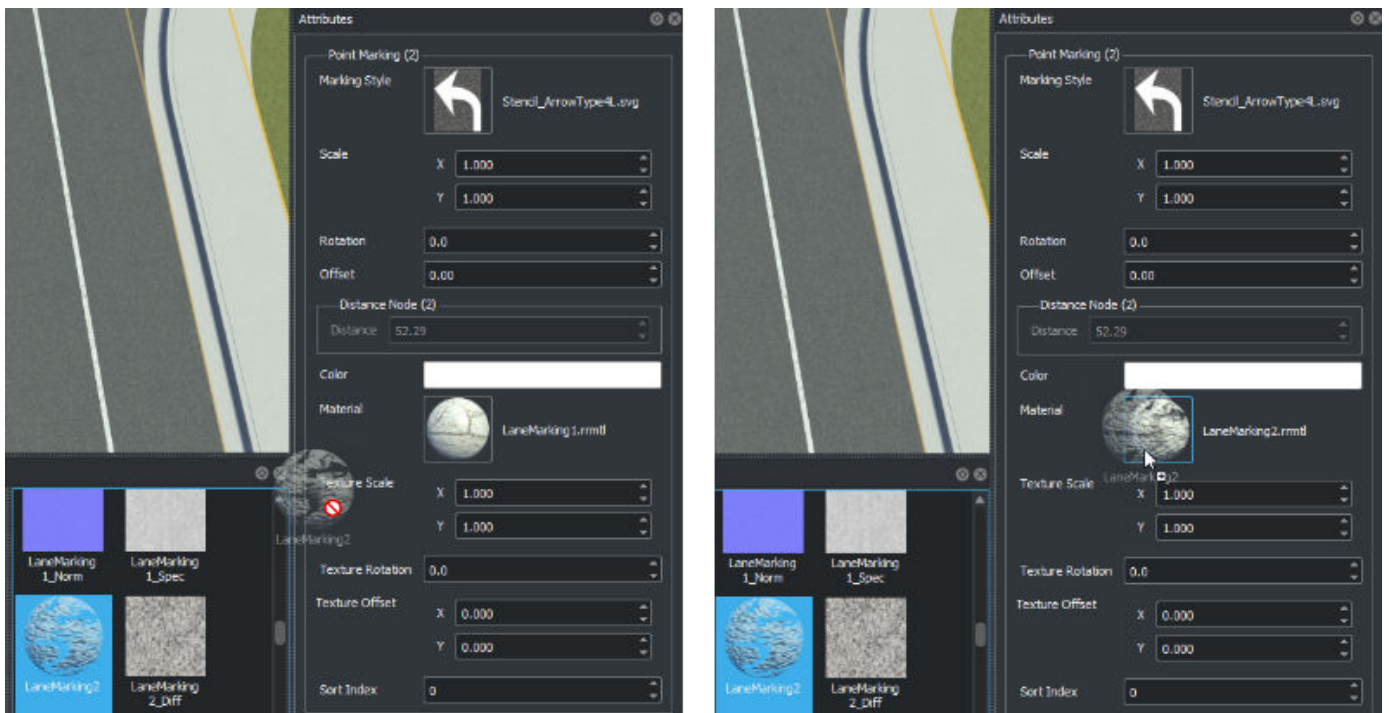
The arrows update to use the new more worn-looking material.
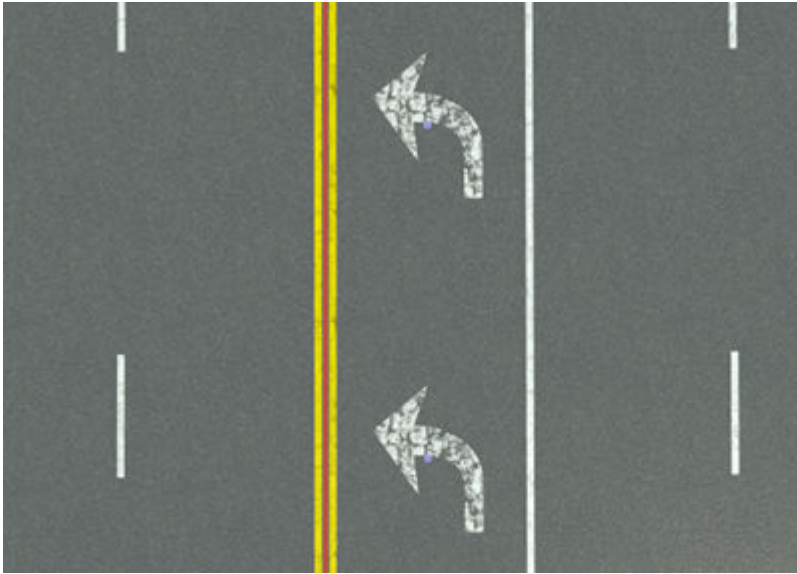
Repeat these steps to create the turning lane on the other side of the intersection.

Add Props

To enhance the scene with more detail, add props to it. Props are 3D objects such as posts, poles, and signs that you can place on and around roads. Add tree props around the road using multiple techniques.

Add Individual Props

Add bushes to one section of the terrain.

1   Zoom out and rotate the camera to fit the entire road network and surrounding terrain in view.
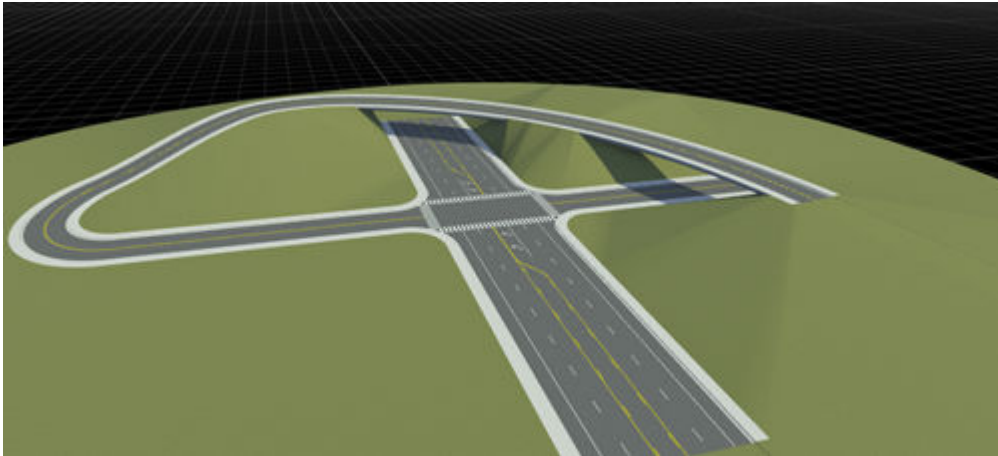
2   In the **Library Browser**, open the `Props` folder and select the `Trees` subfolder.

3   Select a bush prop (one of the asset files that begins with `Bush_`). Drag the bush onto a section

of the scene. RoadRunner switches to the **Prop Point Tool** . Drag additional bushes into the scene or right-click to add more bushes. All the bushes are aligned with the surface terrain.
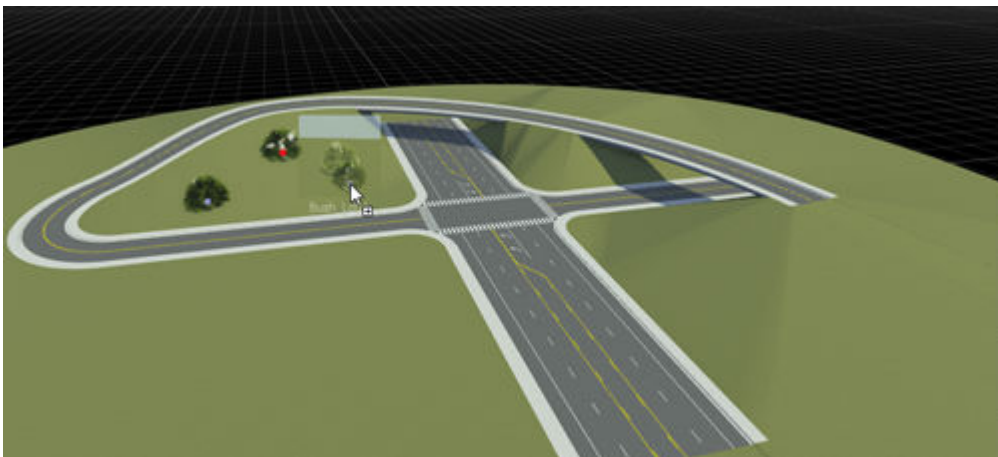
Add Props Along Curve

Add props along a curve to follow the edge of the road.

1

Click the **Prop Curve Tool** button .

2   In the **Library Browser**, in the `Trees` folder, select a California palm tree prop (one of the asset files that begins with `CalPalm_`).

3   Right-click along the road edge of one side of the intersection to add a line of palm trees to it. Click away from the prop curve to complete the line.
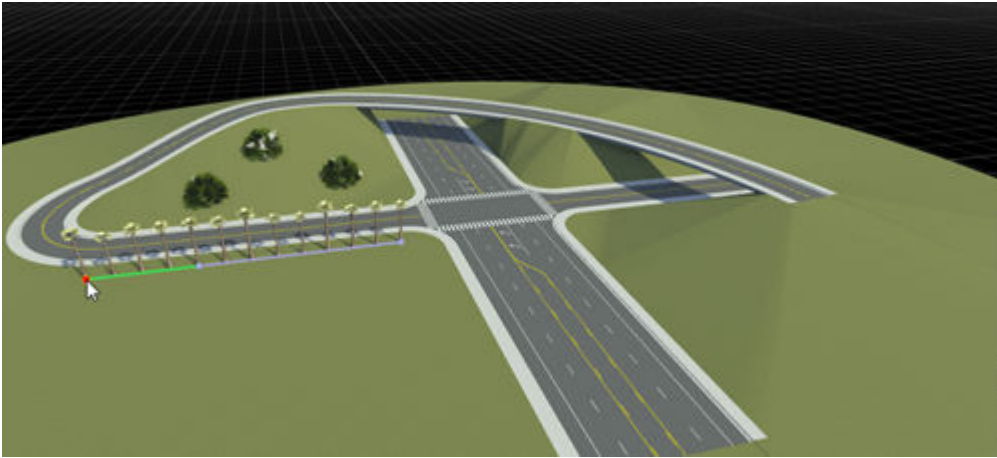
4   To make each individual tree in the span moveable and selectable, you can convert the curve to individual props. Select the prop curve, and in the **Attributes** pane, click **Bake**. The palm trees become individual props and RoadRunner switches to the **Prop Point Tool**. Move some of the palm trees to the other side of the intersection.
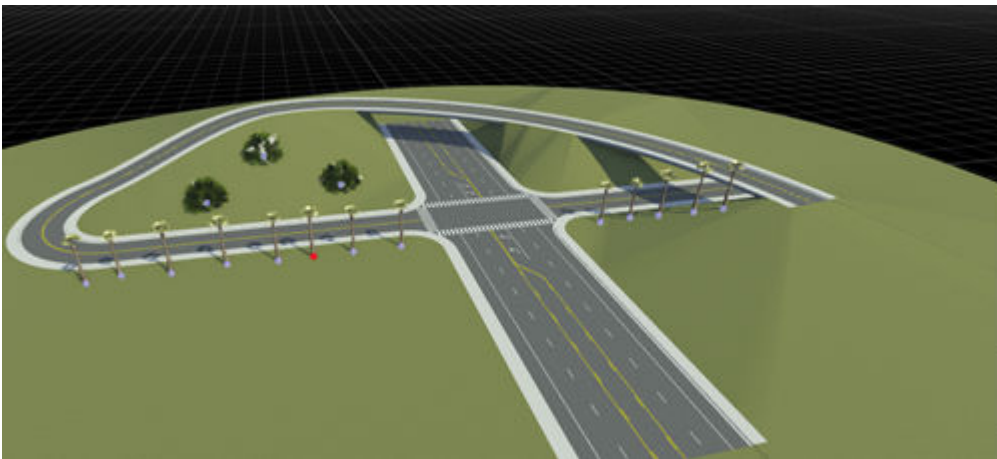
Alternatively, to add a prop along a span of road, you can click the **Prop Span Tool** button , select a road, and drag the prop onto the road edge.

Add Props in Specified Area

Add props in a specified area of the ground surface.

1

Click the **Prop Polygon Tool** button .

2   In the **Library Browser**, in the Trees folder, select a cypress tree prop (one of the asset files that begins with Cypress_).

3   Right-click within an empty area of the surface terrain to draw a polygon containing the selected prop. Click away from the polygon to finish drawing it. Then move the points or tangents to change the shape of the polygon.
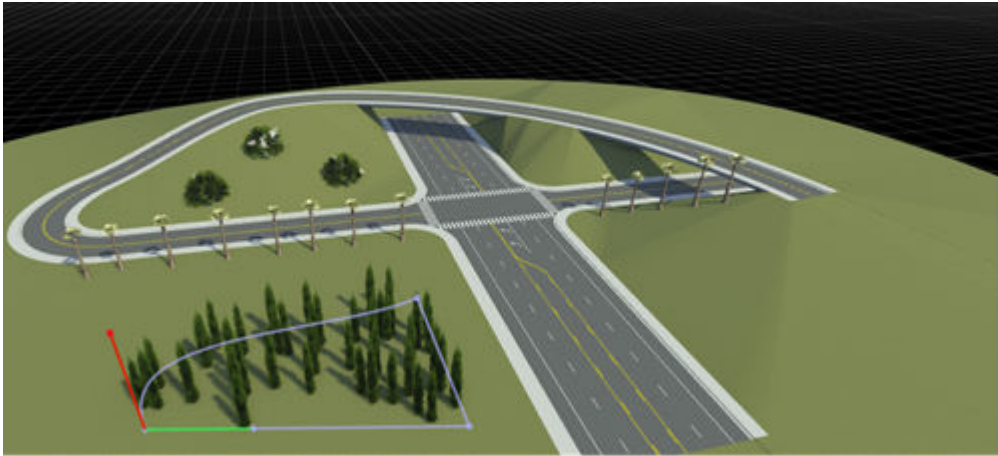
4   Optionally, modify the prop polygon by using the attributes in the **Attributes** pane. For example, to increase or decrease the number of props in the polygon, use the **Density** attribute. To randomize the distribution of assets in the polygon, click **Randomize**.

Add Props of Varying Types

So far you have added a single type of prop to the scene. To add a variety of props to a scene simultaneously, you can create a prop set.

1   In the **Library Browser**, in the Trees folder, hold **Ctrl** and select the three props you added to the scene in the previous sections.

2   Select **New**, then **Prop Set** and give the prop set a name. The new prop set is stored in the Trees folder. The **Attributes** pane displays the three props in the set and a preview of the prop set.
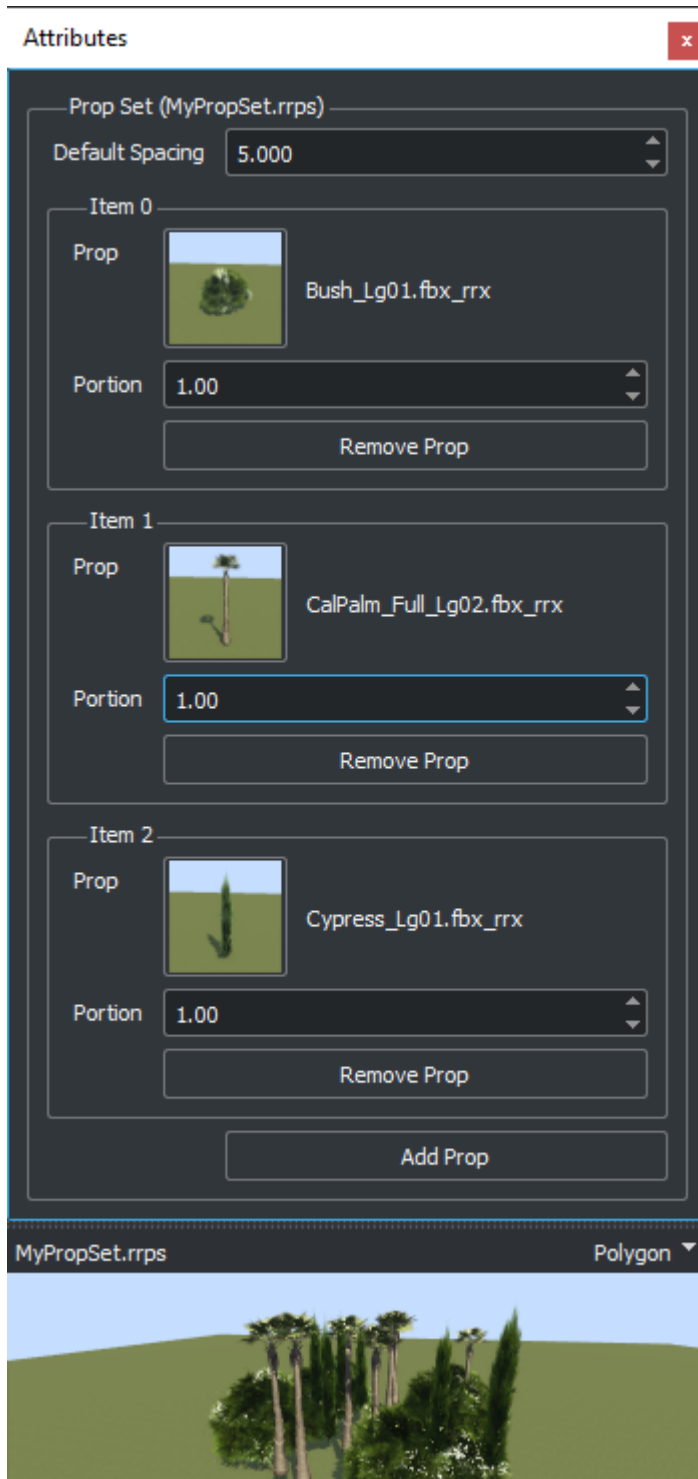
3 Click the **Prop Polygon Tool** button. Create a prop polygon on an empty part of the terrain that contains the new prop set.
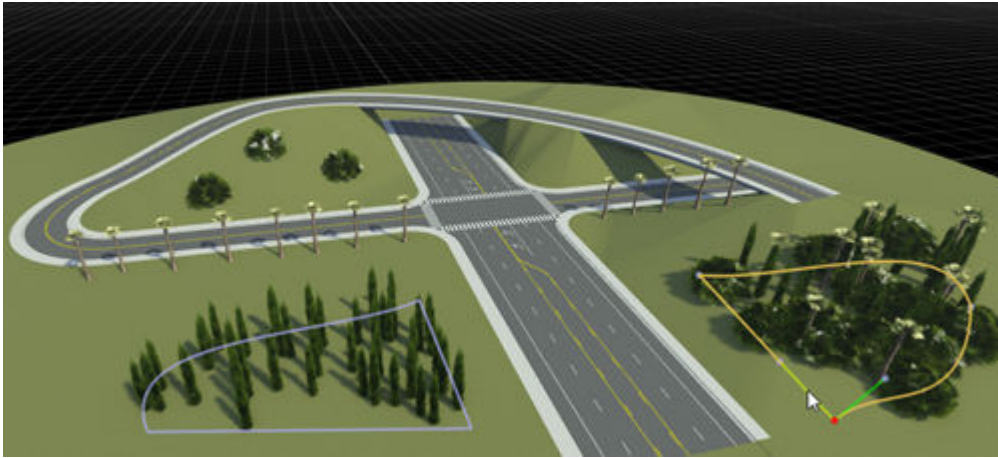
Optionally, you can also replace the existing cypress tree props with the new prop set by dragging the prop set onto the polygon of cypress trees.

Other Things to Try

You have now created a simple road network containing a realistic turning lane, multiple overpasses, and trees of varying types.

You can now enhance the scene using additional tools. For example, try these things:

- Add more roads or connect the existing roads in the scene. To smooth the transitions between roads that have different numbers of lanes, use lane tools such as the **Lane Tool**, **Lane Width Tool**, **Lane Add Tool**, or **Lane Form Tool**.

- Add traffic signals to the intersection by using the **Signal Tool**. To modify the paths through the lanes at each turn signal, use the **Maneuver Tool**. For an example, see "Create Traffic Signals at Junctions" on page 1-64.

- Add additional props to the scene, such as barrels, buildings, and traffic signs. To modify the text of signs, use the **Sign Tool**.

In addition, you can try exporting the scene to one of the supported export formats. These export options are on the **File** menu, under **Export**. To customize export options before exporting, use the

Scene Export Preview Tool. If you are exporting to ASAM OpenDRIVE, use the **OpenDRIVE Export Preview Tool**. This image shows how the export preview of the scene you created looks when you click the **OpenDRIVE Export Preview Tool** button .
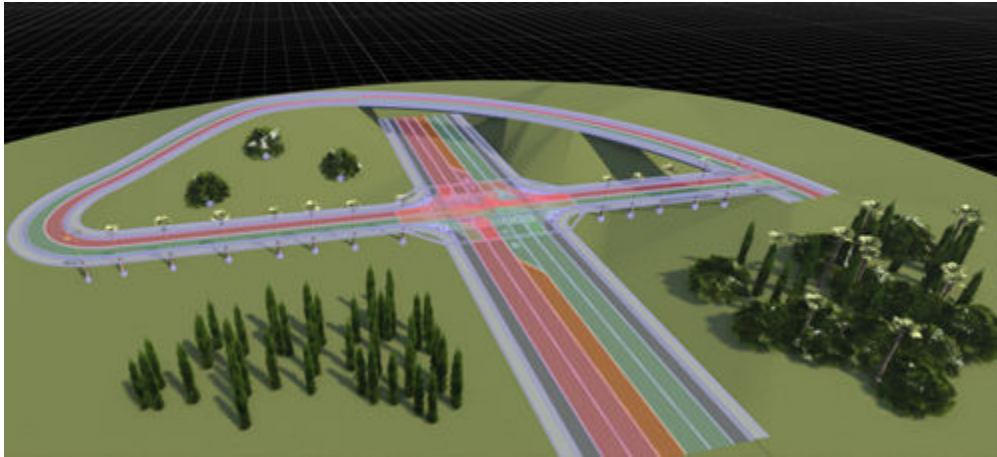
If you want to create a new scene that is based on a real-world location, then you can import geographic information system (GIS) data such as aerial imagery into RoadRunner and create scenes around it. For an example, see "Create Roads Around Imported GIS Assets" on page 1-57.

See Also

Related Examples

- "Window Layouts" on page 2-6
- "Camera Control in RoadRunner" on page 1-44
- "Create Roads Around Imported GIS Assets" on page 1-57
- "Create Traffic Signals at Junctions" on page 1-64
- "Choose a RoadRunner Tool" on page 2-35
- "Keyboard Shortcuts and Mouse Actions for RoadRunner" on page 2-31

Camera Control in RoadRunner

RoadRunner enables you to edit large-scale and small-scale details of a 3D environment that can span many kilometers or miles. The interactive camera controls enable you to navigate this large 3D space quickly and effectively. This example shows you the fundamentals of camera controls in the RoadRunner scene editing environment.

Open Scene

Open a basic scene to move the camera around in. From the menu bar, select **File**, then **Open Scene**. Then, open `FourWaySignal.rrscene`, which is one of the default scenes that is included in the `Scenes` folder of RoadRunner projects. The scene opens top-down in the center of the screen, inclined at an angle of 45 degrees.
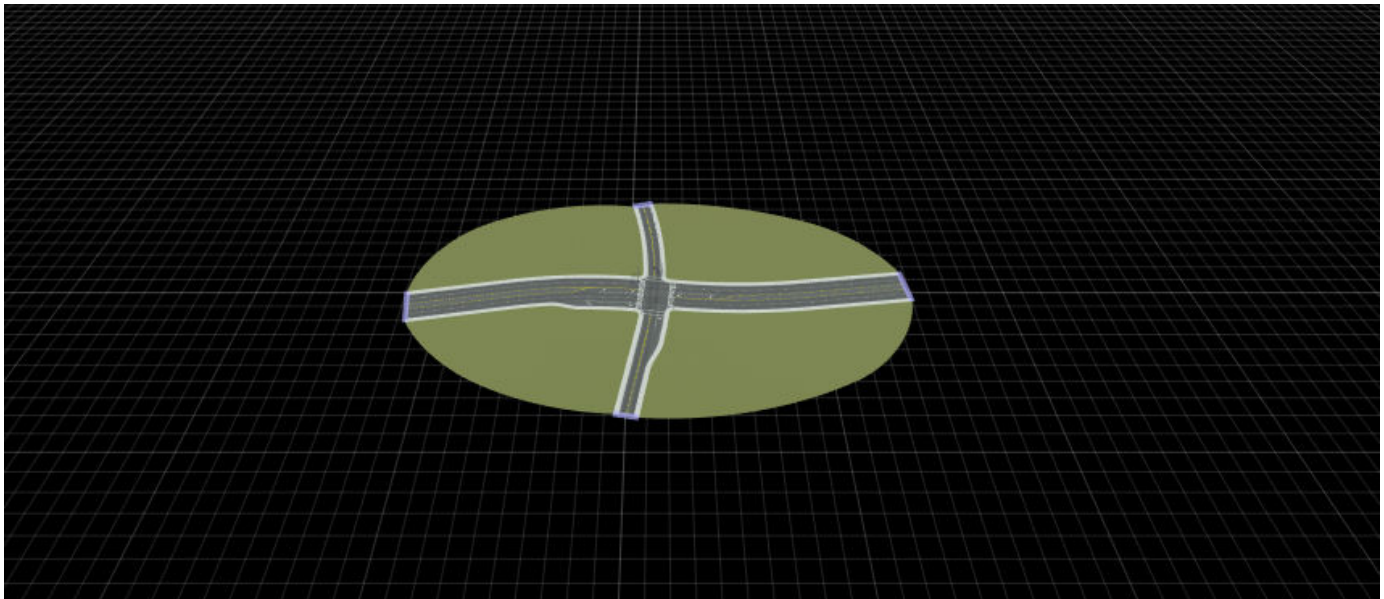
Rotate Camera

Camera control in RoadRunner is based on a polar viewing model, where the camera orbits around a point of interest at a fixed distance. By default, when you open a new scene, the point of interest is 1.5 meters above the origin, to approximate the position of the head of a person standing at the center of your scene. The point of interest for this scene is at the center of the intersection.

You can rotate the camera around the point of interest at any time and from within any tool by pressing and holding either the **Alt** key or the **Windows** key and moving the pointer.

Note In Linux, Ubuntu 16.04, pressing the **Alt** key moves the current window and pressing the **Windows** key shows certain help overlays. To use the **Windows** key instead of the **Alt** key for moving windows, update Ubuntu 16.04 according to the instructions in "Update Linux Ubuntu Key Mapping" on page 2-34.

Hold the **Alt** key, click anywhere in the scene, and drag the pointer. Observe the change in camera rotation with respect to the point of interest.
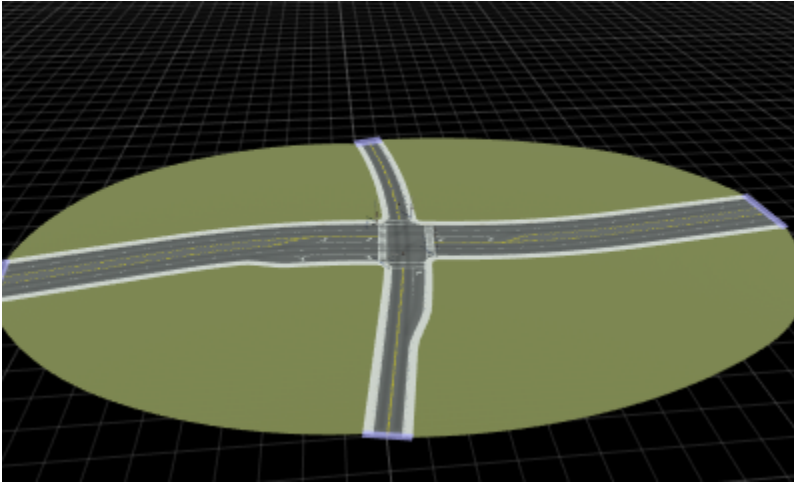
Zoom Camera In and Out

To zoom the camera in, hold **Alt** and the right-click button, and then drag the pointer up or right. Conversely, to zoom the camera out, hold **Alt** and the right-click button, and then drag the pointer down or left. Alternatively, you can use the mouse scroll wheel to zoom in or out.

Hold **Alt** and the right-click button, and then drag the pointer down to zoom out. Observe the change in the fixed distance at which the camera orbits the scene.
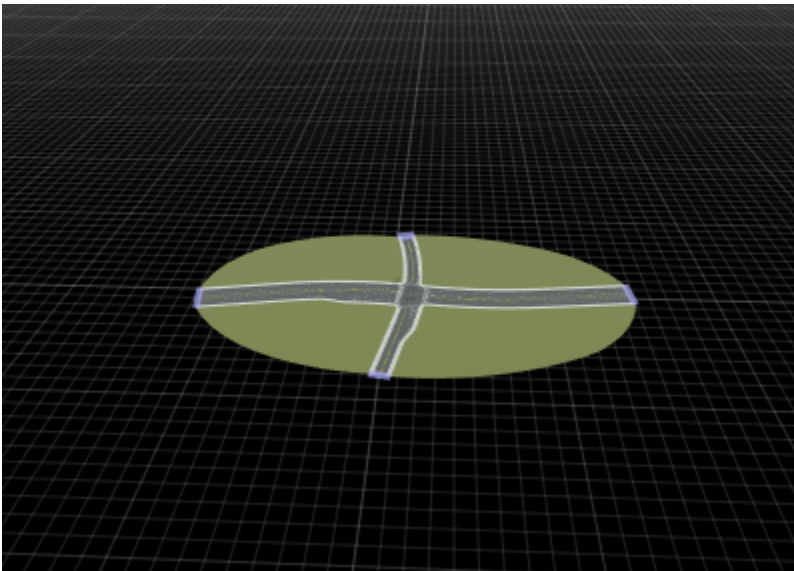
While still holding **Alt** and the right-click button, drag the pointer up and zoom all the way in to the point of interest until the camera stops moving. Zooming in this far focuses the camera at the pavement.

Hold **Alt** and the left-click button to rotate the camera. The camera movement at this distance is similar to standing at a fixed location and looking around as the camera orbits the scene.
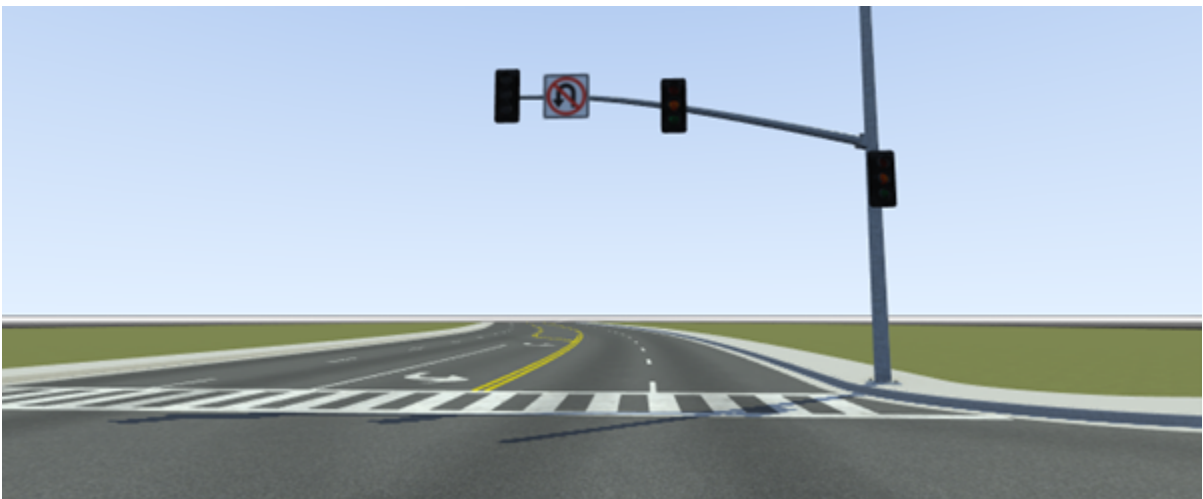
Push Past Behavior

Push past behaviour enables you to scroll past the pivot point, when the pivot point is right in front of the camera. Scrolling past the pivot point allows you to reach the desired location in the scene without being blocked. Once you push past the pivot point, the pivot point is maintained in the new position in the scene.

Move Camera Horizontally

To move the camera horizontally along the ground (xy) plane, first hold the **Alt** key and the left-click and right-click buttons. Then, drag the pointer. Alternatively, you can move the camera by holding the middle-click button and dragging the pointer.

Hold **Alt**, the left-click button, and the right-click button, and then drag the pointer to move the camera to the left and right. The point of interest shifts to the new location.

Move Camera Left (Drag Left)	Original Location	Move Camera Right (Drag Right)
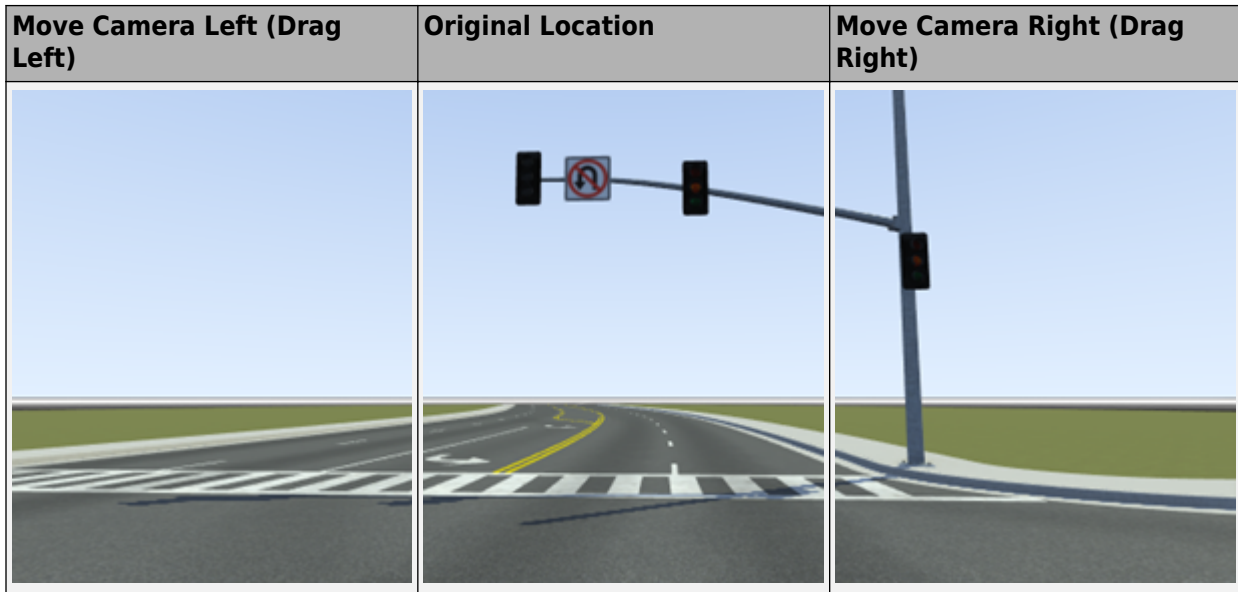
Move Camera Vertically

For simple environments, you can keep the height of the camera point of interest set to the default. However, for more complex environments, you might need to move the point of interest up or down. For example, if you are designing a scene with bridges, you might need to move the point of interest down so that you can maneuver the camera under a bridge.

To move the camera up, hold **Alt**, **Shift**, left-click, and right-click, and then drag the pointer down. Conversely, to move the camera down, hold these same keys and buttons, and then drag the pointer up. Alternatively, you can hold **Alt**, **Shift**, and the middle-click button and then drag the pointer up or down for the same affect.

Hold **Alt**, **Shift**, left-click and right-click, and then drag the pointer up and down. Observe the change in the point of interest as the camera moves up and down.

Move Camera Up (Drag Down)

Original Location

Move Camera Down (Drag Up)
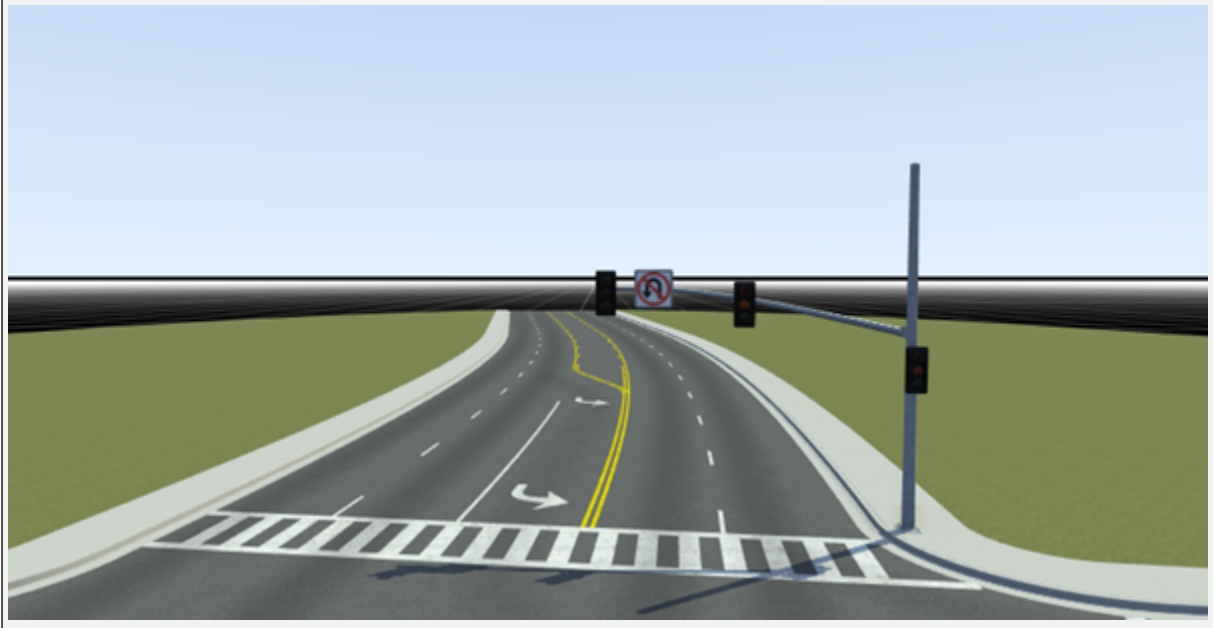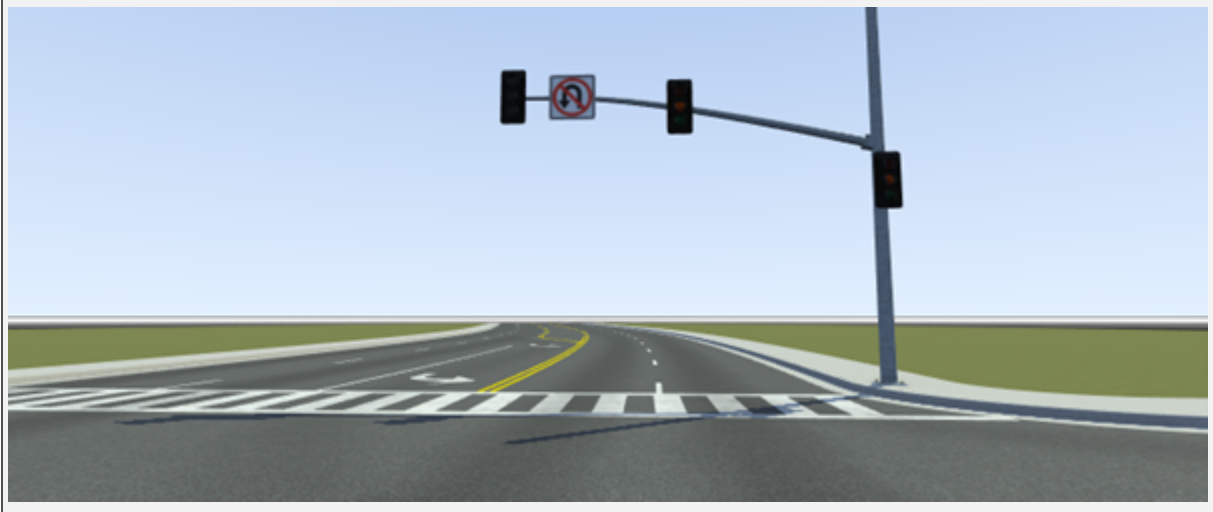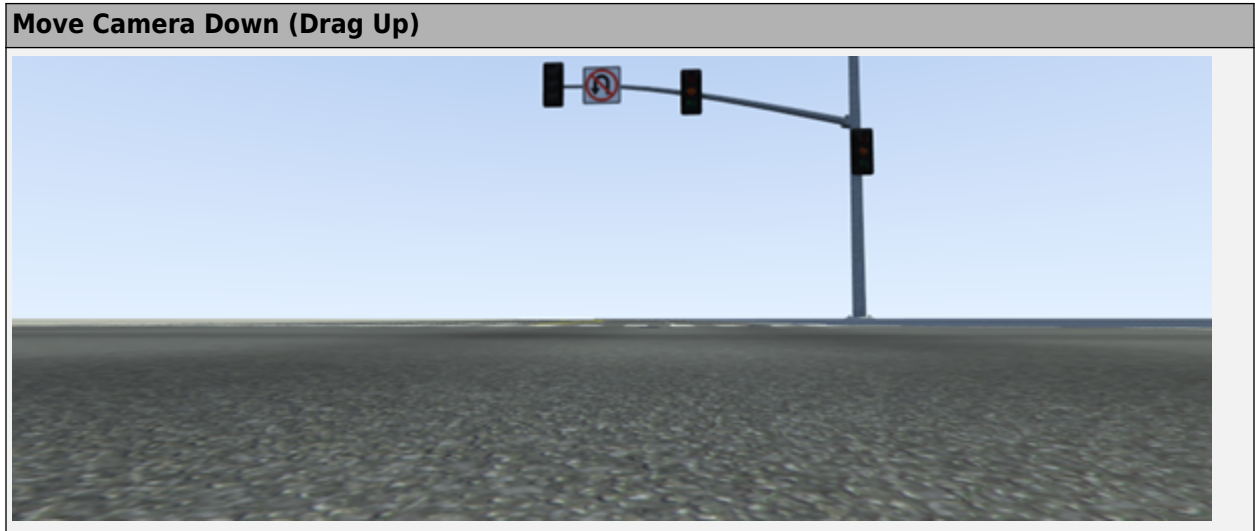
Frame Camera on Selected Object

From within any tool, you can center, or frame, the camera on the currently selected objects. To center the camera on a selected object, press the **F** key. Alternatively, from the **View** menu, select **Frame Selected**.

1   Zoom out of the scene. Hold **Alt** and the right-click button, and then drag the pointer down.

2
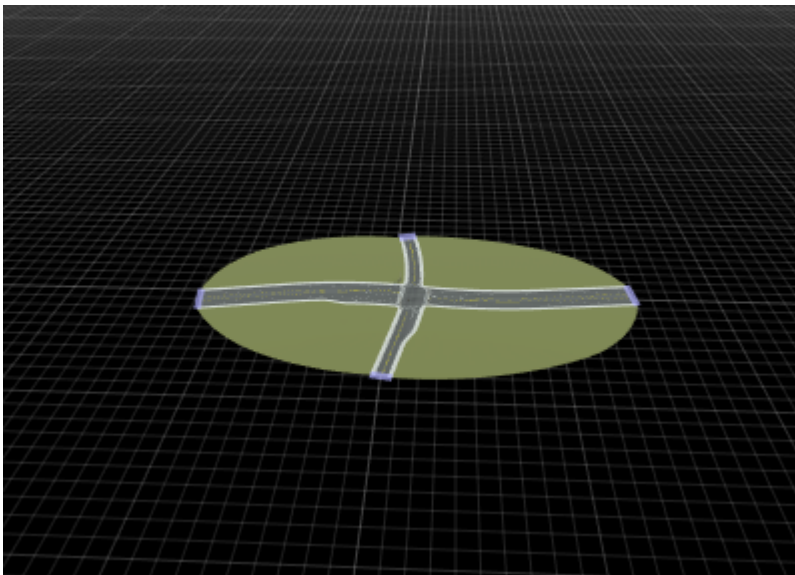
Click the **Road Plan Tool** button to make the roads selectable.

3   Select the longer road and press the **F** key. The camera centers on the longer road.
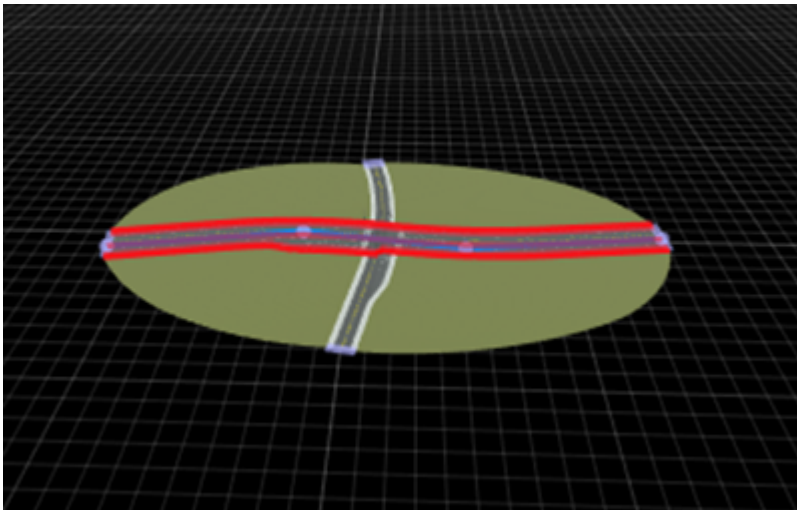
4   Select the other road and press **F**. The camera zooms in to center on the shorter road.
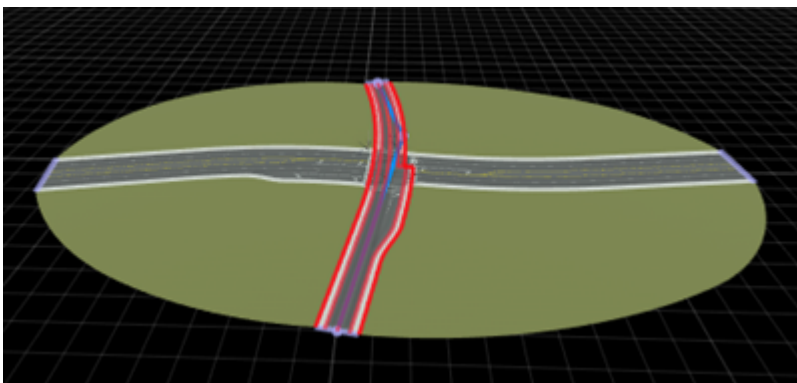
5

   Click the **Prop Point Tool** button to make the traffic light props selectable.

6   Select one of the prop points and press **F**. The camera zooms in on the selected prop.
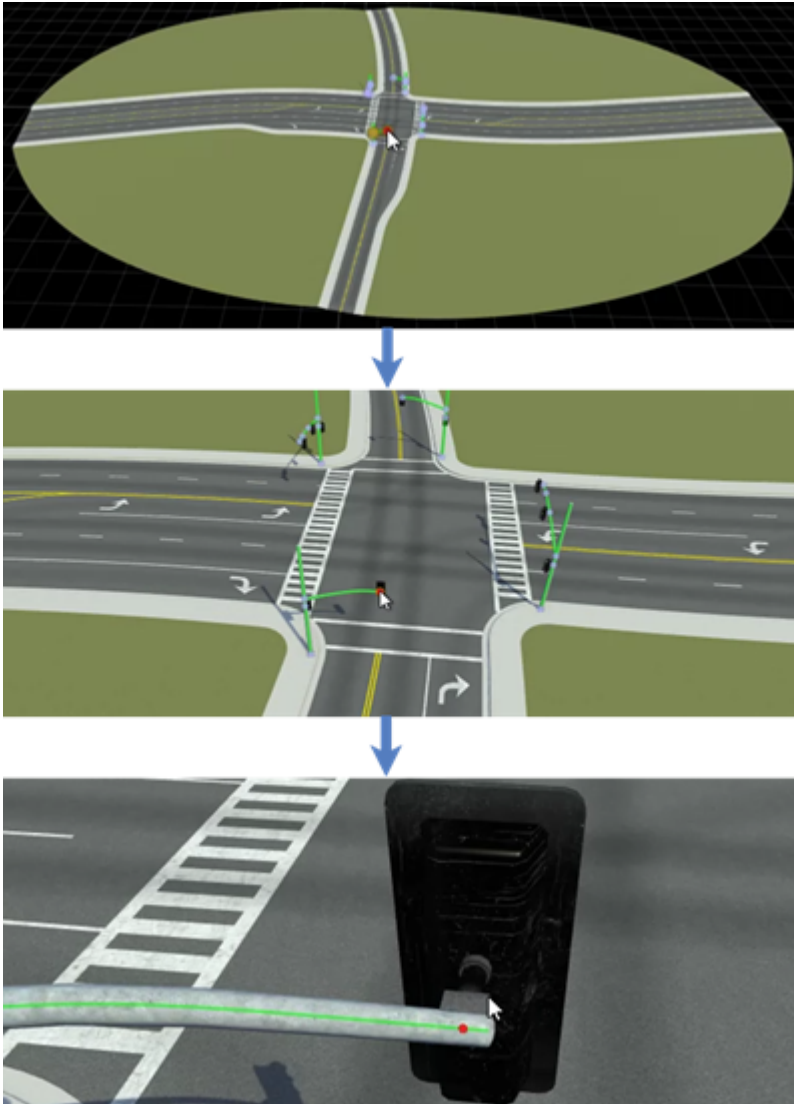
When you rotate the camera by holding **Alt** and then clicking and dragging, the camera rotates around the prop.
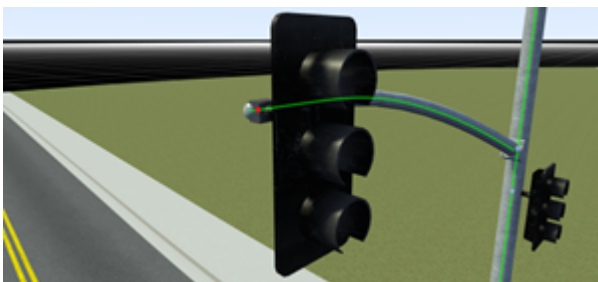
If no view or object is selected, pressing the **F** key or selecting **Frame Selected** from the **View** menu, frames all of the data in the scene, preserving the camera angle position. For example, in this image, the scene is located at one end of the editing canvas. Clicking **Frame Selected** without selecting any view or object, brings the scene data back into focus.
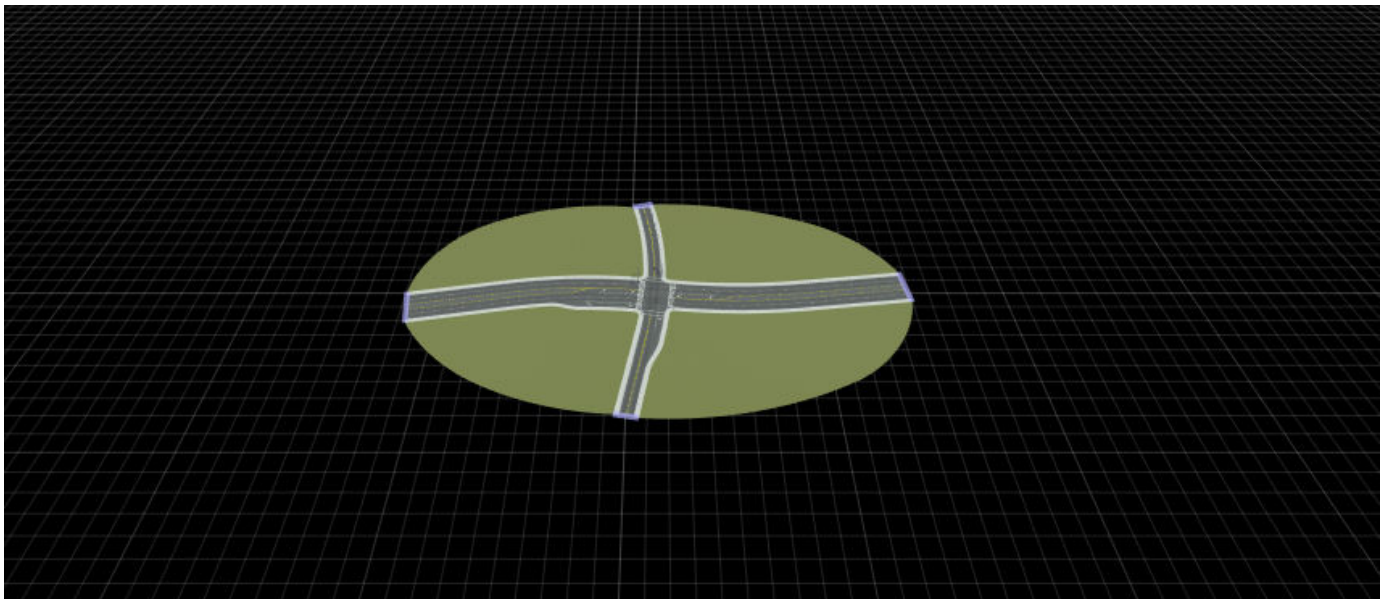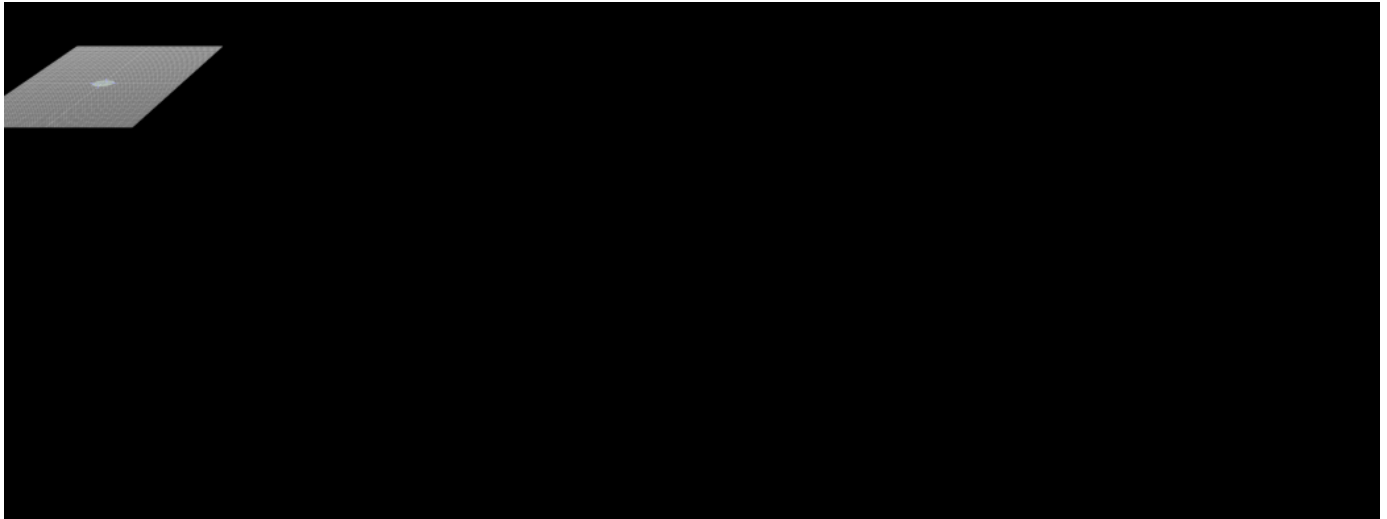
If the scene is empty, pressing the **F** key or selecting **Frame Selected**, takes you back to the origin point in the scene editing canvas.
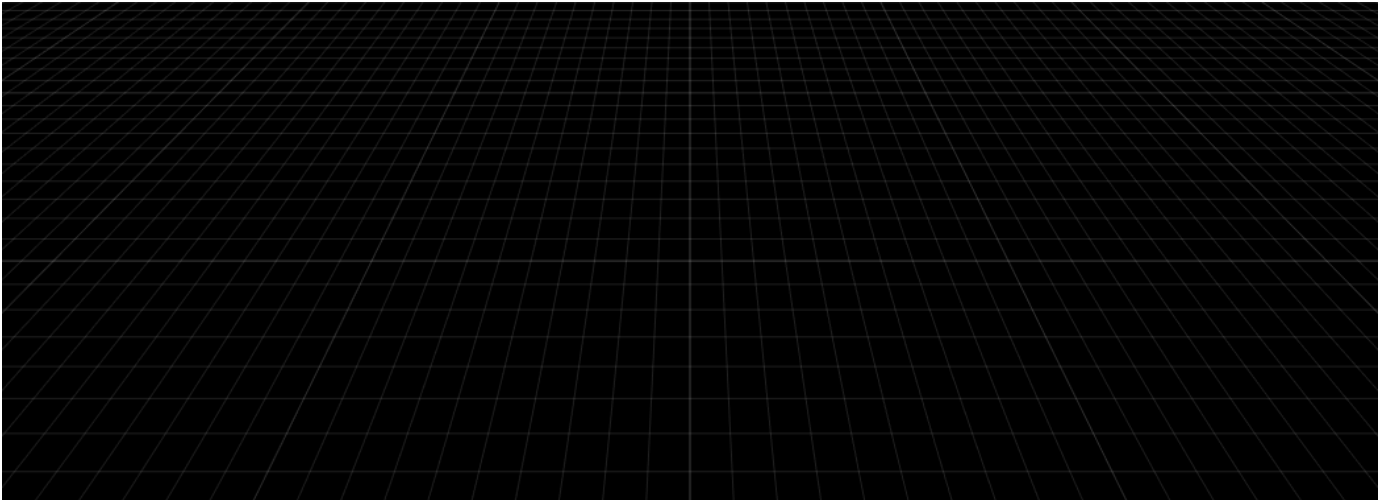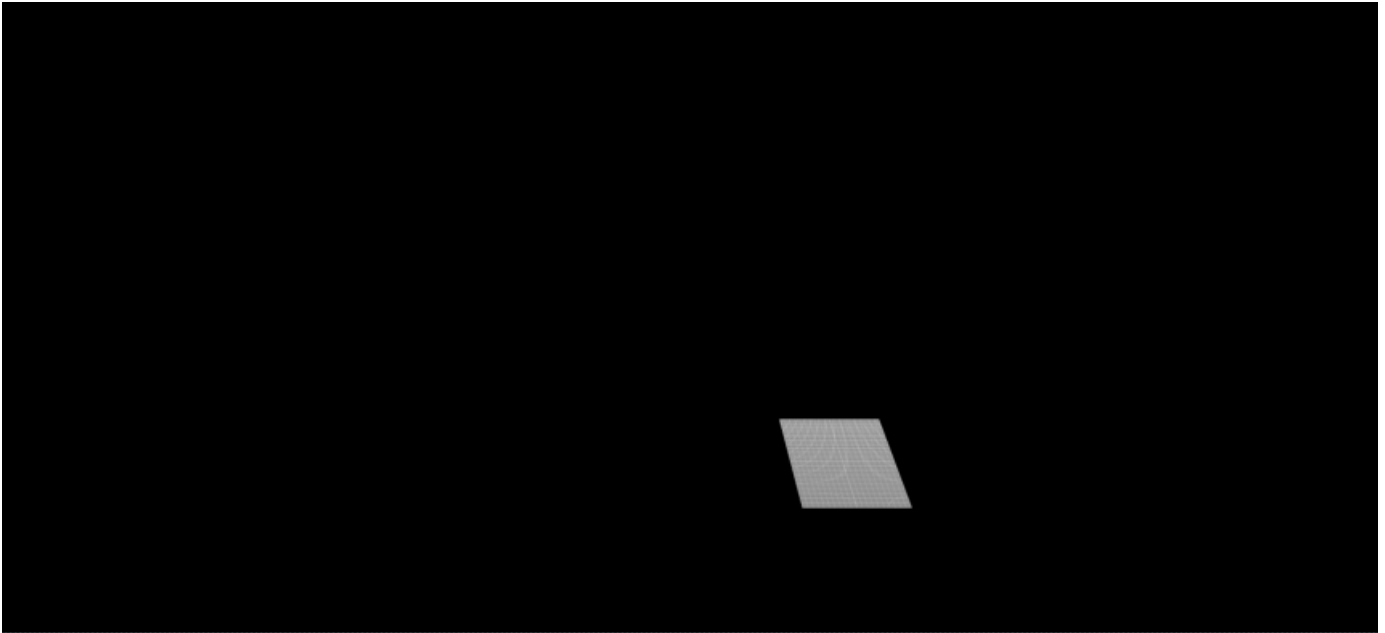
Frame Camera on Cursor

You can center, or frame, the camera to the point where your cursor is currently located. To center the camera on a cursor, follow these steps:

1   Zoom out of the scene. Hover your cursor at one end of the road.
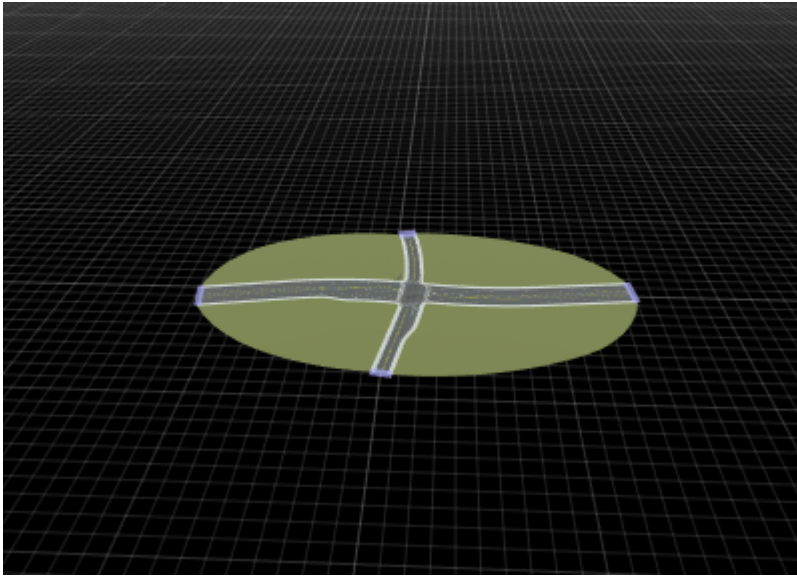
2   Press **V**. The camera zooms in to end of the road.
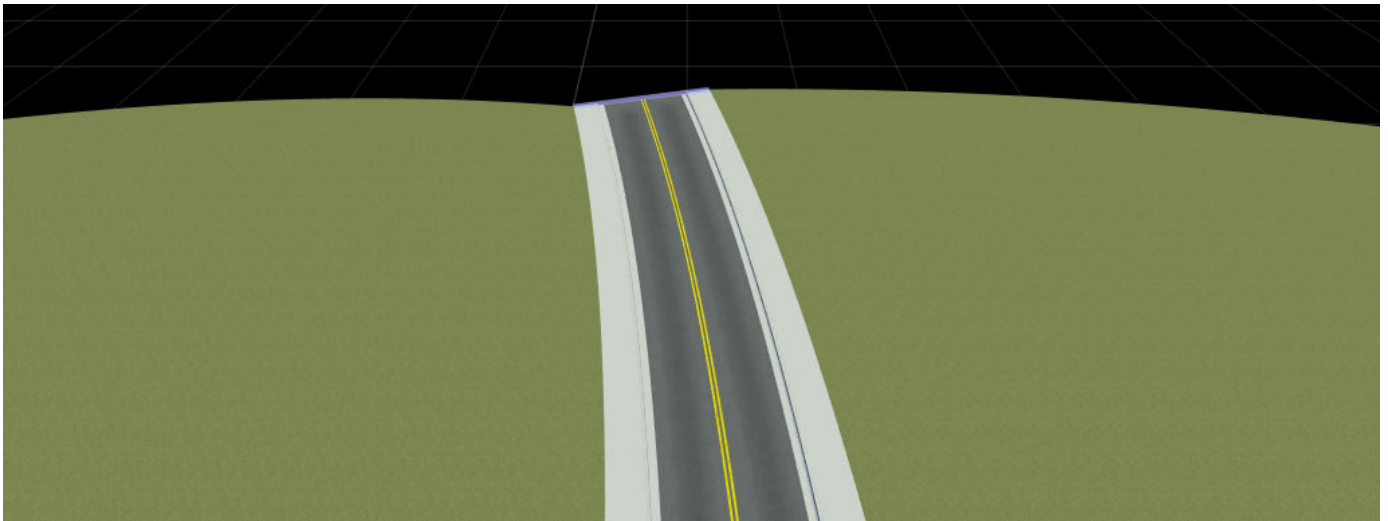
Change View Projections

The RoadRunner camera can use either a perspective or orthographic viewing projection.

- The perspective projection is the default viewing projection, which causes distant objects to appear smaller than close objects.
- The orthographic projection is similar to what you might find in a CAD tool. It is useful for precise positioning, usually from a top-down point of view. In orthographic mode, objects do not change apparent size as they get closer or further away.

The camera controls work the same in both projection modes.

Move the camera so that the entire intersection is in view. Then, press **O** to switch to orthographic mode. In this mode, the traffic lights are all the same size. To zoom in to a specific location in the

orthographic mode, hover your cursor over the location and scroll in. This takes you to the desired location.

Press **P** to switch back to perspective mode, where the traffic lights in the distance appear smaller.

Orthographic Mode (Press O)	Perspective Mode (Press P)
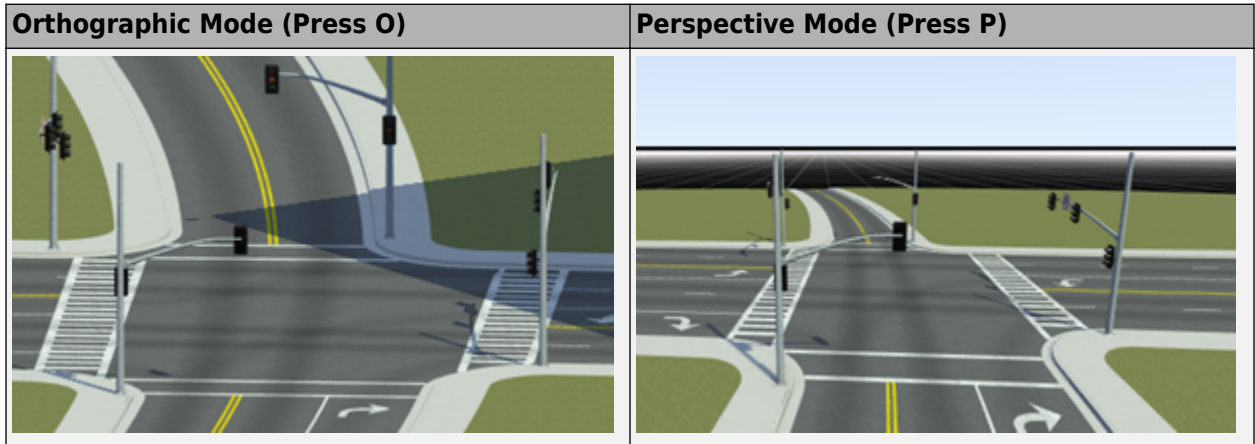
Set View Direction of Camera

You can set the view direction of the camera to due north, south, east, west, or top-down. To change the view direction, on the **View** menu, select **Direction**, and then the select the view direction you want. Alternatively, you can use number pad shortcut keys.

Change the view direction of the scene by using these keys on the number pad. This table shows sample view directions when the camera is at the intersection of the scene and their corresponding number pad shortcut keys. In the top-down view, the camera also rotates to point north.

	North (**8** key)	

West (**4** key)	Top-Down (**5** key)	East (**6** key)
	South (**2** key)	
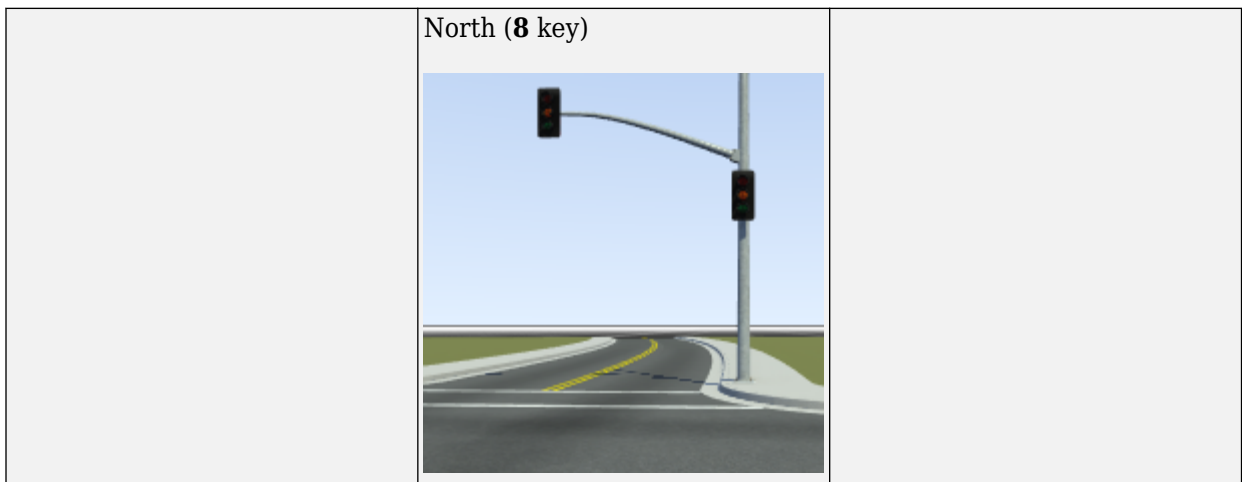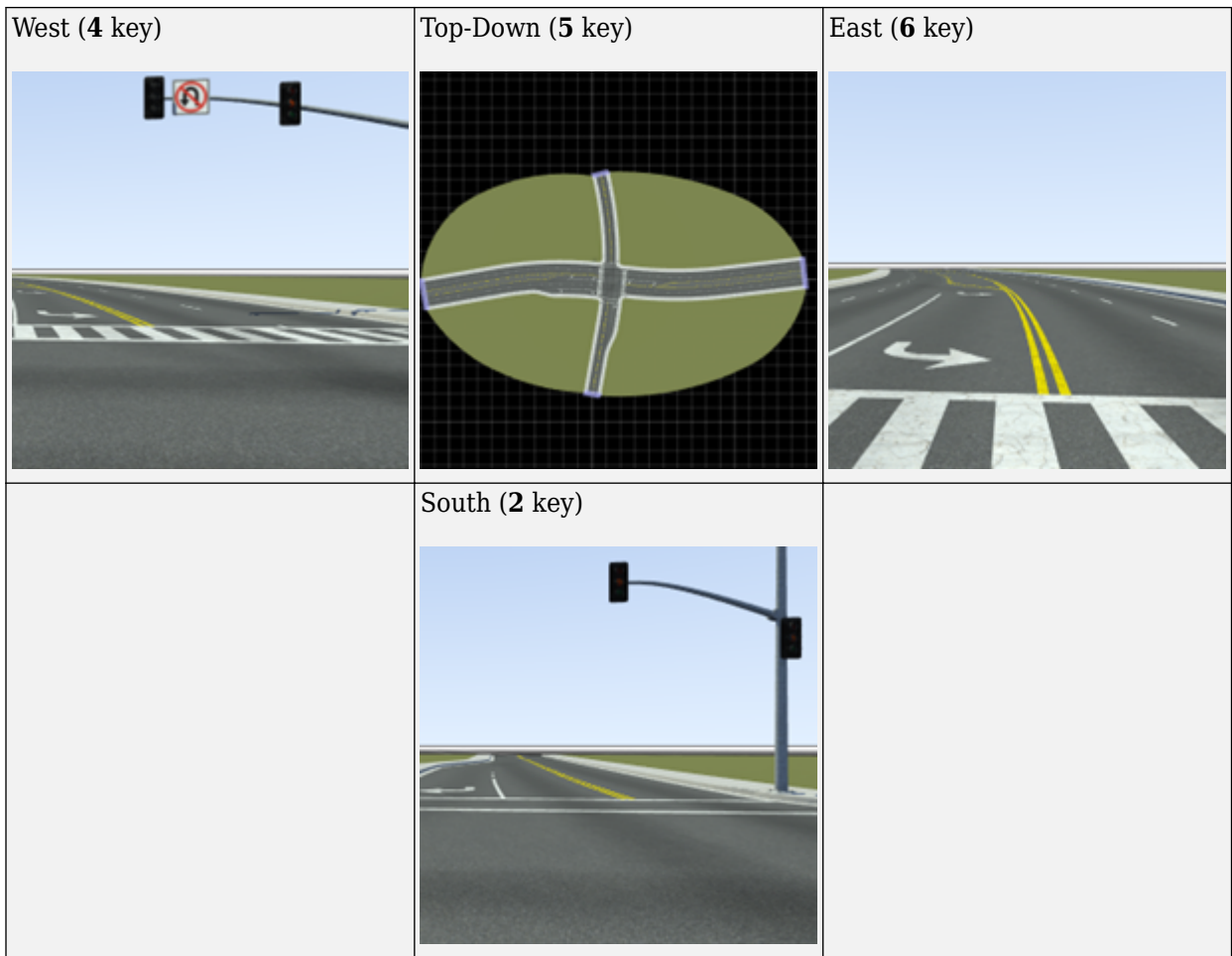
See Also

Related Examples

- "Create Simple RoadRunner Scene" on page 1-19
- "Keyboard Shortcuts and Mouse Actions for RoadRunner" on page 2-31

Create Roads Around Imported GIS Assets

RoadRunner supports a variety of geographic information system (GIS) formats. You can import GIS assets into RoadRunner and use them as a reference to construct your road network.

In this example, you create roads around imported GIS assets that you download from the U.S. Geological Survey (USGS). The example uses these GIS assets:

GIS Asset	Description
Aerial Imagery	Visual reference for roads and surface texture mapping
Point Cloud	Visual reference and object placement information (trees, buildings, markings, and so on)
Elevation	Height information about the terrain

Download and Import GIS Assets into RoadRunner

1   First you need to download and setup the GIS assets that you want to reference.

2   Download aerial imagery, elevation, and point cloud data of a specific location from the Basic National Map Explorer Interface at the USGS. For this example, search for Downey, California, a city in the county of Los Angeles. To learn how to download GIS data supported by RoadRunner, see "Download GIS Data for Use in RoadRunner" on page 3-8.

3   Open a RoadRunner project and create a new scene by selecting **File**, then **New Scene** from the main menu.

4   To store the imported assets , right-click in the **Library Browser** and select **New**, then **Folder**, and name the folder. For this example, name the folder GIS.

5   To organize the assets in the GIS folder, create subfolders for each type of asset: Aerial Imagery, Elevation, and Lidar folders. Move the downloaded assets to the respective folders by dragging the aerial imagery, elevation, and point cloud files into the **Library Browser**.
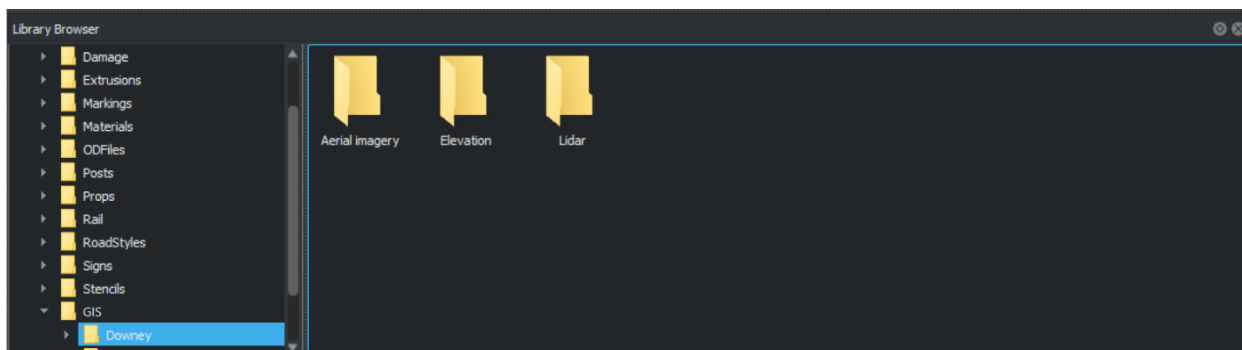
For more details on importing assets, see "Create, Import, and Modify Assets" on page 2-50.

Set World Origin

Now that you have downloaded and stored the GIS assets, set a point of origin around which to apply them in the scene.

1 To project the GIS assets around a specific geographic position, you use the **World Settings Tool** [image]. For this example, set the world origin latitude to `33.9383` degrees and the world origin longitude to `-118.1296` degrees. Click **Apply World Changes**. The region displayed on the scene canvas defines the area where you load the GIS assets.

2 You can adjust the region size in which you want to load the assets. With the **World Settings Tool** selected, use the **Extents** attribute to change the size of the region.

Add GIS Assets

After you set the origin, load the aerial imagery, elevation, and point cloud data into the scene.

1 In the **Library Browser**, navigate to and select the aerial imagery file, making sure that it has a projection. To check the projection of an asset, click the file and in the **Attributes** pane, check the **File Projection (WKT)** section.

2 Drag the aerial imagery file into the scene. A rotating orange wheel on the top right of the canvas appears, indicating the loading progress of the file. RoadRunner places the file in the correct world and scene position with respect to the specified World Origin. For example, in this image, the aerial imagery data covers the top portion of the scene.
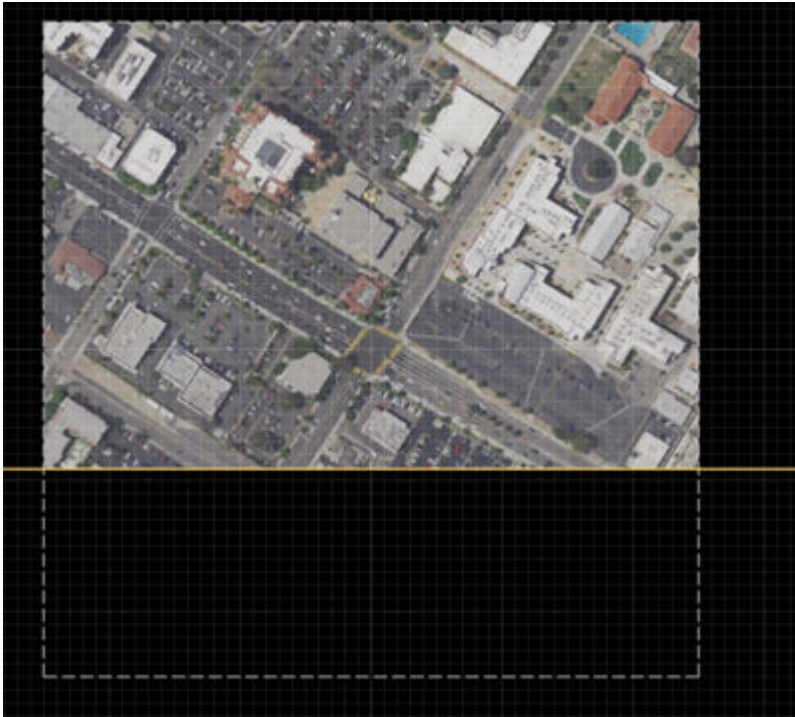
3 Similarly, to add the elevation files, drag the files into the scene. Make sure that it has a projection. You can drag multiple files of the same type into the scene editing canvas.

Dragging elevation files into a scene automatically switches RoadRunner to the **Elevation Map Tool** and adds the appropriate elevation to the scene, projected with respect to the world origin.

To toggle the visibility of the elevation data, from the **View** menu, select **Elevation Map** . RoadRunner provides additional options for toggling the visibility of other GIS asset types.

Elevation Disabled	Elevation Enabled
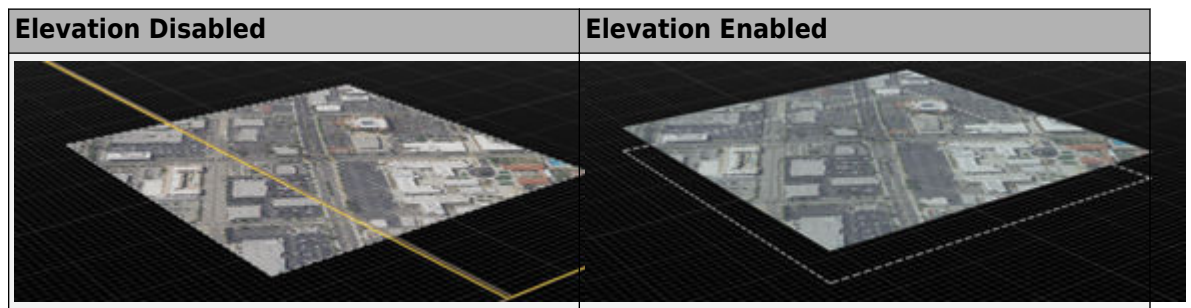
4 Finally add point cloud data. Check for projection in the **File Projection (WKT)** attribute. Select multiple files and drag them into the scene. To toggle the visibility of the point cloud data, from the **View** menu, select **Point Cloud** .

The point cloud data, as shown by the grey dots, are automatically placed in the correct regions with respect to the world origin.
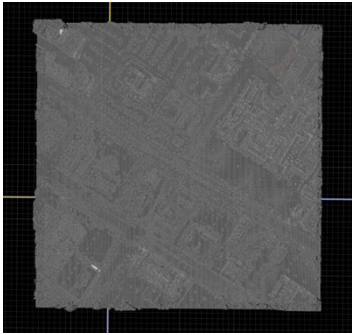
5 Customize the point cloud appearance and feature visualization. From the **Point Cloud Tool** in the **Attributes** pane, set **Color By** to **Intensity**. Then, select **Use Custom Intensity** and adjust the **Intensity Min** and **Intensity Max** values to make the scene more visible. This example shows the difference in intensity when the maximum intensity is reduced from `40000` to `8000`.
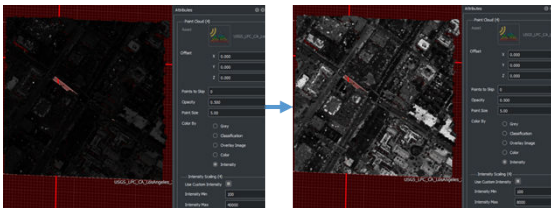
Create Roads Around GIS Assets

After importing the GIS assets, use the tools in RoadRunner to customize the scene.

1 To build roads around the imported assets, toggle the display of aerial imagery assets. On the **View** menu, select **Aerial Imagery**. Then, select the **Road Plan Tool** and draw a road over the existing road imagery.
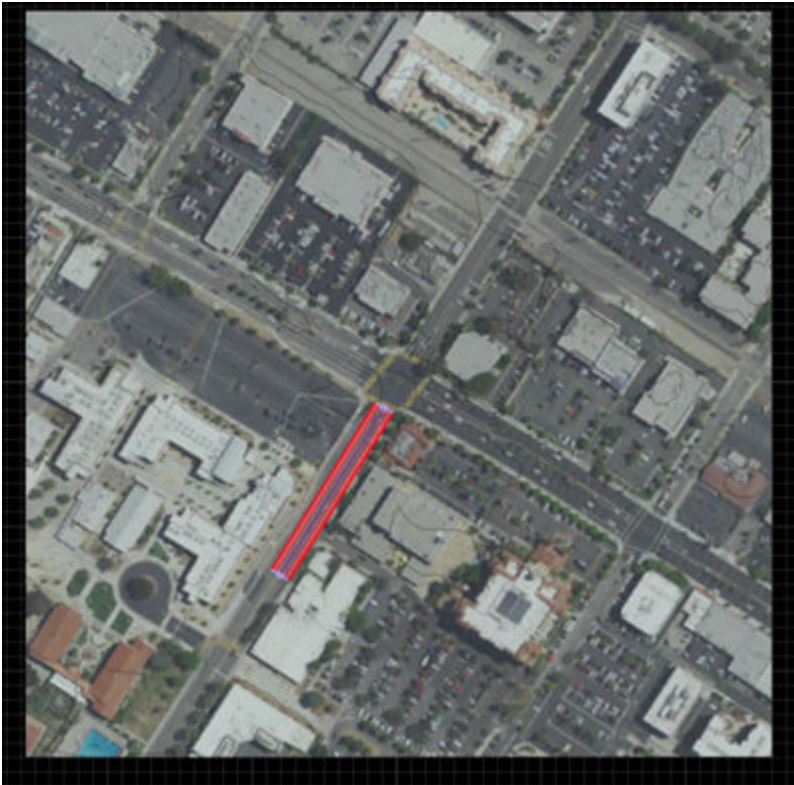
2 To match the elevation of the road to the imported elevation asset, in the toolbar on the left side

of the canvas, click the **Project Roads** button .

3 Use the **2D Editor** to view and adjust the elevation profile of the road.

For example, these views show the road elevation (red) being adjusted to match the imported elevation data (blue). You can also adjust the point cloud data (green) to match the elevation data.
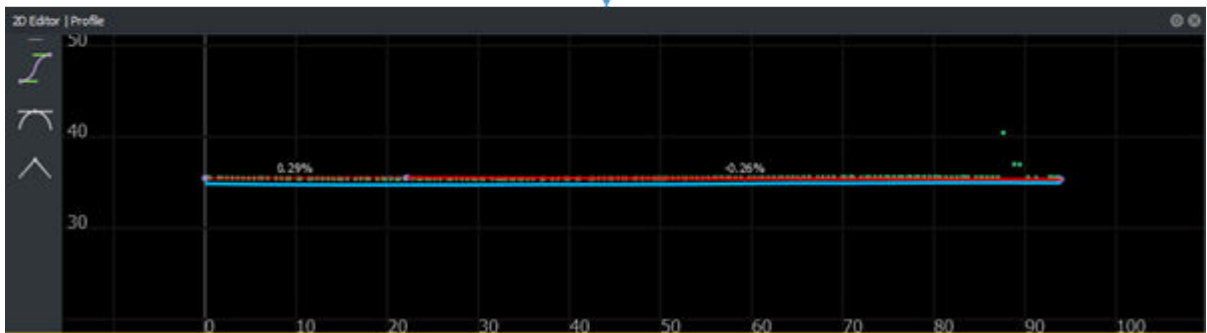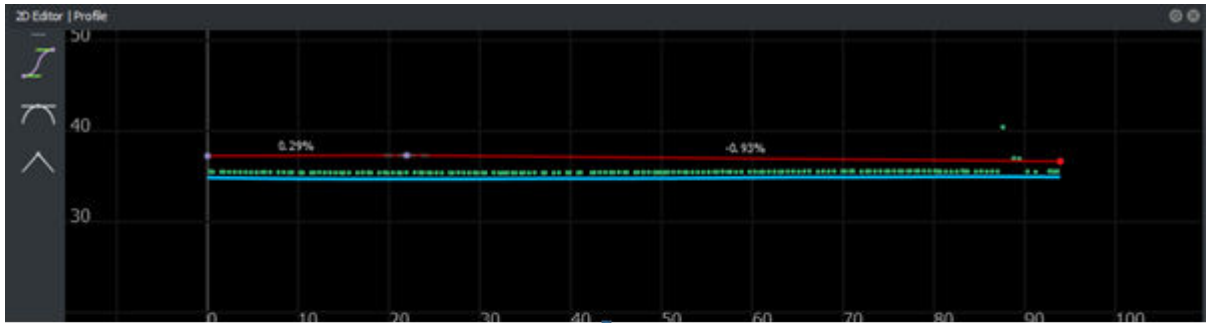
To control the position or height of the road, in the **2D Editor**, select the road centers (purple dots) and adjust them to the desired height. To adjust multiple road centers to the same height or position, press **Ctrl+A** and select the points to maneuver them to the appropriate height or position.

Compare Roads Against Imported GIS Assets

To check the accuracy of road mapping and synchronization, compare the road network that you created with the imported GIS assets data. From the **View** menu, toggle each GIS asset view on and off by selecting and deselecting **Aerial Imagery (F4)**, **Elevation (F5)**, and **Point Cloud (F6)**.
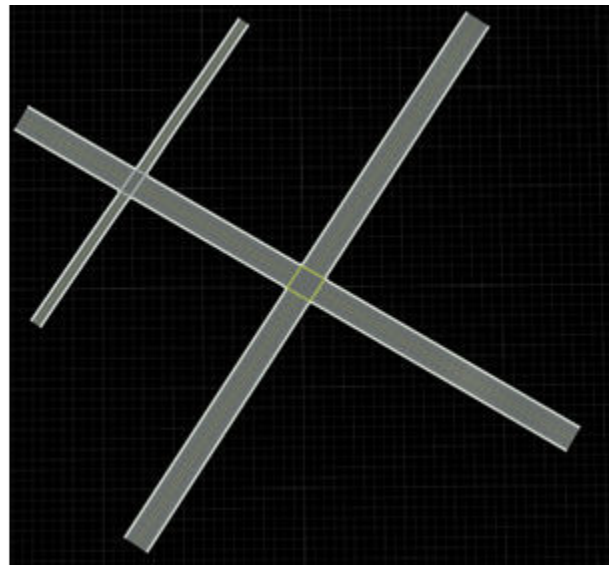
To further customize the scene, you can use different assets to add 3D models, textures, road signs, stencils, and other data shared by multiple RoadRunner scenes. For more information, see "RoadRunner Asset Types" on page 2-45.

For example, the left scene shows an **Aerial Imagery** view for a different data set, and the right scene shows the corresponding customized scene. This scene was designed using different tools within RoadRunner to create buildings, roads, parking lots, signboards, trees, and other scene objects.
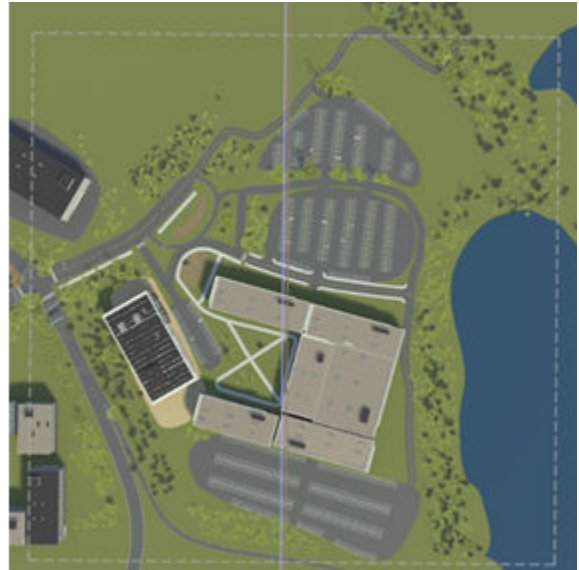
Create Roads Automatically from HERE HD Live Map Road Data

To automatically generate 3D road models, you can use the add-on product, RoadRunner Scene Builder, which imports and automatically synthesizes 3D road models from HERE HD Live Map.

See Also

Related Examples
- "Import Scene Data"
- "Download GIS Data for Use in RoadRunner" on page 3-8

External Websites
- Here Technologies

Create Traffic Signals at Junctions

This example shows how to create working traffic signals at a junction using a four-way protected left traffic pattern. To add traffic signals in RoadRunner, you use the **Signal Tool**, which lets you configure junction signalization and control traffic signal phases. This example shows the entire workflow for creating a junction, adding dynamic signalization to the junction, adding props or assemblies to the signal junction, and modifying signal phases and maneuvers.
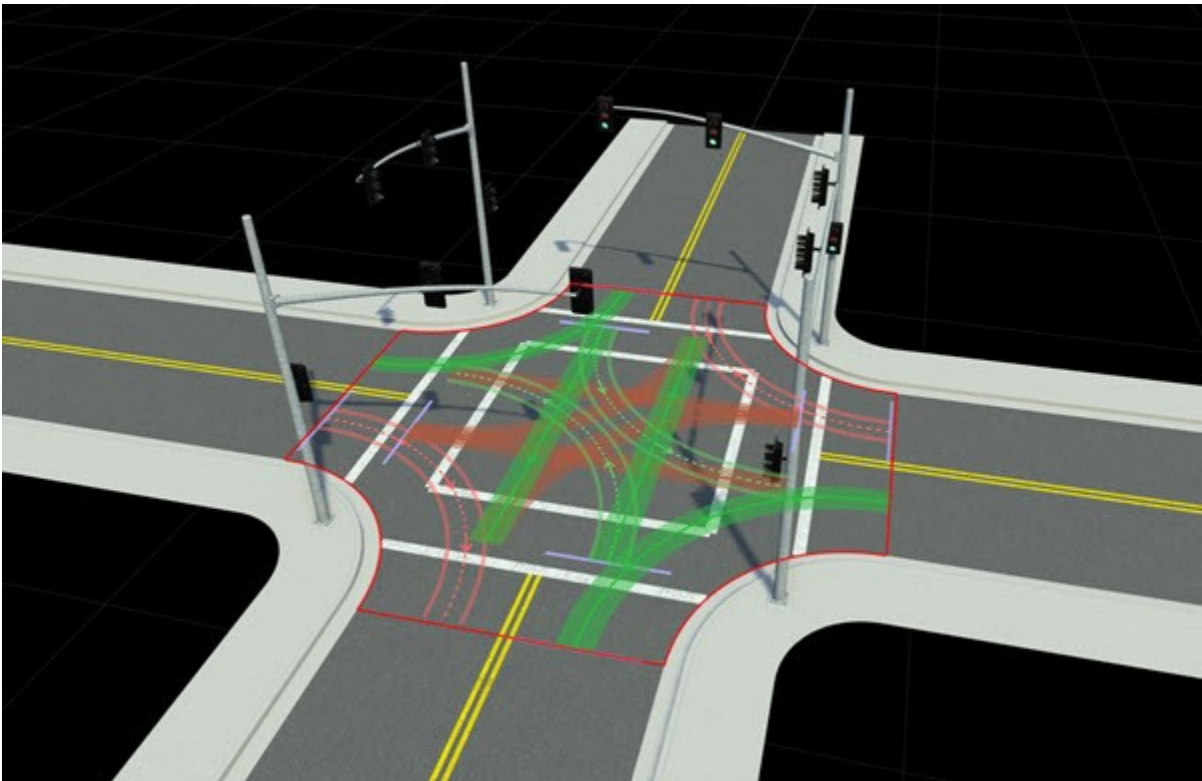
Create New Scene

Create a new scene within a project.

1 Open RoadRunner, and from the start page, click **New Scene**.

2 On the **Select a Project** window, select the project that you want to work in.

The RoadRunner canvas opens, where you can start building your scene.

Create Junctions

To place traffic signals in your scene, first create a junction. You can use the **Road Plan Tool** to create automatic junctions at road intersections.
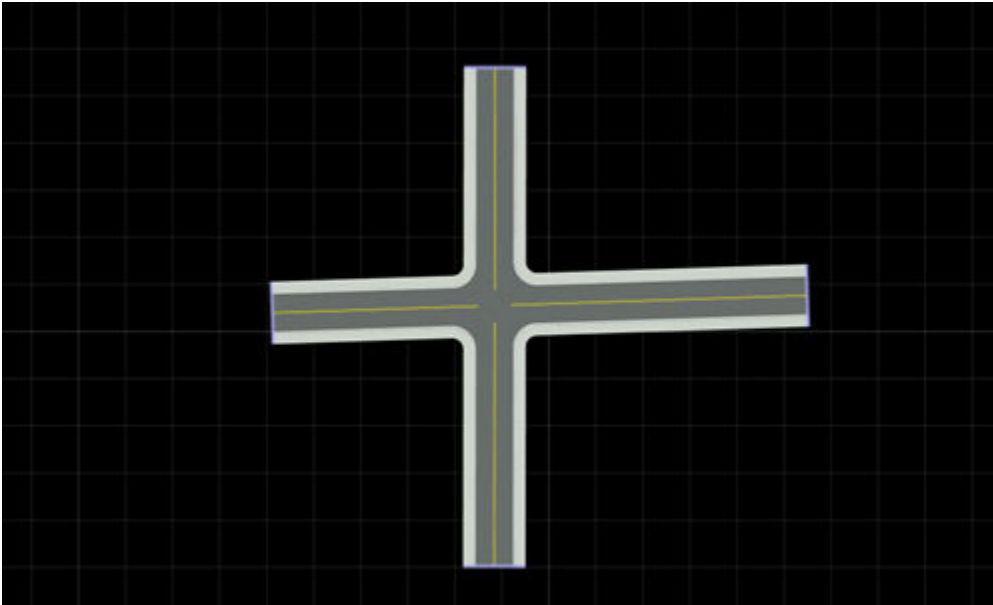
To create a four-way intersection, create two roads that fully overlap:

1 Click the **Road Plan Tool** .

2 Right-click at a start location and an end location to create a road segment.

3 Create another road segment that overlaps the previously created road. This action creates a four-way junction.

To create custom junctions manually, you can use the **Custom Junction Tool**.

Add Signals to Junctions

To configure junction signalization and signal traffic phases, use the **Signal Tool** .

The Signal Tool provides several autosignalization operations for automatically applying predefined signalization templates to a junction. The junction signalization can be static (not changing, for example, controlled by stop signs) or, as in the case of this example, dynamic (controlled by traffic signals).

The autosignalization operations can also automatically place **Prop Assembly Assets** and **Signal Assets** which will automatically link to the corresponding signals. You can also use predefined prop assembly to the link the props to the corresponding signals.

1 To add a predefined prop assembly asset to the scene, select the prop in the **Library Browser**.

Navigate to the `Assets` folder, and select `Assemblies`. For this example, select the **ProtectedLeft1** prop.
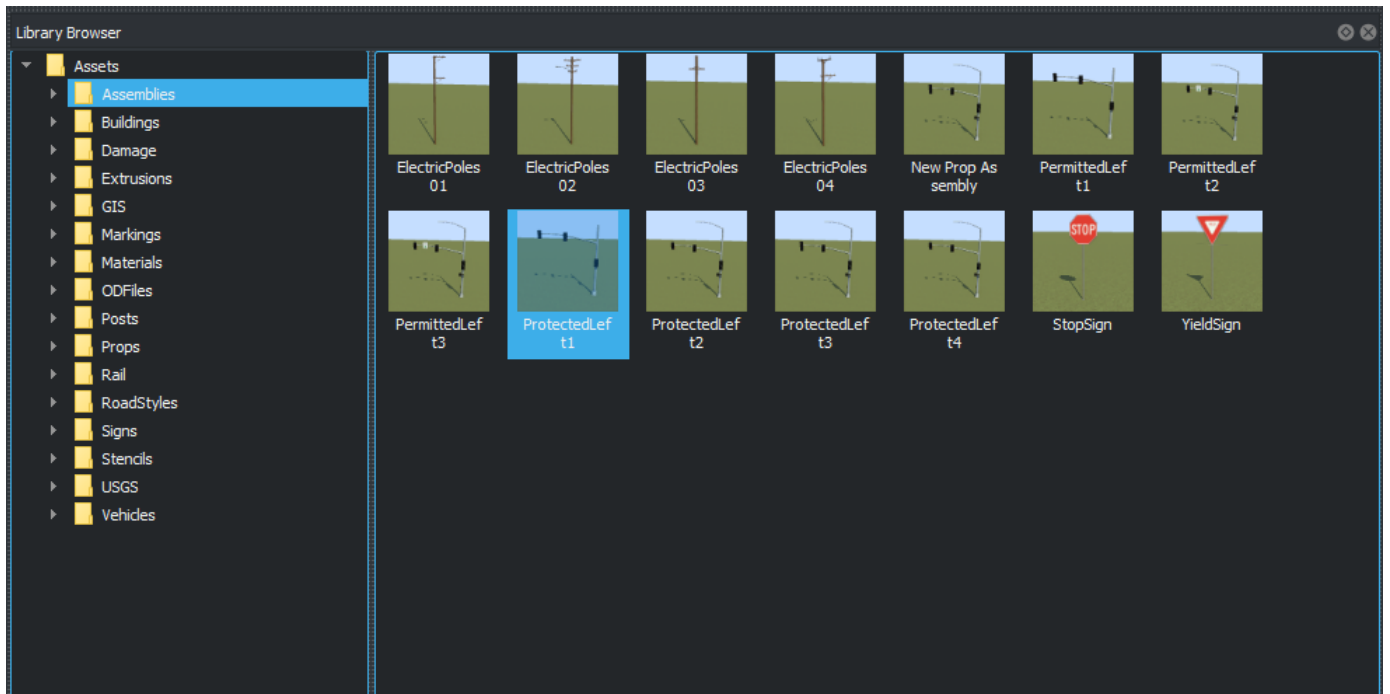
2   To add signalization, click **Signal Tool** and select the junction.
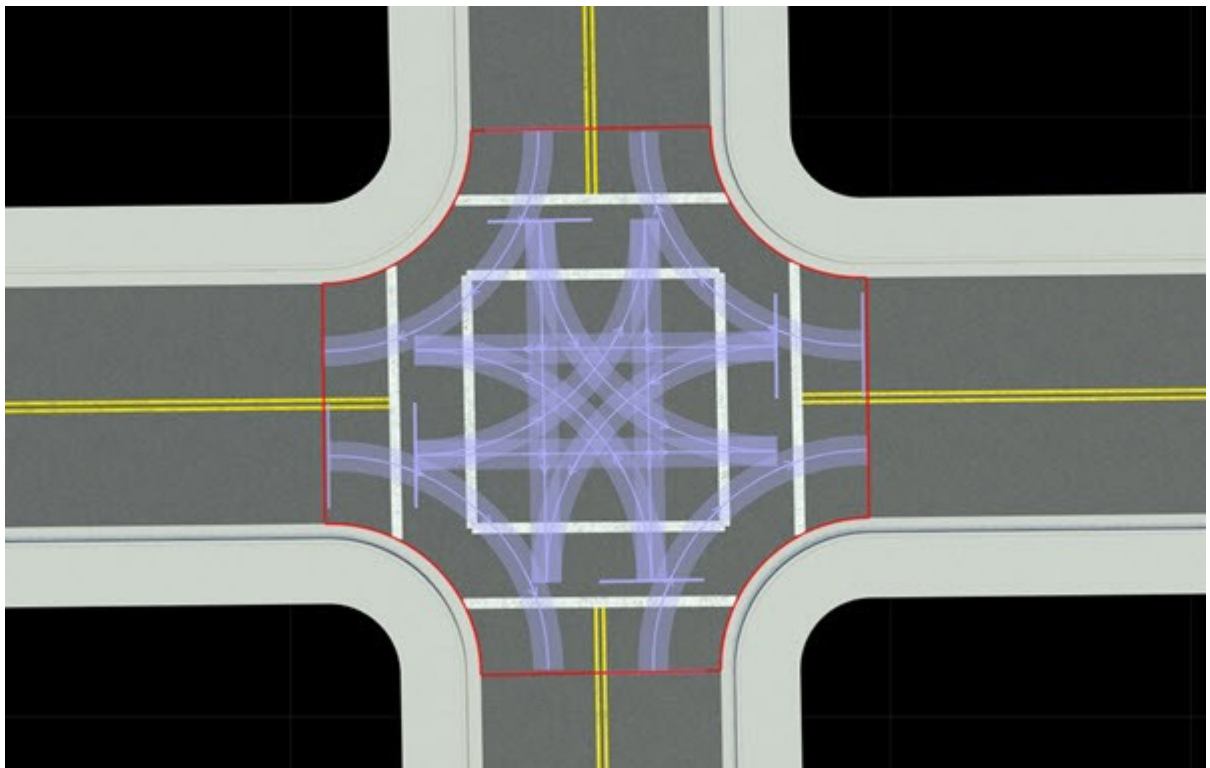
3   To signalize the junction, click **Auto Signalize** in the **Attributes** pane.

4 From the Auto Signalize Junction dialog box, select a signalization template and click **Signalize**. For this example, select the **four-way Protected Left** template. Select **Automatically Place Selected Prop** to add the props and link it to the signal.

The scene shows the autosignalized junction with a four-way protected left traffic pattern. When you add prop or signal assemblies to the scene, they are automatically linked to the junction.
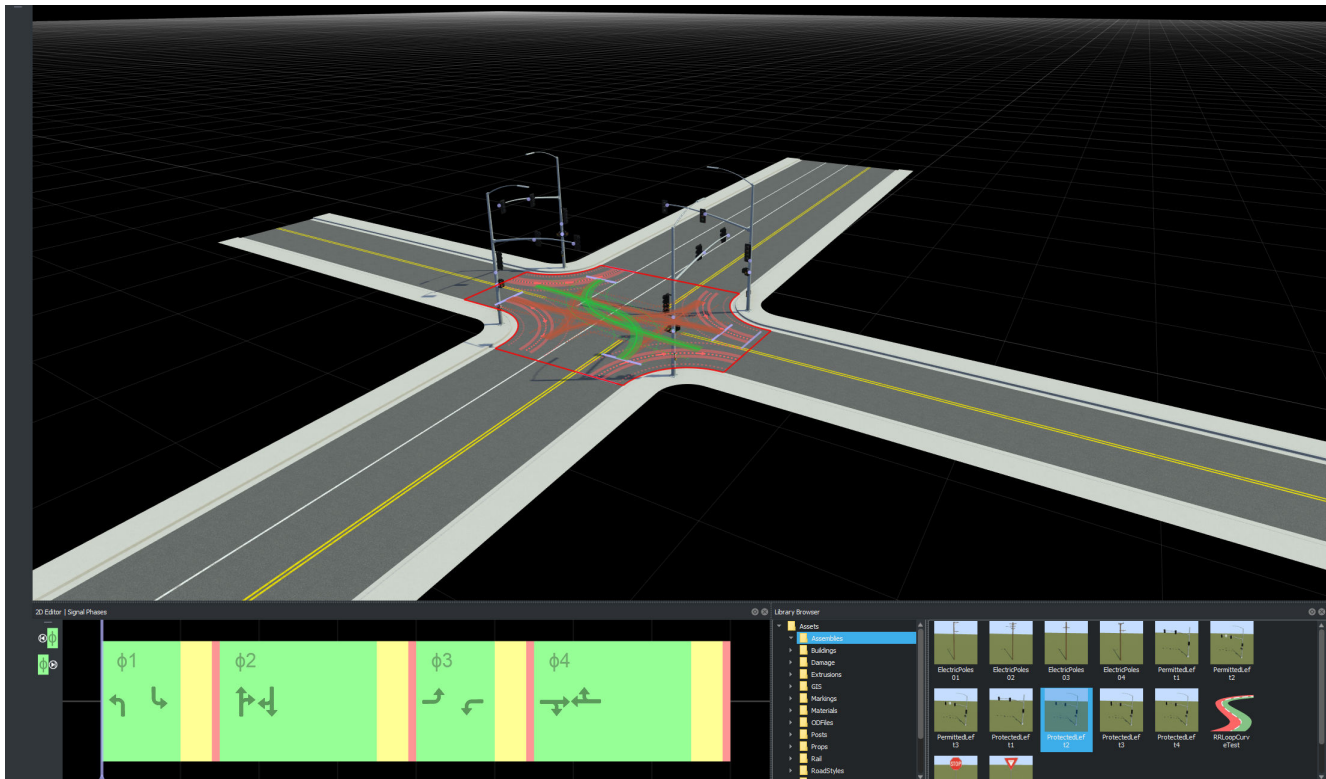
Inspect Phases and Maneuver Roads

You can now inspect and edit maneuver roads, signal phases, and intervals using the **Signal Phase Editor** in the **2D Editor** pane.

A phase indicates which signals are active and the state of the maneuver roads, for example, whether traffic may enter the junction along a given maneuver road.

This example shows four phases with three Red-Yellow-Green intervals per phase. An interval is a period in a junction that corresponds to allowed movements. You can drag the timeline bar across phases and observe the maneuver roads changing.

The highlighted (green) maneuver roads in the scene correspond to the first phase (highlighted by a red box) in the **Signal Phase Editor**. The phase also shows the road maneuver direction symbols. That is, the highlighted green maneuver road matches the road symbol in the current phase.
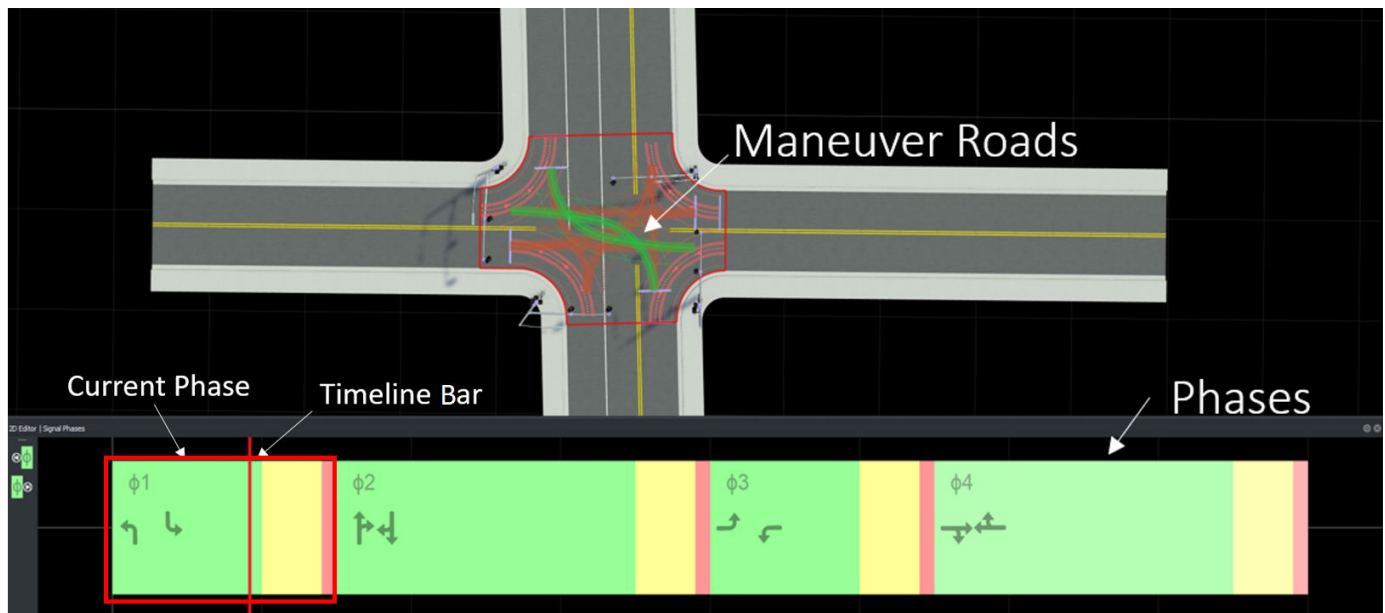
The phases, maneuver roads, and signal assets are automatically linked to the junctions. The gray dotted line shows road gates that link the signal to its corresponding junction.
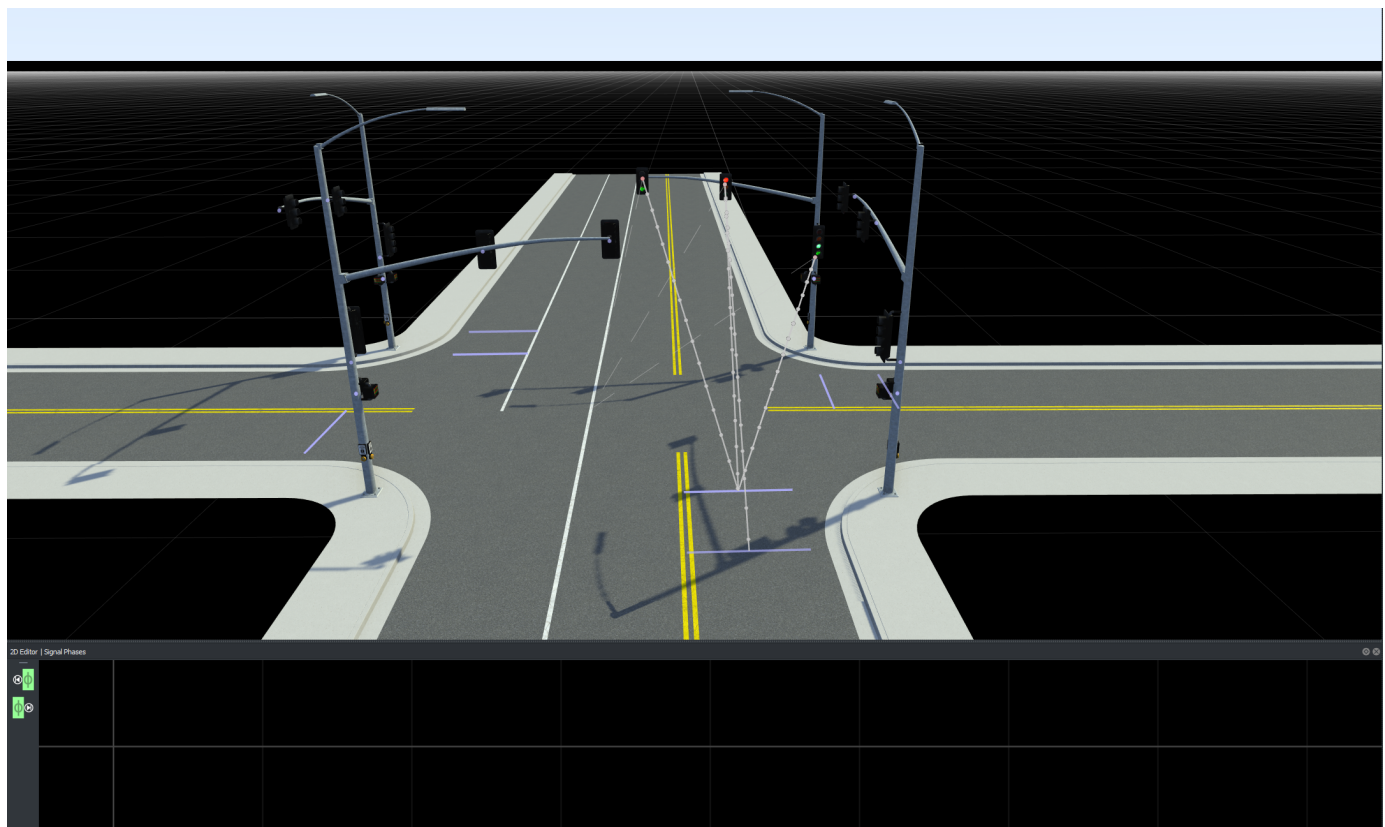
Click the purple junction gate to view its corresponding signal.
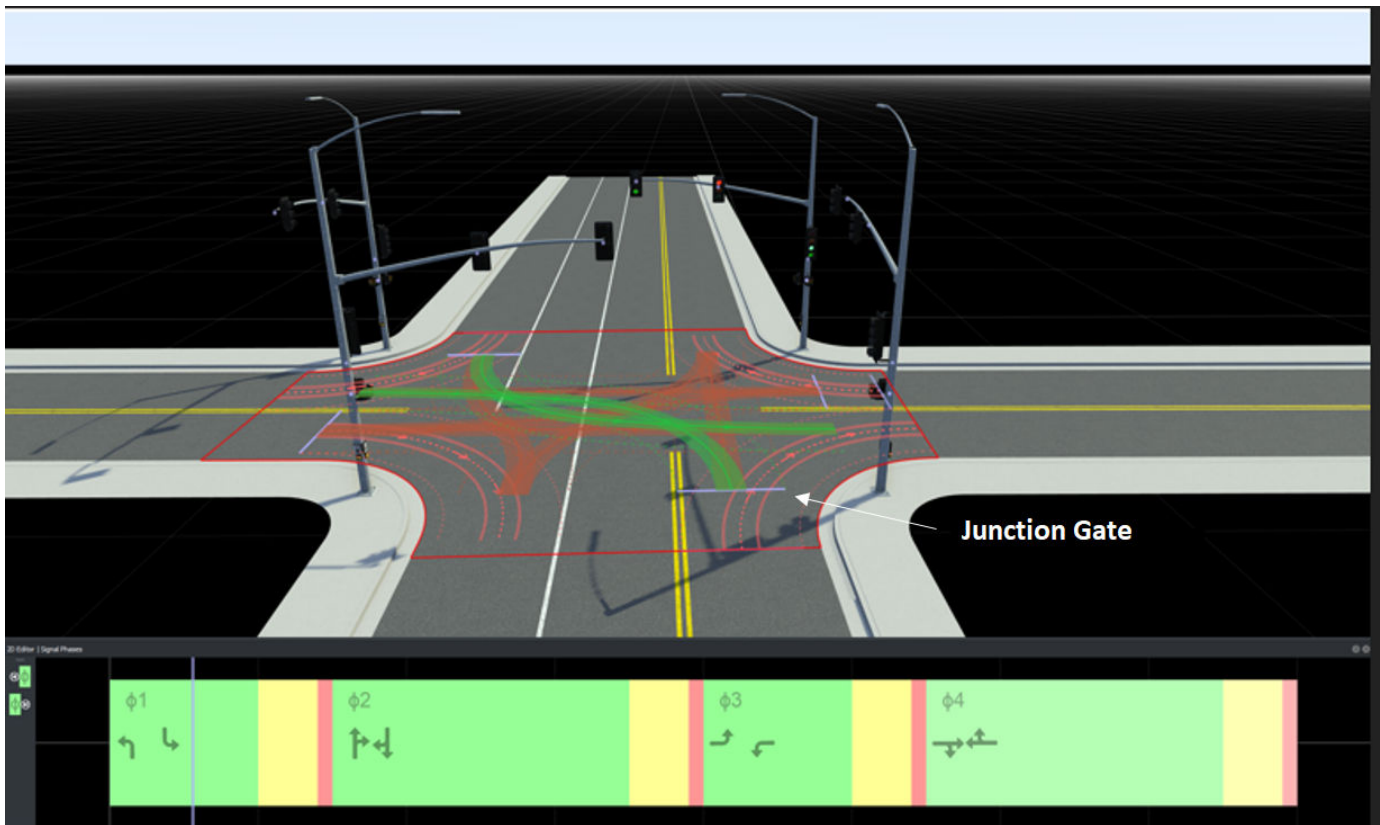
Use the timeline bar to navigate across phases and observe the signal lights changing color with each interval.

Edit Signal Phases

You can add, remove, or navigate between signal phases using the **Signal Phase Editor** in the **2D Editor** pane.

Add and Delete Signal Phases

To add a signal phase to your scene:

1   Click the Signal Tool and select the junction you want to signalize.
2   Right-click in the **2D Editor** pane to create a phase. You can also right-click beyond the end of the phases to create additional empty phases.
3   To duplicate a phase, right-click on the existing phase.

To delete a phase, select individual phases in the **2D Editor** pane and press the **Delete** key. To delete all the phases, select the junction and press the **Delete** key or select **Edit > Delete**.

Navigate Between Signal Phases

In the **2D Editor** pane, you can either left-click the phase, use **Next Phase** and **Previous Phase** on the left, or drag the **timeline bar** across the phases to preview the phases and the corresponding maneuver roads.
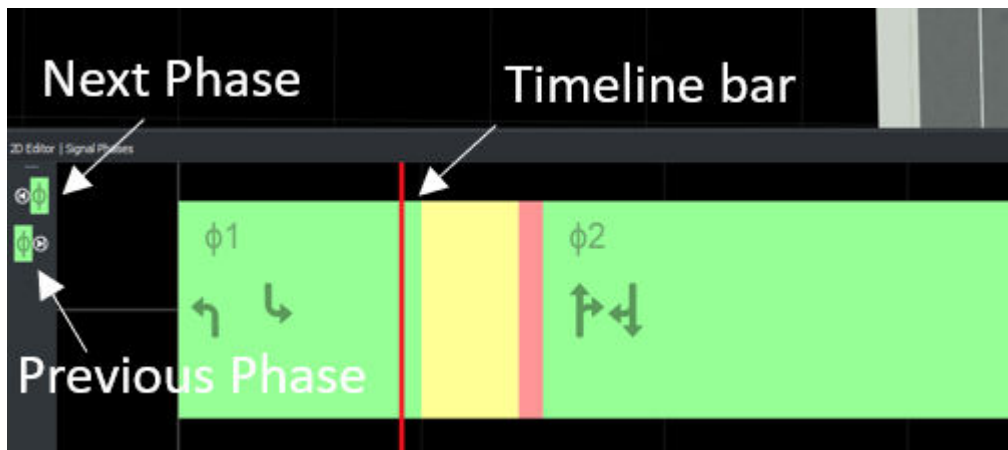
See Also

Related Examples

- Signal Tool
- Custom Junction Tool
- "Create Simple RoadRunner Scene" on page 1-19
- "Create Roads Around Imported GIS Assets" on page 1-57

External Websites

- Getting Started with RoadRunner

RoadRunner Fundamentals

RoadRunner Project and Scene System

In RoadRunner, a project contains assets that are shared by multiple RoadRunner scenes. You can create many scenes within the same project, and the scenes can share assets within the project.

Project assets include various components that are created from files such as 3D models, texture maps, and vector graphics. They also include various files specific to RoadRunner, such as materials and road marking styles.

Projects

When you run RoadRunner, you must select or create a project. The current project is displayed on the title bar. The current project is always the active project in RoadRunner.

Create New Project

1   Open the RoadRunner application, and on the **Scene** tab of the start page, click **New Scene**.

2   In the Select a Project window, click **New Project**, and browse for the folder in which you want to create the project. To create a project in a folder, the folder must be empty.

Note If you create a project that is on a network drive, changes made to asset files might not be automatically reflected in the **Library Browser**. In addition, performance might be slower. For improved performance and full **Library Browser** functionality, create a project on a local disk.

---

3   Select the assets you want to install with the project. By default, RoadRunner projects include a small assortment of materials, models, and other assets. You can also separately purchase the "RoadRunner Asset Library Add-On", which comes with a large array of generic and country-specific assets.

- If you have a RoadRunner Asset Library license, click **Yes** to populate the `Assets` folder with the library assets.

- If you do not have a license, this option is disabled. Instead, click **No** to populate the `Assets` folder with a set of default assets included with RoadRunner.

RoadRunner opens to a new scene that is created within the new project. The project contents are accessible from the folder in which you created the project.

Note It is recommended that you place the entire project folder under version control. For details, see the "Project and Scene Version Control" on page 2-4 section.

Project Folder Contents

A project folder contains these subfolders:

- `Project` — This folder contains a single file named `Project.rrproj`, which defines a unique reference ID for the project. Do not modify or move this file.

- `Scenes` – This folder stores individual scenes that use this project. When you first create a project, this folder contains several sample scenes. When you save a new scene for the current project, RoadRunner defaults to this folder.

- `Exports` – This folder is the default location to write out exported data from RoadRunner. When you first create a project, this folder is empty. You are not required to use this folder for exported data. For more information on exporting data, see "Export Scenes".

- `Assets` – This folder stores all asset files available for use in a scene. This folder and its subfolders appear in the **Library Browser**.
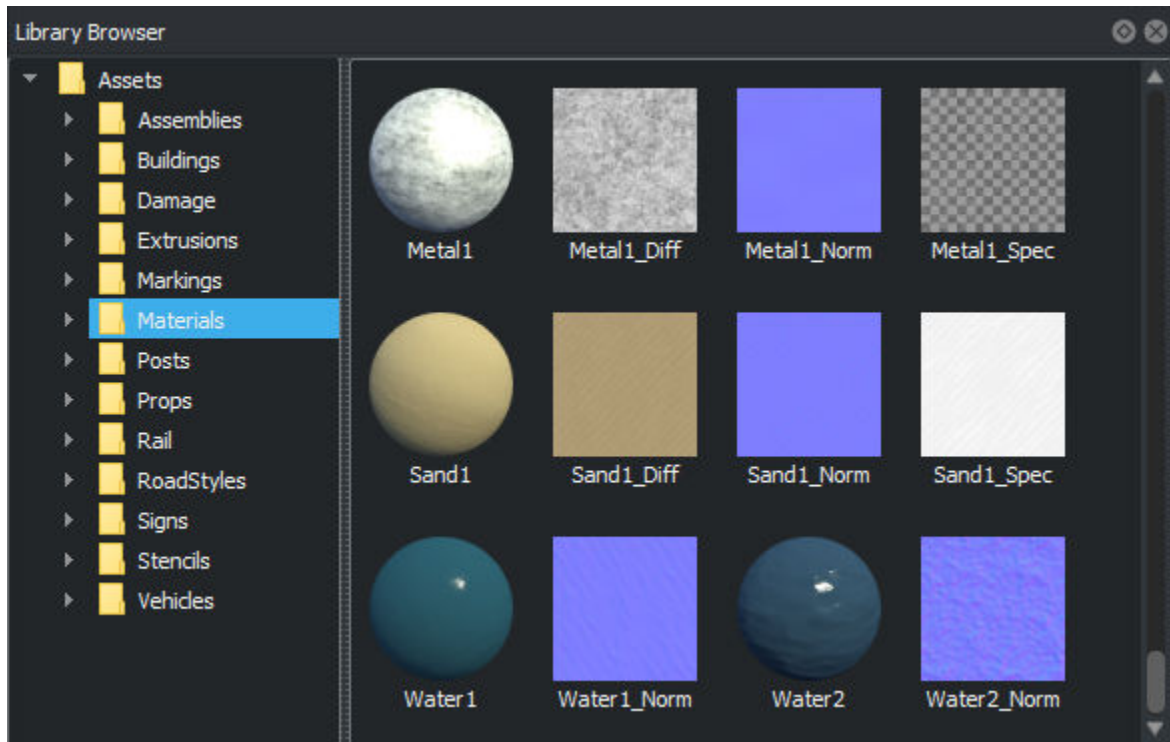
For every asset within the `Assets` tree, RoadRunner automatically creates an associated metafile with the `.rrmeta` file extension. The metafile contains additional data associated with the asset, the details of which vary for different asset types. The metafile also contains a unique ID, which you can use to identify a specific asset. Even if you rename an asset or move it into a different folder within the `Assets` folder tree, this ID does not change.

Always keep this metafile in the same location as the asset itself. Asset operations performed with the **Library Browser** automatically update the corresponding metafiles. For more details on working with assets, see "Create, Import, and Modify Assets" on page 2-50.

Change Current Project

To switch to a new project when you already have one open in RoadRunner, from the menu bar, select **File** and then **Change Project**. Then, in the Select a Project window, create a new project, browse for the root folder of a different project, or select a recently opened project from the **Recent Projects** list.

Save Project

To save a project, from the menu bar, select **File** and then **Save Project**. In addition, saving a scene also saves any changes you made to the current project. Modified assets are not saved until the current project is saved.

Scenes

A scene file contains an area that includes objects such as roads, surfaces, props, and other scene aspects. It is the main type of file edited in RoadRunner. Scenes can represent anything from a small area, such as a single intersection, to a large area, such as a portion of a city. A scene can contain multiple roads, intersections, road markings, props, terrain sections, and so on.

Individual scenes are saved as `.rrscene` files, typically in the `Scenes` folder of a project. You can create many scenes within the same project, and the scenes can share assets within the project.

RoadRunner has exactly one scene active at any given time. The name of the current scene is displayed in the title bar. If the scene has not yet been saved, the title bar displays the scene name as **New Scene**.

If you have unsaved changes in your current scene, RoadRunner prompts you to save your current scene before starting a new scene, loading an existing scene, or exiting the program.

Create New Scene

To create a new scene while working in an existing scene, on the menu bar, select **File** and then **New Scene**. Alternatively, press **Ctrl+N**.

To create a new scene from the start page, follow these steps:

1   On the **Scene** tab of the start page, click **New Scene**.
2   Select a previous project or click **New Project** to create your scene in a new project.

Open Existing Scene

To open an existing scene that was recently opened, open it directly from the **Recent Scenes** menu under **File**. Alternatively, select **File** and then **Open**, or press **Ctrl+O**, and then browse for the scene you want to open.

If the selected scene file is in the `Scenes` subfolder of a project folder, that project is loaded automatically. If your scene is saved elsewhere, the scene stores the relative path to the project directory.

If RoadRunner is unable to find or load the project, the software provides the option of selecting a previous project or browsing for a different one.

Save Scene

To save a scene, from the menu bar, select **File** and then **Save Scene**. Saving a scene also saves any changes you have made to the current project. Modified assets are not saved until the current project is saved.

Project and Scene Version Control

When multiple people are making changes to project assets or scenes, using a revision control system such as Git™ to control file versions. With a revision control system, follow these policies:

- Manage files, including metafiles (`.rrmeta`), within the project tree with the revision control system.

- Do not merge any RoadRunner internal files because they are binary files.
- Make sure only one user makes changes to any particular file at any time.

See Also

Related Examples
- "Create, Import, and Modify Assets" on page 2-50
- "Create Simple RoadRunner Scene" on page 1-19
- "RoadRunner Asset Types" on page 2-45

Window Layouts

The RoadRunner user interface is organized into panes. You can customize the layout of the pane to meet your preferences by resizing the panes and moving them into different configurations. RoadRunner preserves the window layout from session to session.

You can save and name up to five different layouts. You can restore saved layouts by using the layout controls at the bottom of the window menu.

Switch Between Tabbed Panes
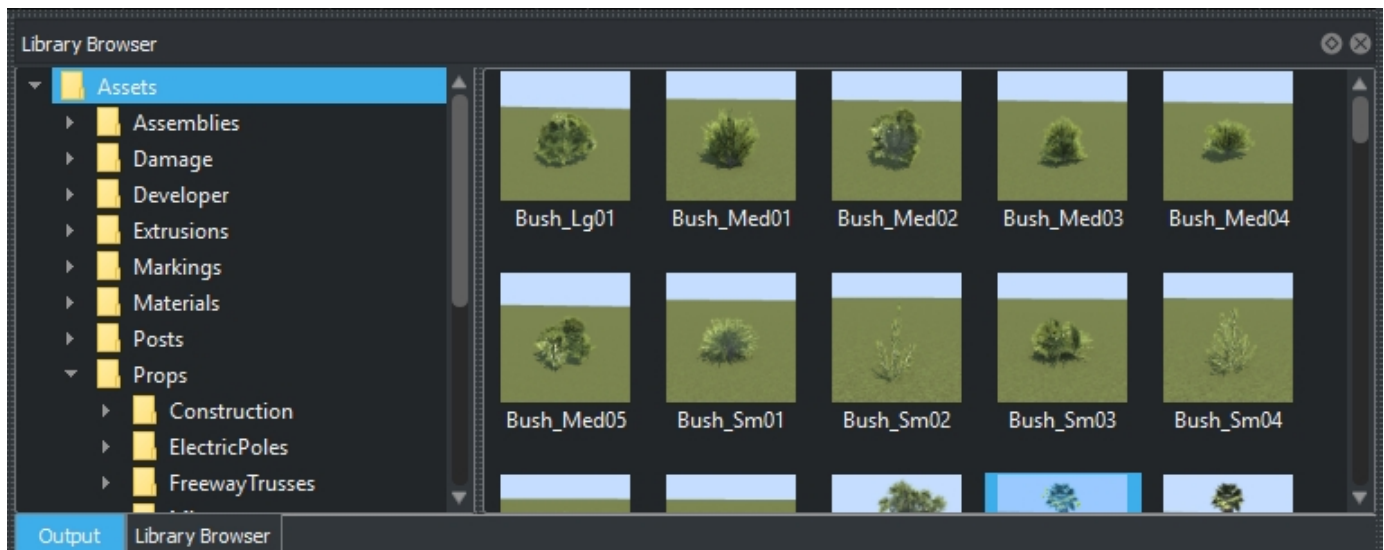
Panes can be stacked on top of each other. By default, the **Output** pane and **Library Browser** pane are tabbed.

To switch between tabbed panes, click the applicable tab at the bottom of the pane.

Undock a Pane
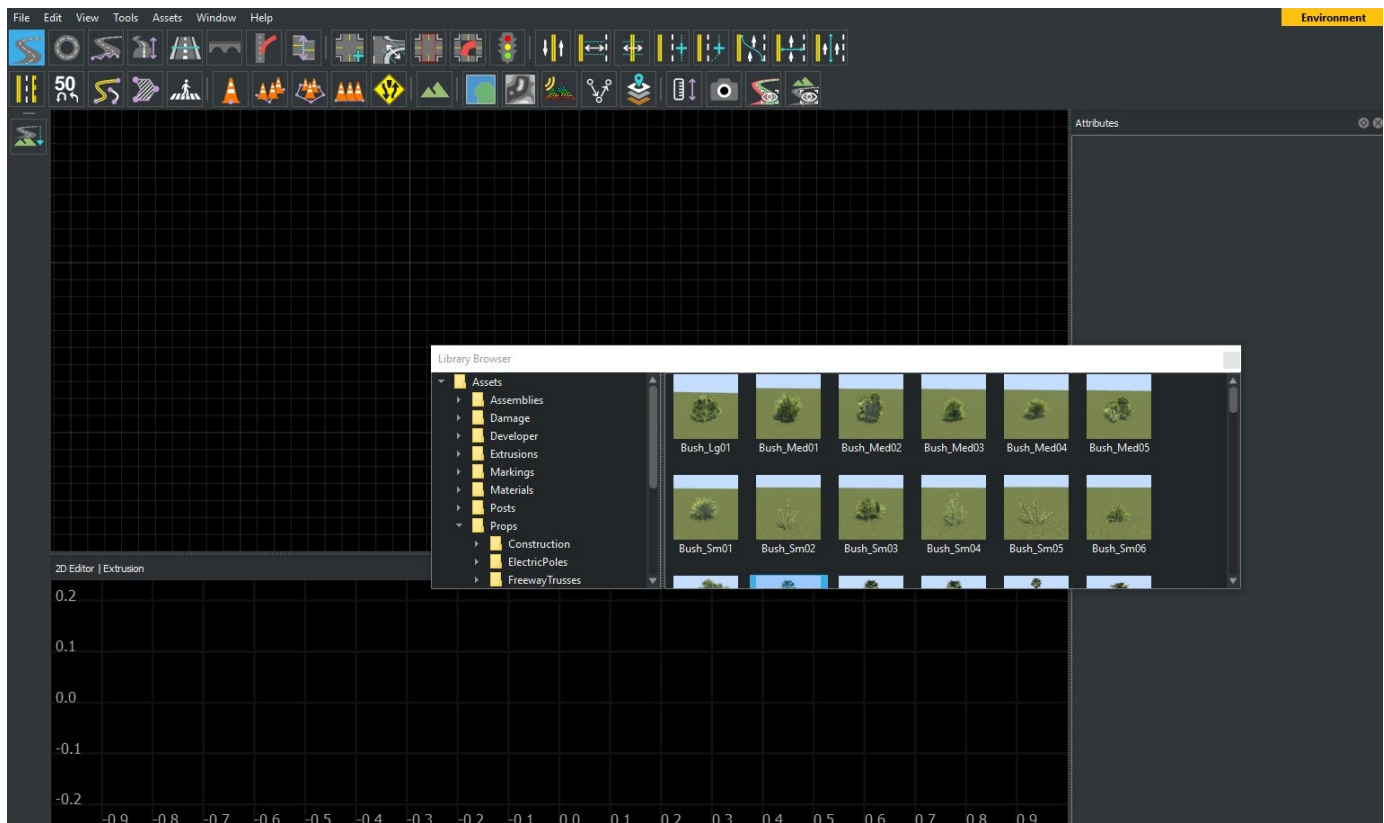
You can move a pane to a separate window (for example, to move it to a different monitor) by clicking and dragging the top of the pane. After moving a pane to a new window, you can move and resize this pane independently from the main application.

Dock a Pane
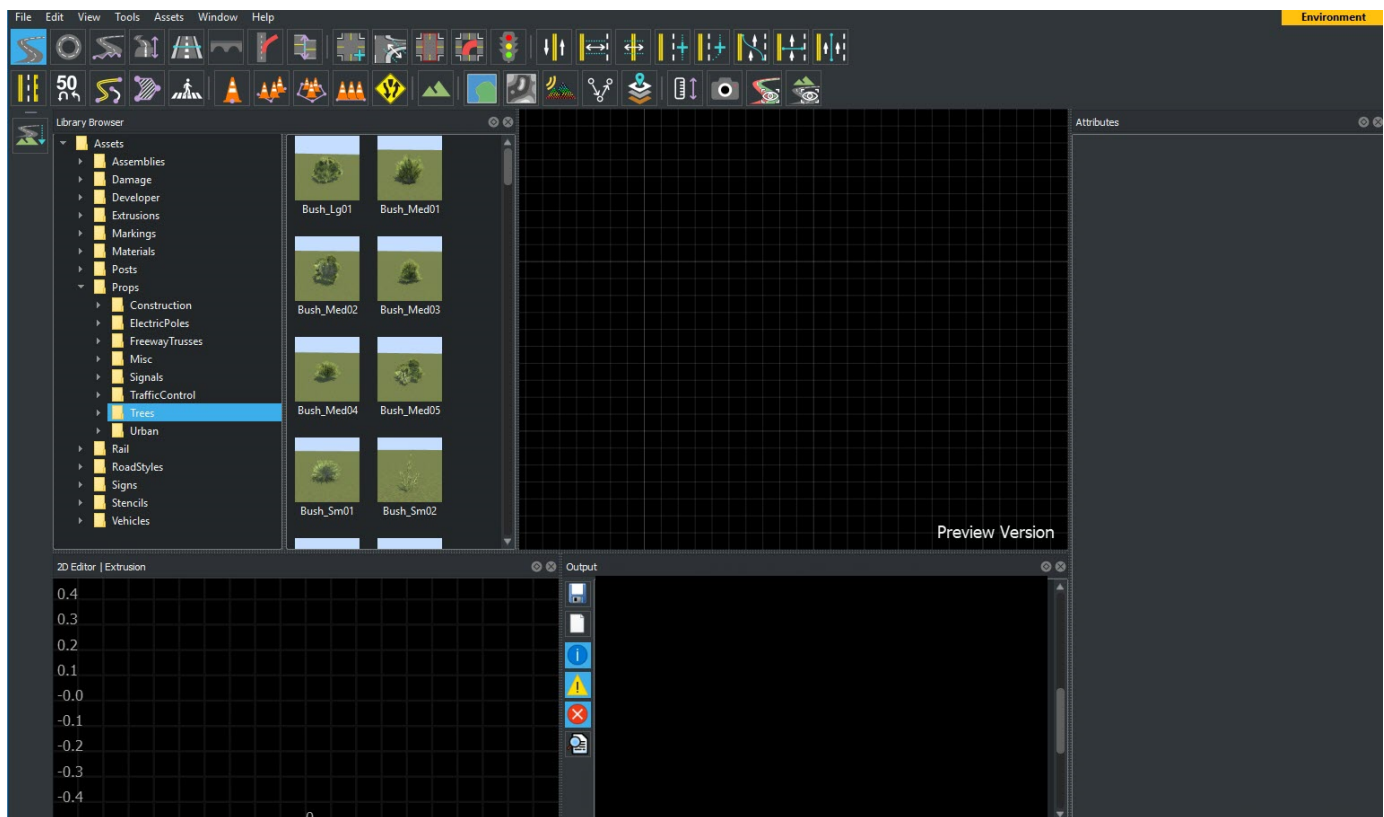
You can change where panes are docked in the application. This option can be useful for making better use of screen real estate on wide monitors.

1 Click and drag the top of the pane (this action works for docked and undocked panes).

2 Hover over the edge of the application where you would like to dock the pane.

- If you hover over the edge of the application without an existing pane, the pane will be docked to that edge.

- If you hover over the middle of another pane, the pane will be docked on top of that pane (that is, the panes will be tabbed).

- If you hover over the left or right side of a bottom pane (or the top or bottom of a side pane), then both pane will be displayed next to each other.

Save the Current Window Layout

1 Select the **Window** > **Save Layout** menu option. A dialog box prompts you to name the layout. If you already have five layouts saved, saving another layout will replace the oldest saved layout.

2 Type in the desired name of the layout. If you type in the name of an existing layout, the new layout will replace that existing layout.

Restore a Saved Window Layout

Select the **Window > Apply Layout > (layout)** menu option, where (layout) is the name of the layout you want to restore.

Delete a Saved Window Layout

Select the **Window > Delete Layout > (layout)** menu option, where (layout) is the name of the layout you want to delete.

Reset the Window Layout to the Default Layout

Select the **Window > Reset Layout** menu option.

Coordinate Space and Georeferencing

Local Coordinate System
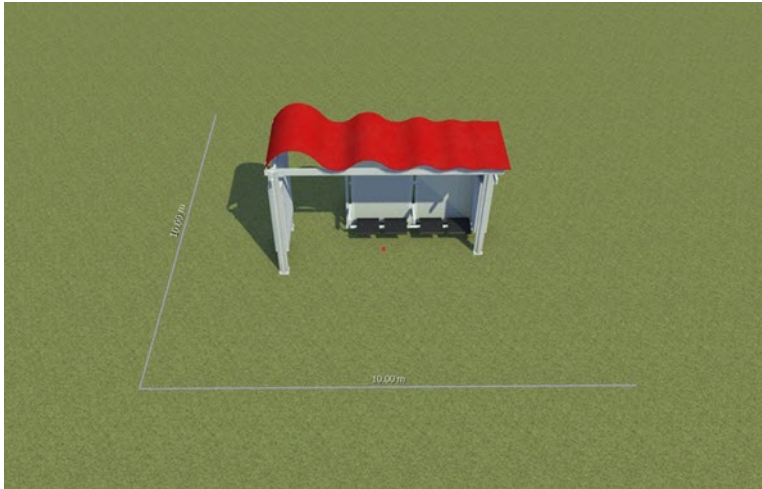
3D coordinates are displayed and edited in a right-handed Cartesian coordinate system. All spatial units are represented in meters, and angles are represented in degrees.

The 'X' and 'Y' dimensions represent 'Easting' and 'Northing' directions, respectively. The 'Z' dimension is height.

This table illustrates local object transformations along each axis (for example, when using the **Prop Point Tool**). Each image shows a transformation of the prop in the image in the positive direction for each axis.

	X (Easting)	Y (Northing)	Z (Height)
Move			
Rotate			
Scale			

Georeferencing (Geographic Coordinates and Projections)

RoadRunner scenes can be optionally georeferenced, which means that coordinates in the scene can be mapped to locations on the Earth. This mapping is important when you want to model a real-world location by using GIS reference data. For more details, see "Import Scene Data".

Georeferencing Basics

Georeferencing is a varied and complex topic. RoadRunner hides most of this complexity, especially if you are using well-formed GIS reference data.

In many cases, georeferencing data is carried through when exporting. If you want to align exported data with other GIS data (such as a GPS trace), then a basic familiarity with geospatial transformations is required.

To perform geospatial coordinate transformations, RoadRunner uses the PROJ library, which is a robust and industry-standard library for transforming horizontal and vertical coordinate systems. If you need to work with georeferenced data in your own application stack, you can use PROJ for optimal robustness and compatibility (or use a library that uses PROJ internally, such as GDAL or PDAL).

Georeference a Scene

To add or modify a scene's location on the Earth, use the **World Settings Tool**. An initial location is also applied automatically when first dragging any GIS asset into a nongeoreferenced scene.

Georeferenced Coordinate System

RoadRunner supports a variety of input projections and datums when loading external GIS data. However, all editing and displaying is performed in a specific georeferenced coordinate system.

Any external GIS data is transformed automatically into this coordinate system before it is displayed.

Horizontal Georeferenced Coordinate System

To map the X and Y coordinates of the Local Coordinate System on to the Earth, an application must define a horizontal coordinate system (typically by defining a geospatial projection and datum).

RoadRunner uses a coordinate system that reduces scale and rotational distortion surrounding (within ~100 km of) a latitude/longitude point of interest. You can control the latitude/longitude point (using the **World Settings Tool**), but control over the projection is not permitted.

Specifically, RoadRunner uses a Transverse Mercator projection (with a scale factor of 1.0) over the WGS84 datum. For example, a scene centered at a latitude of 32.0 and a longitude of -118 has a horizontal georeferenced coordinate system defined as (in Proj syntax):

```
+proj=tmerc +lat_0=32.0 +lon_0=-118.0 +k=1 +x_0=0 +y_0=0 +datum=WGS84 +units=m
```

Vertical Georeferenced Coordinate System

To map Z (height) coordinates of the Local Coordinate System on to the Earth, an application must define a vertical coordinate system.

Roadrunner uses heights over the EGM96 Geoid, as defined by a 15-minute grid (such as the one found here). Grid files are used to convert between WGS84 ellipsoid heights and geoidal heights.

You can find the exact grid file used by RoadRunner by searching for the "egm96_15.gtx" file in the RoadRunner installation directory.

The vertical coordinate system is also defined in the PROJ string. The full PROJ string for the example in the horizontal section above is:

```
+proj=tmerc +lat_0=32 +lon_0=-118 +k=1 +x_0=0 +y_0=0 +datum=WGS84 +units=m +geoidgrids=egm96_15.gtx +vunits=m +no_defs
```

Georeferencing and Exported Data

In addition to the information in the Georeferencing Basics section, this section provides information about georeferencing information in data exported from RoadRunner.

To align data exported from RoadRunner with other GIS data (or to transform between latitude/longitude coordinates and coordinates in the export data), you must know the projection and datum of each data source. RoadRunner expresses this information as a Proj syntax or WKT string.

Many export formats also include projection information. For example, OpenDRIVE on page 5-24 data exported from RoadRunner includes a <georeference> tag defining the projection information as a PROJ string.

Note In almost all cases, it is not possible to align two georeferenced data sets by simply shifting them. Projection transformations are more complicated than a simple shift and scale. Instead, rely on a library like PROJ.

RoadRunner exports data in the same georeferenced coordinate system used by the scene (see Georeferencing Basics). You can view the PROJ/WKT strings for the current scene in the **World Settings Tool**. Control for transforming data into a different projection during export is not supported.

Exported Data and Grid Files

The exported data also uses the same vertical coordinate system as the scene itself (see Georeferencing Basics). To interpret the elevations in the exported data, you might need to make use of the same grid files used by RoadRunner. This might require supplying the grid files to your external application (if not already present).

In some specific cases, you might be able to ignore the grid files. Examples of these types of cases include if you do not need to vertically match exported RoadRunner data and external GIS data or if all of your external GIS data is already using the same vertical datum as the RoadRunner scene.

If you are confident that you do not need the grid files in your external application, you can remove the +geoidgrids=egm96_15.gtx portion of the PROJ string in any exported data. Because there

can be upwards of a 30 m vertical difference between geoidal heights and ellipsoidal heights, if you are confident in how the data is to be used downstream.

Manipulate Scene Objects

In RoadRunner, the tool you select can affect which objects in a scene that you can select, move, create, delete, or modify.

Select Objects

You can select objects in the scene editing canvas and in the **2D Editor** pane.

Most operations in RoadRunner require selecting one or more objects to act on. The attributes of the selected objects are displayed in the **Attributes** pane. Many operations, such as those in the left toolbar for a selected tool, apply to the currently selected objects.

The current tool defines which types of objects are selectable. For example, the **Road Plan Tool** permits the selection of roads but not props, whereas the **Prop Point Tool** permits the selection of prop points but not roads.

Some types of objects can be selected only after first selecting a parent object. For example, in the **Road Plan Tool**, you must first select a road before the control points for that road are displayed.

RoadRunner enables you to select multiple objects together. Some tools permit the selection of multiple different types of objects at once.

This screenshot of a simple scene was taken in the **Prop Polygon Tool**. The scene contains four prop polygons, which are displayed as light purple outlines. The sections that follow use images from this scene to show how object selection works.

Selection Colors

Most tools in RoadRunner use a common color language to indicate the selection state. This image shows polygons in three different selection states.
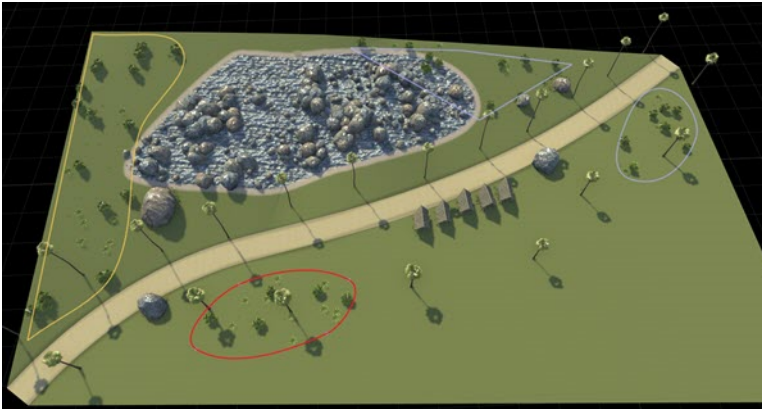
- The light purple polygons are objects that are not currently selected.
- The red polygon contains the objects that are currently selected.
- The yellow polygons contain the objects that the mouse is pointing to. This state provides a visual indication of the object that will be selected if you click.

A fourth color, gray, is also used when selecting overlapping objects. See Cycle-Select Overlapping Objects.

Select Single Object

To select an object, click the object in the scene. This action deselects any previously selected objects and selects the object the mouse is pointing to. In this image, the selection that was previously being pointed to has been clicked and is now selected.

Add Object to Selected Objects

To select an additional object, hold **Shift** and click an unselected object in the scene. In this image, an additional object has been added to the previous selection.

2-15

To remove an object from selection, hold **Ctrl+Shift**, and then click a selected object in the scene. This action removes that object from the selected objects, leaving the remaining objects selected.

Alternatively, to add or remove selected objects, hold **Shift** or **Ctrl+Shift** and perform a box select.

Box Select a Group of Objects

To perform a box selection, in the scene editing canvas, click and drag to draw a rectangle around the objects you want to select. There are two box selection options:

- Overlap Box Selection — Select any objects that touch the box.
- Containment Box Selection — Select only objects that are fully contained within the box.

The direction in which you draw the box dictates which selection type is used. This table indicates the selection type according to the direction in which you draw the box.

Toward top-left	Toward top-right
Containment	Overlap
Toward bottom-left	**Toward bottom-right**
Overlap	Overlap

In this image of overlap box selection, both polygons that are at least partially within the region are selected.
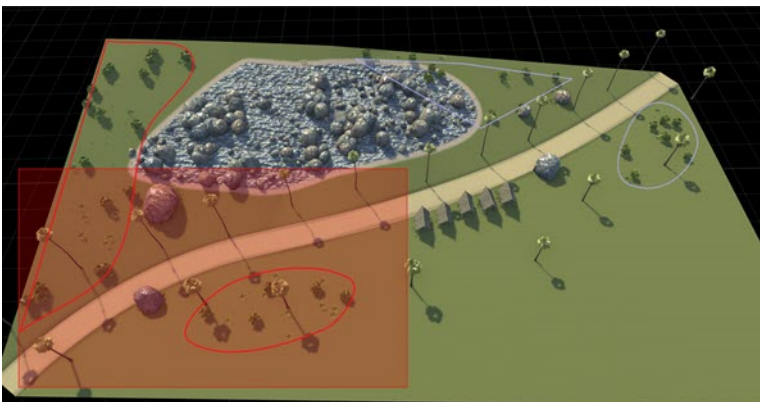
To perform an overlap box selection, click and drag in one of the directions indicated by the red boxes in the previous table. A red box appears, and any objects touching that box are selected. Optionally, hold **Shift** to add the objects to the selected set. Hold **Ctrl+Shift** to remove the objects from the selected set.

Tip If no box appears, check that you did not click on or inside a draggable object in the scene. If you hold **Shift** prior to the click, a box selection occurs, even if your drag starts outside of a selectable object.

In this image of containment box selection, only the polygon that is fully contained within the region is selected.
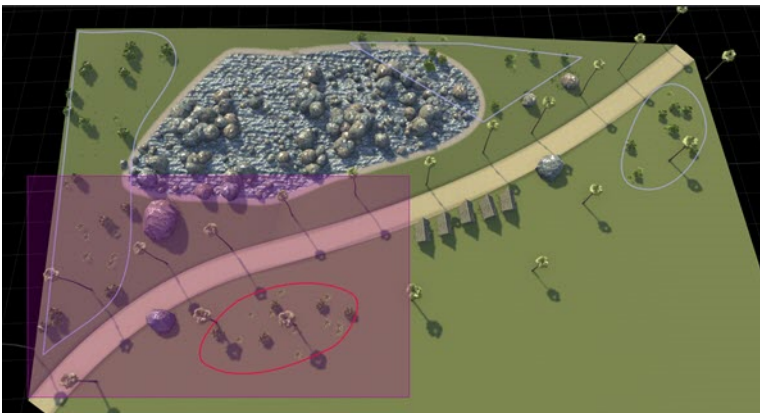
To perform a containment box selection, click and drag in the upper-left direction. A purple box appears, and only those objects fully contained within that box are selected. Optionally, hold **Shift** to add the objects to the selected set. Hold **Ctrl+Shift** to remove the objects from the selected set.

Tip If no box appears, then it is likely that you started the box on or inside a selectable object in the scene. Check that your drag starts outside of a selectable object.

Select All Objects

To select all objects in the scene, from the menu bar, select **Edit** and then **Select All**, or press **Ctrl +A**. This image shows all of the prop polygons selected.
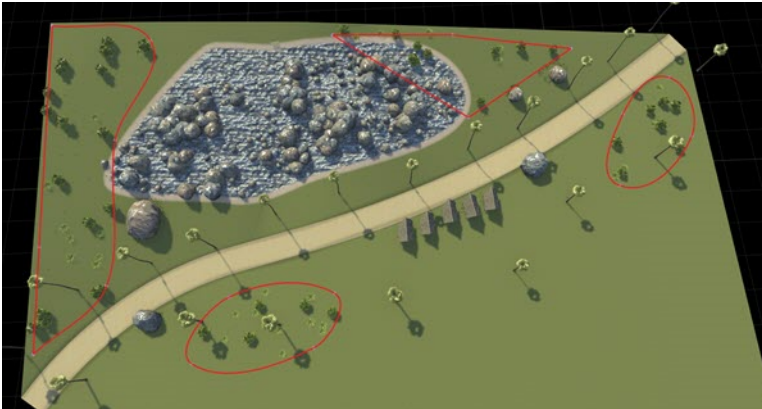
The behavior of a select all action depends on which objects you currently have selected.

- If you have no objects selected, then all selectable objects in the scene are selected.
- If you have objects selected, and any of those objects have unselected child objects, then the unselected child objects are selected. For example, if you select a single prop polygon using the **Prop Polygon Tool**, then the points on that polygon are displayed but are not selected. Performing a select all operation selects all the points on that polygon, not other polygons in the scene.
- If you have objects selected, and all child objects are already selected (or no child objects exist), then all selectable objects in the scene are selected.

Deselect All Objects

To deselect all objects in the scene, from the menu bar, select **Edit** and then **Deselect All**, or press **Ctrl+D**. This image shows all previously selected polygons now deselected.

Cycle-Select Overlapping Objects

Sometimes multiple selectable objects overlap each other. For example, this image shows three overlapping prop polygon objects.

In these cases, you can cycle between the different objects by repeatedly clicking on the overlapping portion. Each click selects the next overlapping object.

1   Move the pointer over the area where the object overlap, which in this case is in the middle of the three overlapping polygons. The first object you can select displays in yellow, while the other overlapping objects display in gray.

2   Click the overlapping portion to select the object. The selected object displays in red, and the object you can select on the next click displays in yellow.

3   Continue clicking to cycle through the overlapping objects until you reach the object you want to select. This image shows the next cycle in the selection.

Move Objects

Default Tool Translation and Rotation

For tools that enable you to move objects in the scene by selecting and dragging them, follow these steps:

1 Select one or more objects.
2 Click and drag a selected object to move it. If multiple objects are selected, dragging one object moves all of the selected objects.

The exact behavior when moving objects depends on the specific tool and type of object. For example:

- Moving props in one of the prop tools automatically projects their heights to the ground surface. For more details on prop tools, see "Props and Signs".
- Moving lane marking nodes in the **Lane Marking Tool** are constrained to lie along the lane boundary curve.
- Moving a road control point in the **Road Plan Tool** can automatically update other roads to enforce tangential continuity.

You can also move many objects during their initial creation.

Translation Tooltip

After selecting an object using the **Selection Tool**, road tools, or property tools, select **View > Translate** to display the translation manipulator, shown in this image. Alternatively, you can select the translation manipulator by clicking the [button image] button from the vertical menu on the left side of the screen.

Click and drag the red, blue, or green arrow to translate the object in the horizontal and vertical directions, respectively. This image shows the rock translated in the horizontal direction.
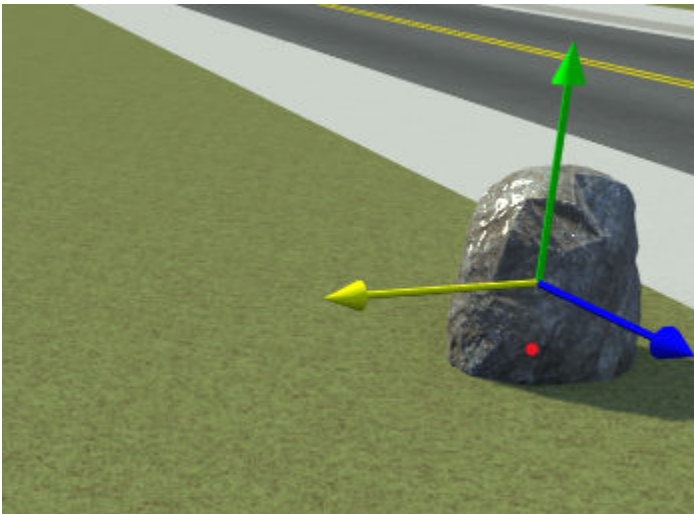
Rotation Tooltip

Use the rotation manipulators to seamlessly rotate a single object or a group of objects interactively within a scene. Rotation manipulators support the rotation of these objects in a RoadRunner scene:

- Roads and their slip connections
- Surfaces
- Buildings
- Props (rotation along control points, curves, and polygons)

- Markings (rotation along control points, curves, polygons, and parking spaces)

To rotate a single object, select the **Prop Point Tool** and then select a prop in the scene. From the **View** menu, select **Rotate**.
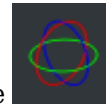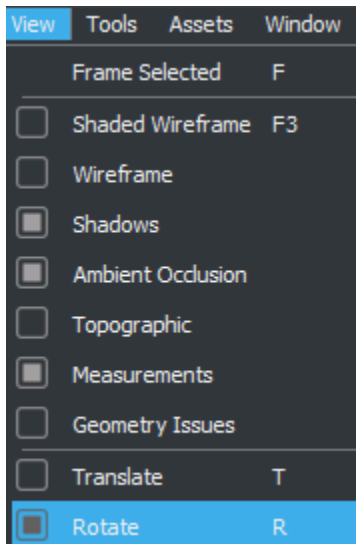
Alternatively, you can select the rotation manipulator by clicking the button from the vertical menu on the left side of the screen.

Three rings, representing rotation around the x-, y-, and z- axes, appear on the scene canvas around the selected object.
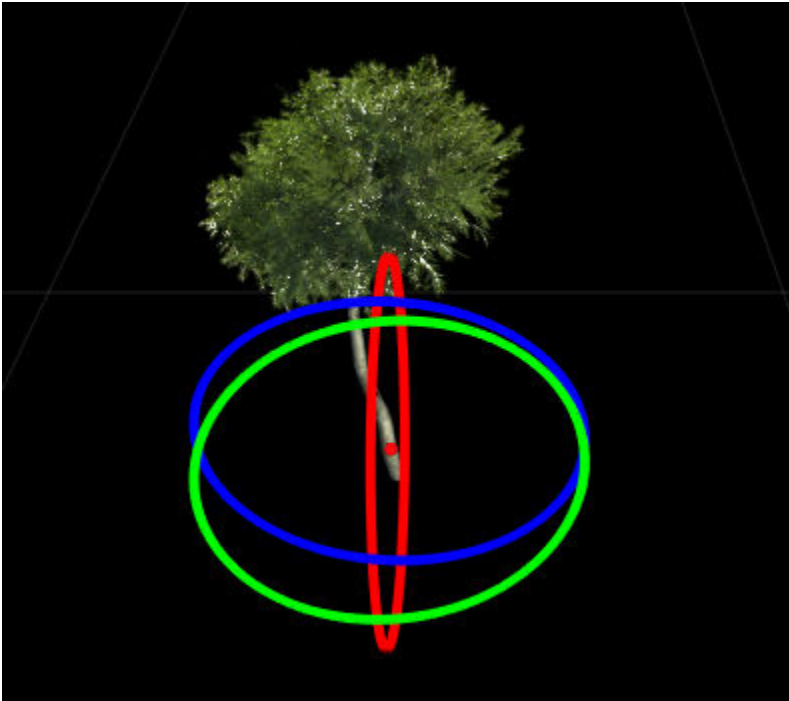
Drag the rings to make rotational adjustments to the object. During rotation, the selected ring becomes yellow and the other two rings disappear from the canvas. The angle indicator alongside the selected ring indicates the angle of rotation along the axis of rotation around the selected axis. These images show the rotation of a tree prop along the *x*- and *z*- axes, respectively.
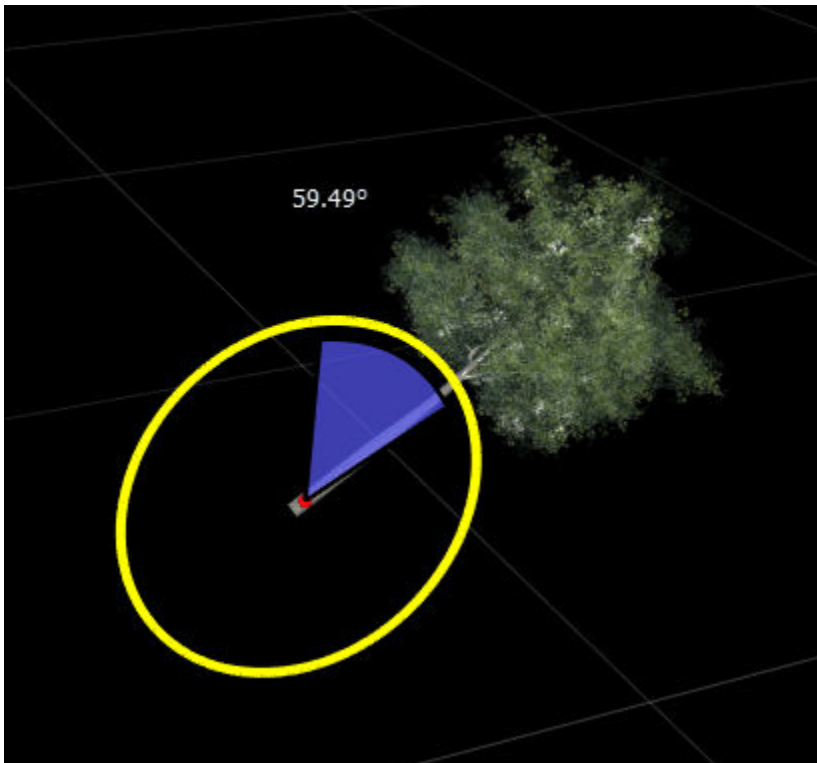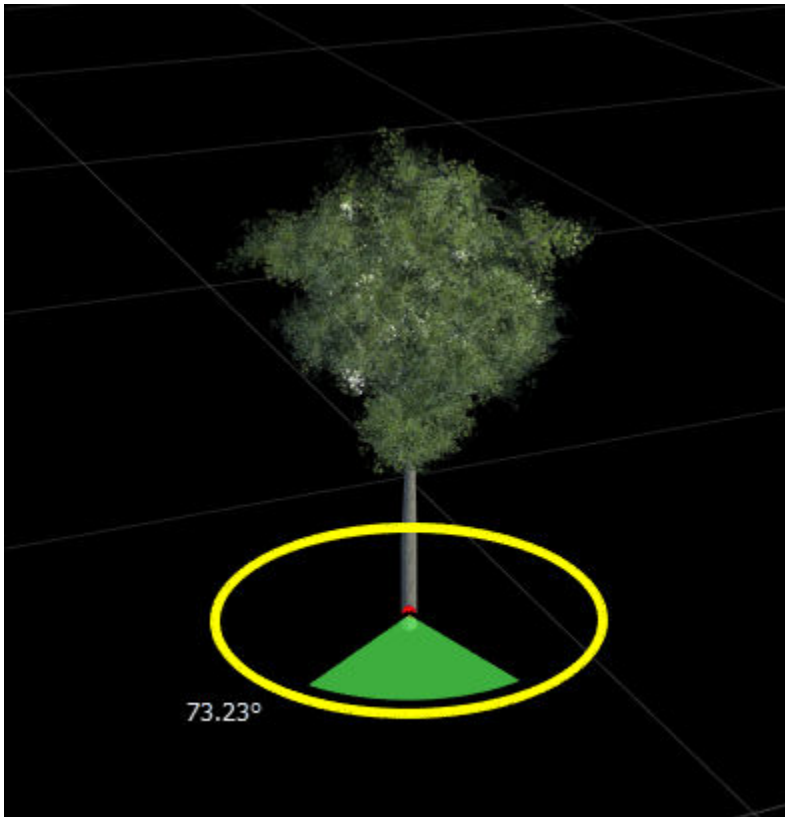
Note Object rotation in RoadRunner scenes is compound. If you rotate an object around one axis (for example, the x- axis), this affects the angle of rotation around the other two axes (y- and z- axes).

To rotate a group of objects, first select the **Selection Tool**. Then, click the scene canvas and drag to select all the objects in the rotation group. For more information on selecting multiple objects at once, see "Select Objects" on page 2-14.

From the **View** menu, select **Rotate**. Alternatively, you can select the rotation manipulator by clicking

the [button] button from the vertical menu on the left side of the screen.

Three rings, representing rotation around the x-, y-, and z- axes, appear on the scene canvas around the center point of the selected objects.
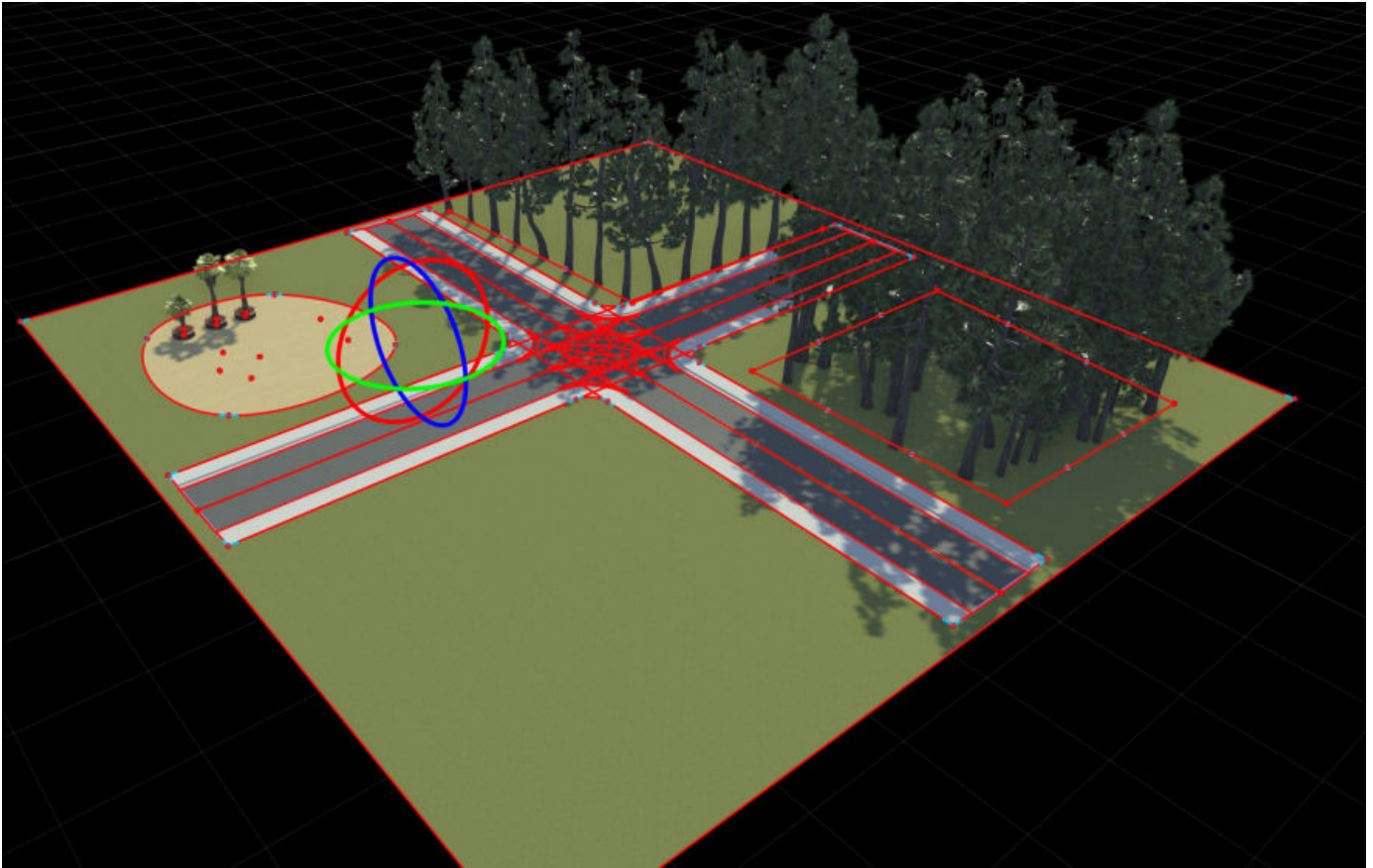
Drag the rings to make rotational adjustments to the selected objects. The angle indicator alongside the selected ring indicates the angle of rotation around the selected axis.
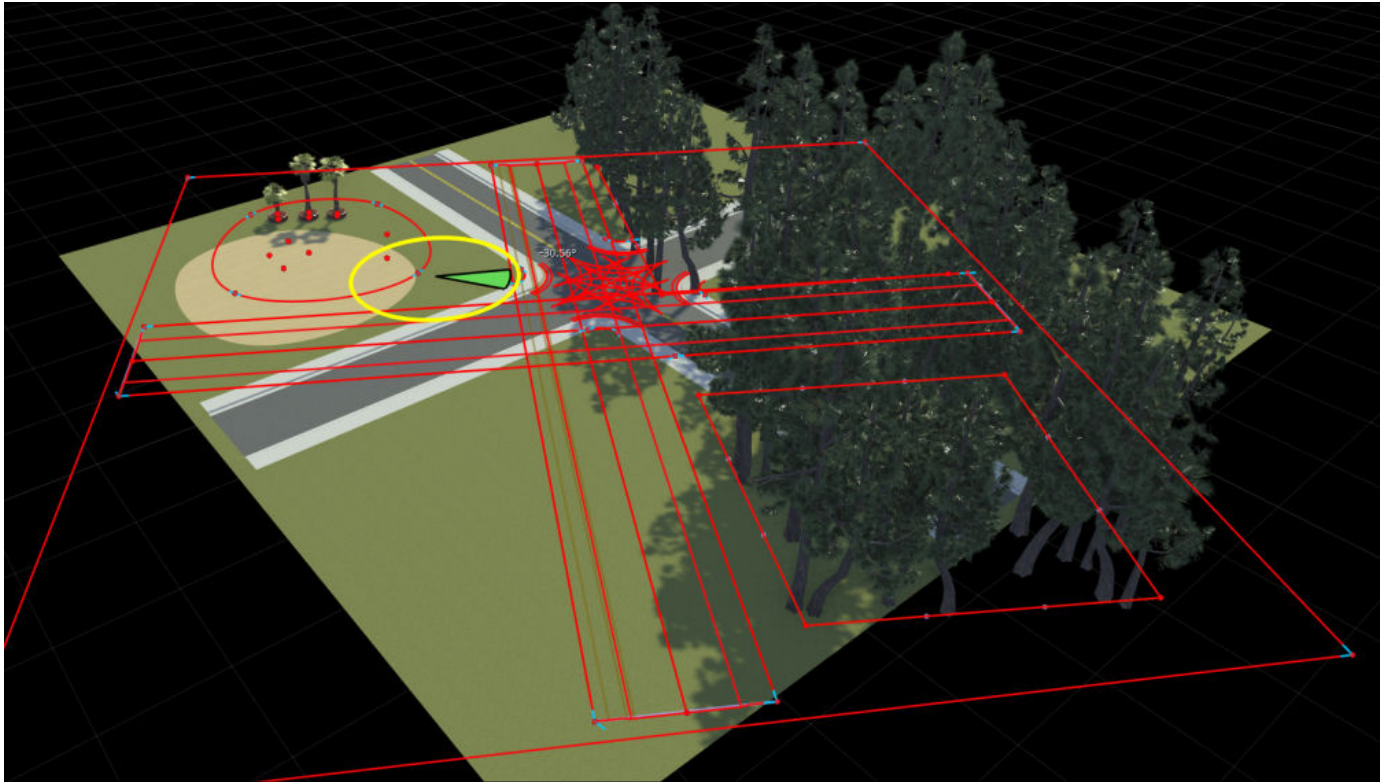
The new rotation angle does not apply to the selected objects. The objects in the scene maintain their relative positions to the center point, and characteristics such as slip roads, corners, and maneuver roads are preserved.
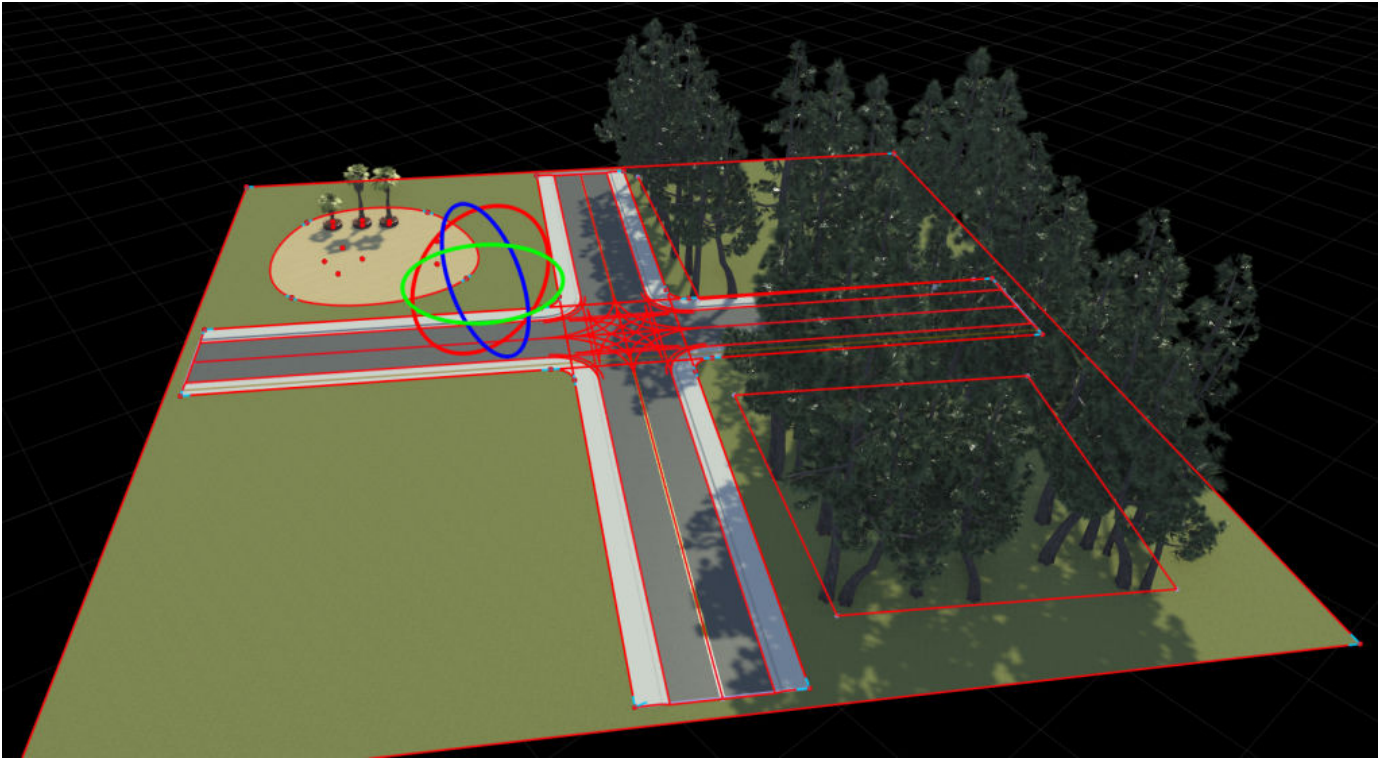
Note Roads in RoadRunner scenes support rotation around the z- axis, because the control points along the roads do not support rotation along the x- and y- axes.

A selected set of props rotate around the center point of the selection, such that all props maintain their starting distance from to that center point. In this image, three selected trees rotate around a center point, maintaining their initial distance from the center point.
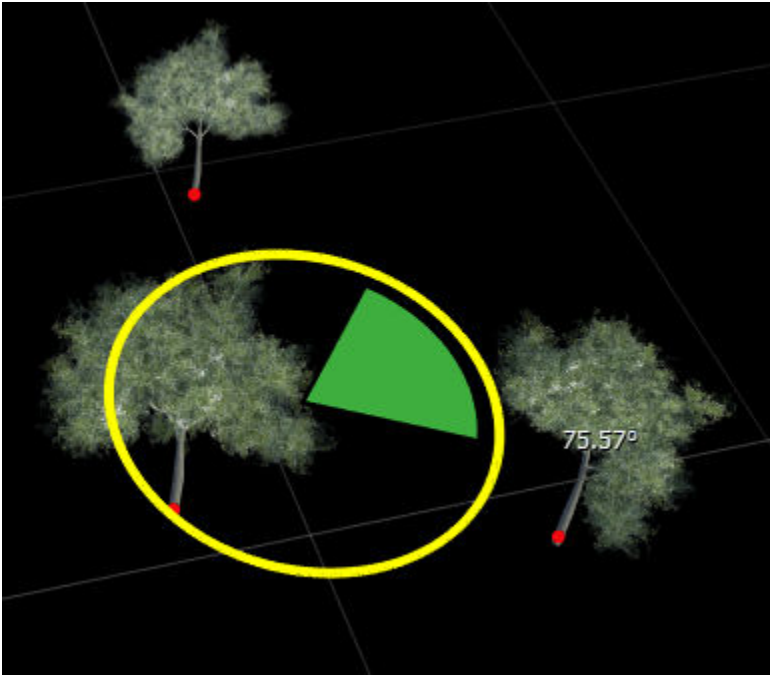
Create Objects

Many tools provide the ability to create objects. The type of object created and the specific creation steps depend on the tool. For step-by-step creation instructions, see the documentation for the specific tool.

In most cases, you can right-click either an existing object or an empty location in the scene to create an object. Often, you can keep holding down the right-click button to simultaneously create and drag the object.

Depending on the specific tool, you might need to first select an appropriate asset in the **Library Browser**. Some tools require an asset to be selected, while others will change their behavior depending on whether or not an asset is selected.

Some types of objects can be created by clicking an asset in the **Library Browser** and dragging the object into the scene. For example, dragging **Prop Model Assets** into the scene adds prop points and automatically switches the currently selected tool to the **Prop Point Tool**.

Delete Objects

Most tools enable you to delete selected objects. You can refer to the documentation for a specific tool to learn about deleting objects by using it, however, the steps are often similar to these ones:

1   Select one or more objects.
2   Select the **Edit** option and then the **Delete** option in the menu bar, or press **Delete**.

If an object cannot be deleted, the bottom status bar typically displays an error message.

Modify Objects

When you select an object in a scene, you can then modify its properties in the **Attributes** pane. If you select multiple objects, then the **Attributes** pane provides additional features for interacting with them.

This **Attributes** pane shows the selection of two props created by the **Prop Point Tool**.
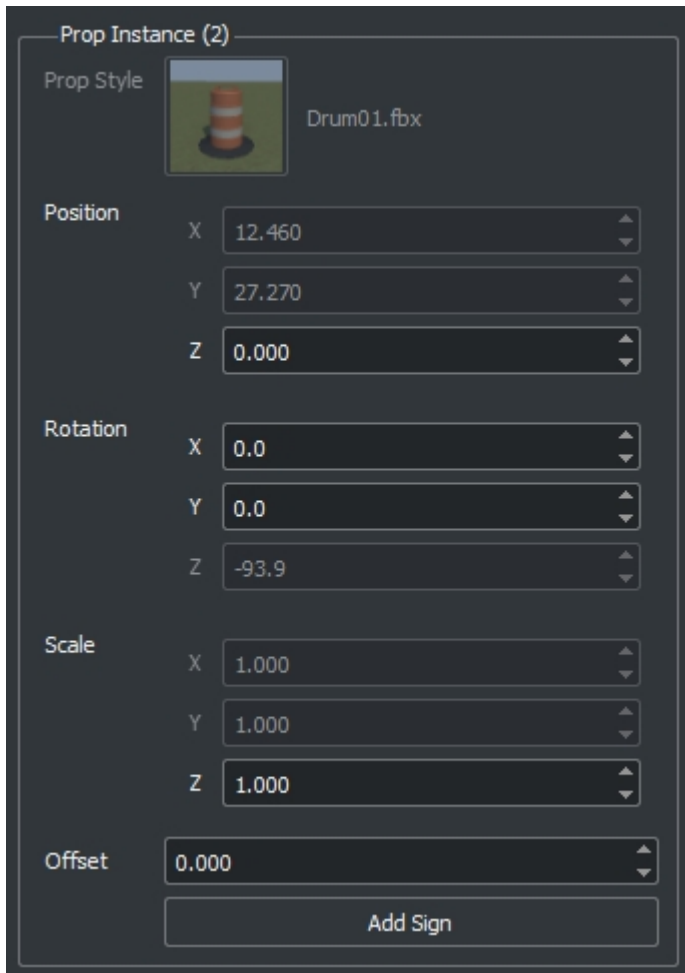
In this selection:

- The group label, **Prop Instance (2)**, shows the number of selected objects selected, which is two in this case.
- The **Prop Style** asset picker is dimmed, which indicates that the two props have different assets assigned to them.

  - The asset picker shows the style of the first selected object, which in this case is a drum barrel.
  - If the asset picker value changes, then the picker modifies all selected objects and the asset picker is longer dimmed.
- The selected objects have different `Position.X` and `Position.Y` values, but their height (`Position.Z`) value is the same. Modifying the `Position.X` or `Position.Y` values applies the same value to all selected objects.

- Clicking a button, such as the **Add Sign** button in the figure, applies the operation to all selected objects.

To modify an object to use a different asset, select an asset from the **Library Browser** and drag it onto the asset picker image in the **Attributes** pane.

Note Most asset pickers accept only certain types of assets. For example, you can assign **Prop Model Assets** and **Extrusion Assets** to a prop curve by using the **Prop Curve Tool**, but you cannot assign a **Material Assets**.

Some asset pickers enable you to have no asset assigned. To remove an asset from an asset picker, right-click the asset picker and select **Clear**. Alternatively, click the asset picker and press **Delete**.

To quickly locate the currently displayed asset in the **Library Browser**, click the asset picker. The asset is selected in the **Library Browser**.

Although the asset is selected, its attributes do not display. To see and modify the asset attributes, click the selected asset in the **Library Browser**.

See Also

Related Examples
- "Create, Import, and Modify Assets" on page 2-50
- "Choose a RoadRunner Tool" on page 2-35

Keyboard Shortcuts and Mouse Actions for RoadRunner

Editing

These operations apply to the editing window and in some cases panes such as the **2D Editor**, **Library Browser**, and **Attributes** panes.

Task	Action
Undo or redo.	**Ctrl+Z** or **Ctrl+Y** Undo and redo also apply to most selection and tool change actions. RoadRunner has an infinite undo and redo stack. This stack is related to the current scene or scenario. Operations that change the current scene or scenario, such as opening a different scene or creating a new scene, remove all actions from the stack.
Select all.	**Ctrl+A**
Deselect all.	**Ctrl+D**
Delete.	**Delete**
Copy.	**Ctrl+C**
Paste.	**Ctrl+V**

Object Selection and Manipulation

For more details about object selection and manipulation, see "Manipulate Scene Objects" on page 2-14.

Task	Action
Select an object.	Left-click.
Add an object to a selection.	Hold **Shift**, then click the object.
Remove an object from a selection.	Hold **Ctrl+Shift**, then click the object.
Select multiple objects.	Hold **Shift**, then left-click and drag to draw a box around the objects you want to select. • Drag the box toward the top-left of the canvas to select only objects that are fully contained in the box. • Drag the box in any other direction (bottom-left, top-right, bottom-right) to select any object that is at least partially within the box.
Translate objects.	Select an object or objects and press **T** to show the translation tool.

Task	Action
Rotate objects.	Select an object or objects and press **R** to show the rotation tool.

Camera Control (Editing Canvas)

For more details on controlling the camera in the canvas, see "Camera Control in RoadRunner" on page 1-44.

Task	Action
Rotate the camera around a point of interest.	Hold **Alt**, then click and drag. **Note** In Linux, Ubuntu 16.04, pressing the **Alt** key moves the current window and pressing the **Windows** key shows certain help overlays. To use the **Windows** key instead of the **Alt** key for moving windows, update Ubuntu 16.04 according to the instructions in "Update Linux Ubuntu Key Mapping" on page 2-34.
Zoom in or out.	Hold **Alt**, then right-click and drag up or right to zoom in, or down or left to zoom out. Alternatively, move the mouse scroll wheel up or down.
Pan across the scene in the *x*-direction or *y*-direction.	Hold **Alt**, left-click, and right-click, then drag. Alternatively, hold the middle-mouse button, then drag.
Raise and lower the camera in the *z*-direction.	Hold **Alt+Shift**, left-click, and right-click, then drag.
Frame the camera around a selected object.	Select an object and press **F**.
Switch to perspective view, where distant objects appear smaller than close objects.	Press **P**.
Switch to orthographic view, where objects do not change size as they get closer or farther away.	Press **O**.
Set the camera view to point top down.	Press **5** on the number pad.
Set the camera view to point north.	Press **8** on the number pad.
Set the camera view to point south.	Press **2** on the number pad.
Set the camera view to point west.	Press **4** on the number pad.
Set the camera view to point east.	Press **6** on the number pad.

Camera Control (2D Editor)

Task	Action
Zoom in or out.	Hold **Alt**, then right-click and drag up or right to zoom in, or down or left to zoom out. Alternatively, move the mouse scroll wheel up or down.
Pan across the scene in the *x*-direction or *y*-direction.	Hold **Alt**, then click and drag. Alternatively, hold the middle-mouse button, then drag.
Frame the camera around an object of interest.	Select an object and press **F**.

Scene Views

Use these keyboard shortcuts to show or hide various scene aspects on the canvas.

Task	Action
View shaded wireframe.	**F3**
View aerial imagery.	**F4**
View elevation map.	**F5**
View point cloud.	**F6**
View vector data.	**F7**
View scene.	**F8**
View ASAM OpenDRIVE data.	**F9**
View RoadRunner HD Map.	**F10**
View SD Map.	**F11**

Scenarios (Requires RoadRunner Scenario)

Task	Action
Switch to scene editing.	**Shift+1**
Switch to scenario editing.	**Shift+2**
Run or restart simulation.	**Ctrl+R**

Utilities

Task	Action
Take screenshot.	**Ctrl+P**

File Operations

If you are in scene editing mode (**Shift+1**), then these operations apply to scenes. If you are in scenario editing (**Shift+2**), then these operation apply to scenarios. Use of scenario editing mode and switching between modes requires RoadRunner Scenario.

Task	Action
Create new scene or scenario.	**Ctrl+N**
Open scene or scenario.	**Ctrl+O**
Save scene or scenario.	**Ctrl+S**
Save scene or scenario as.	**Ctrl+Shift+S**
Exit RoadRunner.	**Alt+F4**

Update Linux Ubuntu Key Mapping

In Linux, Ubuntu 16.04, pressing the **Alt** key moves the current window, and pressing the **Windows** key shows certain help overlays. It is recommended that you update Ubuntu to use the **Windows** key (instead of the **Alt** key) for moving windows. To make this update, follow these steps:

1 Install `dconf-editor` if it is not already installed. At the command line, enter this code:

   ```
   sudo apt-get install dconf-editor
   ```
2 Open `dconf-editor`.
3 Navigate to **org** > **gnome** > **desktop** > **wm** > **preferences**.
4 Change the `mouse-button-modifier` to `<Super>`.

Note It is important to assign a valid key to `mouse-button-modifier`. Leaving that option blank prevents the mouse from interacting with any windows.

See Also

More About
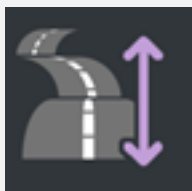
- "Create Simple RoadRunner Scene" on page 1-19
- "Camera Control in RoadRunner" on page 1-44
- "Manipulate Scene Objects" on page 2-14

Choose a RoadRunner Tool

The RoadRunner toolbar contains a variety of tools for editing scenes. The tool that you select determines the objects that you can select and edit in the scene editing canvas. In addition, certain actions automatically change the active tool. For example, dragging certain types of assets from the **Library Browser** into the scene adds that asset to the scene and automatically switches RoadRunner to an appropriate tool for that asset.

Road Tools

Road tools are used to create and modify roads and their attributes.
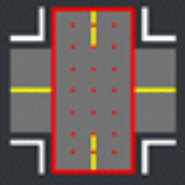
Tool	Description
Road Plan Tool	Create and lay out roads.
Road Circle Tool	Build closed circular loop road, such as for creating roundabouts.
Cross Section Tool	Manipulate banking, crowning, and curb shapes at road cross-sections.
Road Height Tool	Manipulate vertical profile of roads.

Tool	Description
Road Superelevation Tool	Adjust superelevation (slope and banking angle) for full width of road.
Road Chop Tool	Chop single road into two connected roads.
Road Construction Tool	Specify physical construction of road sections.
Road Speed Limits Tool	Set speed limits along road sections.
Slip Road Tool	Create onramps, offramps, and road splits.
Road Offset Tool	Adjust connection between two end-to-end roads.

Tool	Description
Road Anchor Tool	Define road anchors to place scenarios within scenes (requires RoadRunner Scenario)

Junction Tools

Junction tools are used to modify junction (intersection) geometry and lane connectivity. In most cases, junctions are initially created using the **Road Plan Tool** or **Slip Road Tool**. In advanced situations, the **Custom Junction Tool** provides more explicit control over junction creation.

Tool	Description
Custom Junction Tool	Override RoadRunner automatic junction functionality for advanced cases.
Corner Tool	Adjust shape and materials of junction corners.
Junction Surface Tool	Control how road elevations and cross-sections influence interior triangulation of intersections.
Maneuver Tool	Manipulate individual maneuver roads (paths) within junction.

Tool	Description
Signal Tool	Configure junction signalization and signal traffic phases.

Lane Tools

Lane tools are used to create and edit lanes and their properties.

Tool	Description
Lane Tool	Delete lanes and update lane type and travel direction.
Lane Width Tool	Adjust lane widths along road.
Lane Offset Tool	Adjust location of center lane of road.
Sidewalk Height Tool	Modify sidewalk and curb heights.

Tool	Description
Lane Add Tool	Add fully formed lanes along road.
Lane Form Tool	Add forming or ending lane along road.
Lane Carve Tool	Create tapering cut in lanes, such as for creating dedicated turn lane in median.
Lane Chop Tool	Cut single lane into two lanes.
Lane Split Tool	Split lane lengthwise into two lanes.

Marking Tools

Marking tools are used to create and modify road paint, lane markings, and decals in the scene.

Tool	Description
Lane Marking Tool	Add linear markings to lane boundaries.

Tool	Description
Marking Point Tool	Place point markings (stencils), such as arrows and words, on road surfaces.
Marking Curve Tool	Place straight or curved markings at arbitrary locations.
Marking Polygon Tool	Define areas of asphalt patches or repeated marking stripes on roads and terrain surfaces.
Traffic Island Tool	Create freeform traffic islands
Parking Tool	Define parking spaces and other parking-related markings.
Crosswalk And Stop Line Tool	Add crosswalks and stop lines between corners at intersections.

Prop Tools

Prop tools are used to create and modify 3D props and road furniture, such as trees, signs, and guardrails.

RoadRunner can import props from a variety of file formats and uses a rich set of tools to place props in the scene. These tools are used to place a variety of different asset types, including **Prop Model Assets**, **Prop Set Assets**, **Sign Assets**, and **Extrusion Assets**.

Tool	Description
Prop Point Tool	Place individual props in a scene and connect them to other props.
Prop Curve Tool	Place props and extrusions along free-form curves.
Prop Polygon Tool	Place props within arbitrarily shaped regions.
Prop Span Tool	Place props and extrusions along road.
Sign Tool	Modify custom signs, such as street name signs and freeway billboards.
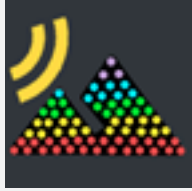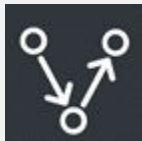
Terrain Tools
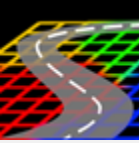
Terrain tools are used to create and modify the surfaces around and between roads. For more details on surfaces, see "How Surfaces Work in RoadRunner" on page 4-4.

Tool	Description
Surface Tool	Model surfaces around roads, such as walkways, driveways, parking lots, and natural terrain.

GIS Tools
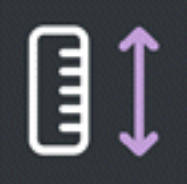
GIS tools are used to import and adjust a variety of common geographic information system (GIS) files.

Tool	Description
Aerial Imagery Tool	Manage import and configuration of aerial imagery files.
Elevation Map Tool	Manage import and configuration of digital elevation model (DEM) files.
Point Cloud Tool	Manage import and configuration of lidar point cloud files.
Vector Data Tool	Manage import and configuration of vector data files and explore shape attributes.

Tool	Description
OpenDRIVE Viewer Tool	Visualize ASAM OpenDRIVE data for import.
Road CRG Tool	Manage import and configuration of ASAM OpenCRG® files.
Scene Builder Tool	Generate 3D road models from HD Map data.
World Settings Tool	Configure geographic position and size of environment model for data import and export.
SD Map Viewer Tool	Generate 3D road models from Zenrin Japan Map API 3.0 (Itsumo NAVI API 3.0) data.

Utility Tools

These assorted utility tools enable you to measure distances, capture screenshots, and preview export data.

Tool	Description
Measurement Tool	Measure positions, distances, and angles in scene.
Screenshot Tool	Generate and save image of current camera view.
OpenDRIVE Export Preview Tool	Visualize and validate ASAM OpenDRIVE export of scene and load external ASAM OpenDRIVE files.
Scene Export Preview Tool	Preview scene geometry to be exported.

See Also

Related Examples

- "Point Editing" on page 2-65
- "Curve Editing" on page 2-66
- "Polygon Editing" on page 2-68
- "Tangent Editing" on page 2-70
- "Span Editing" on page 2-75
- "Region Graph Editing" on page 2-78

RoadRunner Asset Types
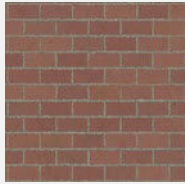
Assets are 3D models, textures, GIS files, and other data that are shared by multiple RoadRunner scenes. You can view and modify assets from the **Library Browser**. For information about how assets are stored in a project, see "RoadRunner Project and Scene System" on page 2-2.

RoadRunner also supports a variety of file formats for developing your own assets. For more details, see "Create, Import, and Modify Assets" on page 2-50.

RoadRunner comes installed with a small library of various assets to get you started. With a "RoadRunner Asset Library Add-On" license, you can install hundreds of additional assets to use in your scenes.
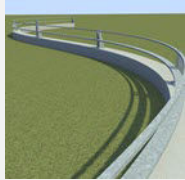
Texture and Material Assets
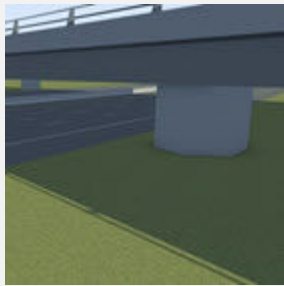
Material assets are used to define the visual properties of surfaces, sidewalks, lanes, and other objects. Texture assets are image files, typically used as texture channels for material assets.

Asset	Description
Texture Assets	Define texture channels for material assets.
Material Assets	Define visual properties of surfaces, sidewalks, lanes, and other objects.

Prop Assets

Prop assets define 3D objects that you can place within a scene.

Asset	Description
Extrusion Assets	Define extruded geometry for features such as walls, guard rails, and fences.

Asset	Description
Post Assets 	Define building support posts, such as for bridges and overpasses.
Prop Model Assets 	Define external 3D model files to add to scene.
Prop Assembly Assets 	Define collection of prop instances stored as single asset.
Prop Set Assets 	Define collections of props that have a random distribution.
Sign Assets 	Define standard and custom street signs.

Asset	Description
Signal Assets 	Define dynamic traffic signal heads with lights.

Marking Assets

Marking assets define markings found on roads, such as crosswalks, lanes, and road stencils such as arrows, text, and symbols.

Asset	Description
Crosswalk Marking Assets 	Define crosswalk markings, such as color, width, and spacing.
Lane Marking Assets 	Define lane markings, such as color, width, and dash spacing.
Polygon Marking Assets 	Define space-filling road markings, such as crosshatch and chevron markings.

Asset	Description
Stencil Marking Assets	Define road paint features, such as arrows, text, and symbols.
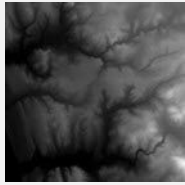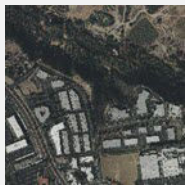
Road Assets

Road style assets are templates that specify the properties of new roads.

Asset	Description
Road Style Assets	Define templates that specify properties of new roads.

GIS Assets

Geographic information system (GIS) assets carry georeferencing information, which you can use to position these assets on the Earth. For information on finding GIS data resources that are compatible with RoadRunner, see "Download GIS Data for Use in RoadRunner" on page 3-8.

Asset	Description
Elevation Map Assets	Add GIS raster elevation data to scene.
Aerial Image Assets	Add GIS satellite and aerial imagery to scene for visual reference.

Asset	Description
Vector Data Assets 	Add GIS shapefiles and other vector data to scene for visual reference.
Point Cloud Assets 	Add aerial or vehicular point clouds to scene for visual reference.

See Also

Related Examples

- "Create, Import, and Modify Assets" on page 2-50
- "RoadRunner Project and Scene System" on page 2-2
- "RoadRunner Asset Library Add-On"

Create, Import, and Modify Assets

Assets are the materials, textures, props, and other 3D objects that are available in a RoadRunner project and that can be added to a scene. To browse, create, and modify assets in a current project, use the **Library Browser**. The **Library Browser** is divided into two panes:

- The left pane displays the directory structure within the `Assets` folder, enabling you to quickly navigate the folder hierarchy.
- The right pane displays the contents of the currently selected folder. When you select an asset from this pane, you can view its attributes from the **Attributes** pane and preview it in the asset viewer.
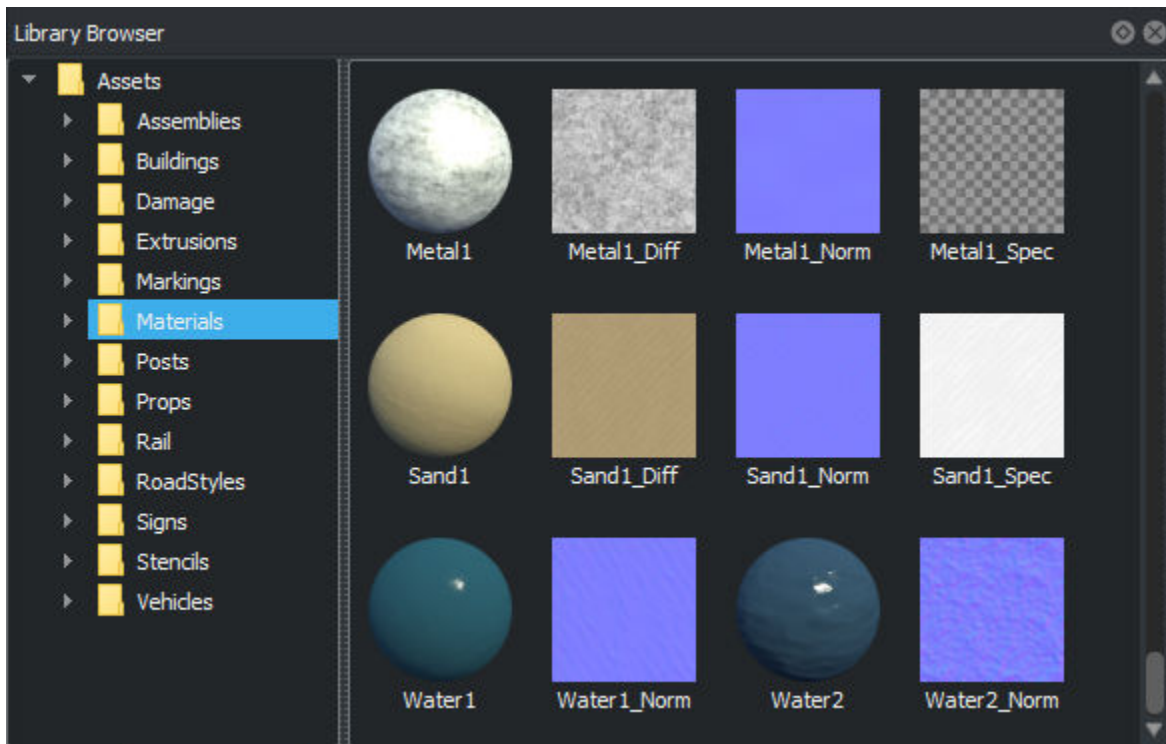
Using these two panes, you can create and modify assets, manage asset files in your project, and add assets to scenes.

Only files recognized by RoadRunner as assets are displayed. Other system files and auxiliary files in the directory are not shown. For more details on the asset files contained in a project, see "RoadRunner Project and Scene System" on page 2-2.

Create and Import Assets

Depending on the asset type, you can either create new assets for a project either directly in the **Library Browser** or you can import files created outside RoadRunner into the **Library Browser** to create RoadRunner assets.

Create Asset Within RoadRunner

You can create some assets directly within the **Library Browser**, such as materials and road styles. Follow these steps:

1 Navigate to the folder in the **Library Browser** where you want to create the new asset.

2 Right-click in the **Library Browser** and select **New**, then select **(Asset Type)**. Alternatively, select **Assets**, and then select the **(Asset Type)** menu option.

3 Specify a name and press **Enter**.

You can also:

1 Right-click the asset in the **Library Browser** and select **Duplicate**.

2 Specify a name for the new asset and press **Enter**.

You can use these steps only for assets created within RoadRunner, such as a material or road style asset. If you want to duplicate an asset that depends on another file, such as a texture image or 3D model, duplicate the dependent file only (not the `rrmeta` file) by using the file explorer for your operating system.

Create Asset by Importing File Created Outside RoadRunner

Some assets depend on files created outside of RoadRunner, such as a texture image saved as a PNG file or a 3D building model saved as an STL file, or a 3D tree saved as an FBX file. To create such assets for use in RoadRunner, drag the dependent file into the **Library Browser**. Follow these steps:

1 Navigate to the folder in the **Library Browser** where you want to add the new asset.

2 In the file explorer window for your operating system, navigate to the location of the dependent file (for example, the texture image or the 3D model).

3 Select the file (and any associated files or folders) in the file explorer.

4 Drag the file, and any associated files or folders, into the **Library Browser**.
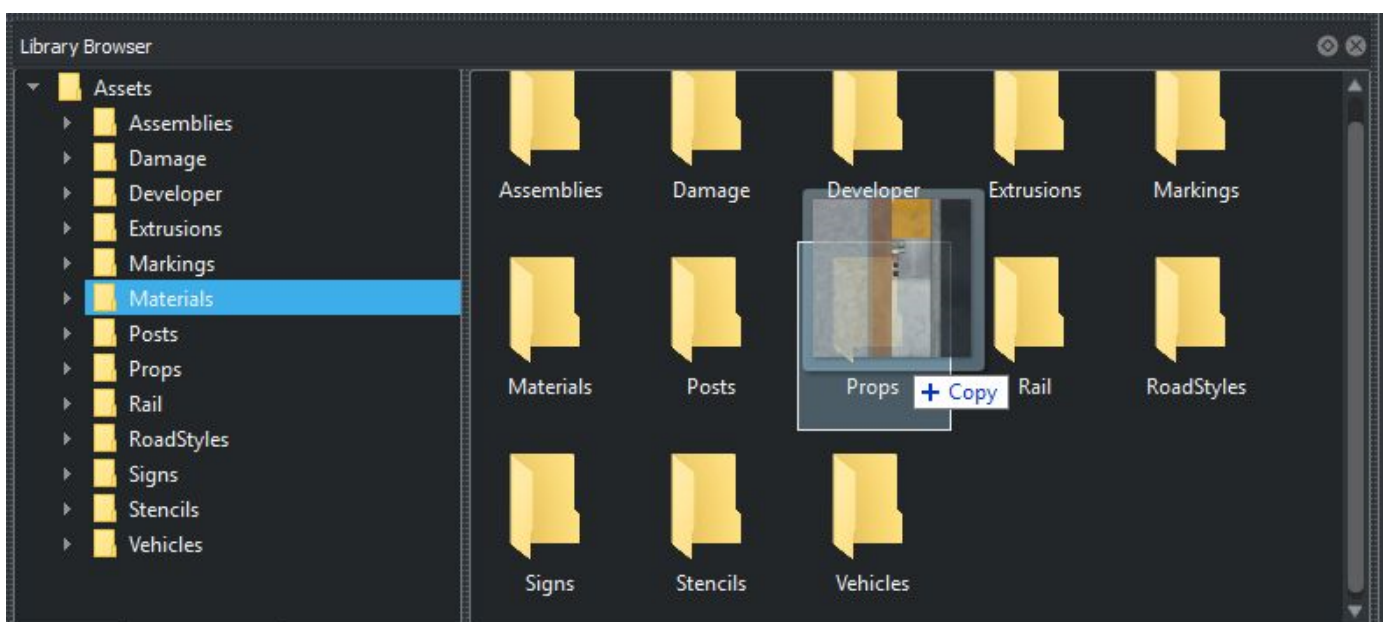
This operation copies, rather than moves, the selected files into the directory of the current project.

Alternatively, you can perform these steps by using the file explorer window by moving the files somewhere under the `Assets` directory of your project. This option can be useful if you want to move rather than copy the files, or if you want to use an external script to create assets in a project.

Modify Assets

The steps to modify an asset differ depending on the specific type of asset. For more details, refer to the documentation for the specific asset type.

In most cases, you can modify an asset by following these steps:

1   Select the asset in the **Library Browser**.

2   View and modify the asset attributes displayed in the **Attributes** pane.

Some assets, such as **Sign Assets**, are modified using the **2D Editor** pane.

Modifications made to an asset are saved only when you next save the project.

Reload Modified Assets

If you change an asset by using an external application, such as modifying a texture file using an image editor, you can force RoadRunner to reload the asset. Right-click in the **Library Browser** and select **Update Assets**. Alternatively, select **Assets** and then the **Update Assets** menu option.

Manage Assets

You can use the **Library Browser** to manage the assets in your project.

Rename Asset

1   Right-click the asset in the **Library Browser** and select **Rename** (or press **F2**).

2   Specify a new name and press **Enter**.

Move Assets or Folders

1   Select assets or folders in the **Library Browser**.

2   Click and drag the assets or folders to a different folder in either the left or right pane.

Moving an asset automatically updates asset references in the current scene, but other saved scenes might still reference the old asset location. See "Find Moved Assets" on page 2-53.

Create New Asset Folder

1   Right-click in the **Library Browser** and select **New**, then **Folder**. Alternatively, select **Assets**, then **New**, then **Folder**.

2   Specify a name and press **Enter**.

Delete Asset or Asset Folder

1   Select assets or folders in the **Library Browser**.

2 Right-click and select **Delete**. Alternatively, select the **Edit** , then **Delete** menu option, or press **Delete**.

Find Moved Assets

If an asset or asset folder has been moved or renamed, then existing scene files might still refer to the old location. If a scene cannot find an asset, RoadRunner replaces references to that asset with a visually distinct fallback asset (for example, props display as pink barrels, and textures display as striped red and blue images).
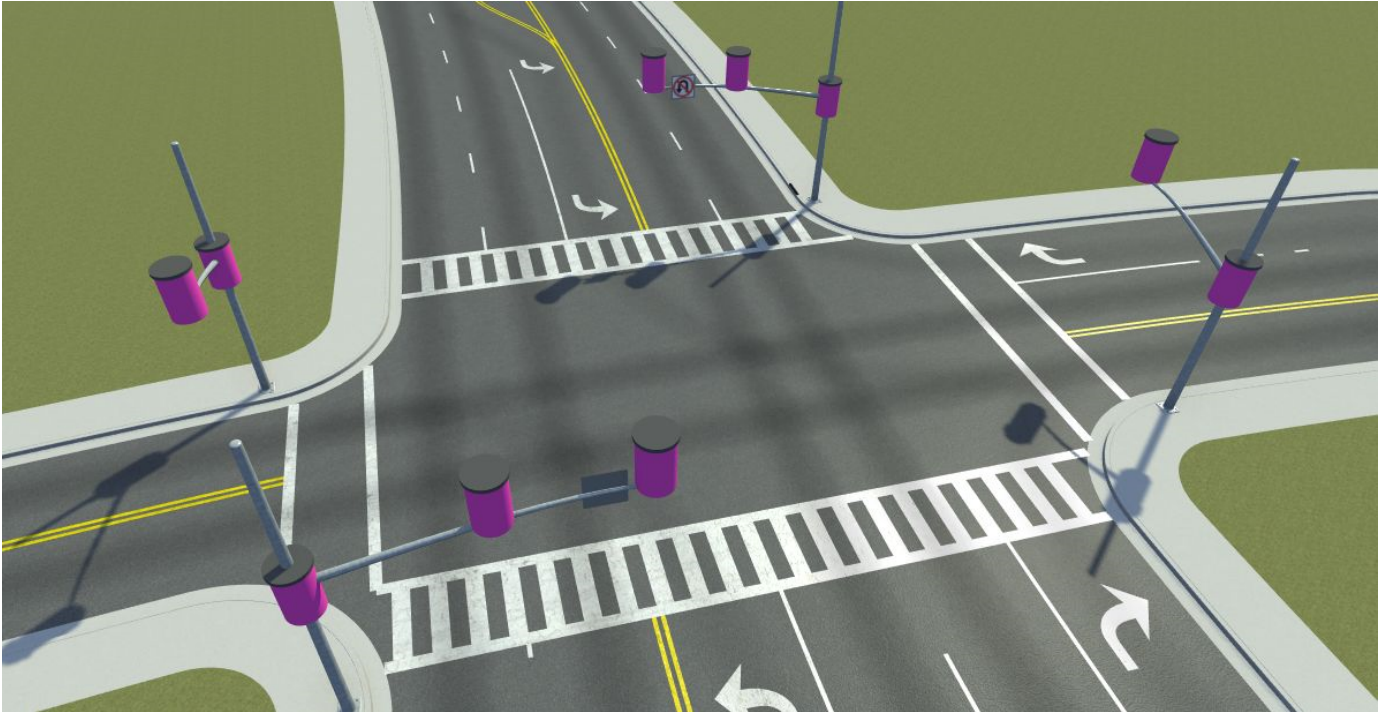
If you encounter this situation, RoadRunner can search for the missing references and attempt to relink them. Follow these steps:

1 Open the scene file containing the missing asset references.

2 Right-click in the **Library Browser** and select **Update Assets**. Alternatively, select the **Assets** , then **Update Assets** menu option.

This search finds moved assets only if the corresponding `rrmeta` file was also moved or renamed and was left intact.

Select Assets

To select all the assets of a given type in a scene, follow these steps:

1 In the **Library Browser**, navigate to the folder that contains the asset.

2 Right-click the asset file and select **Select in Scene**.

RoadRunner selects all assets of that type in the scene and switches to the primary tool used to select that asset. If you are already using a tool that can select the asset, then RoadRunner does not switch tools.

Note Selecting road style assets in a scene is not supported.

Visualize Assets

Using the asset viewer, you can visualize the currently selected asset in the **Library Browser**. By default, the asset viewer appears below the **Attributes** pane when a single asset is selected in the **Library Browser**. This image shows a sample material asset.
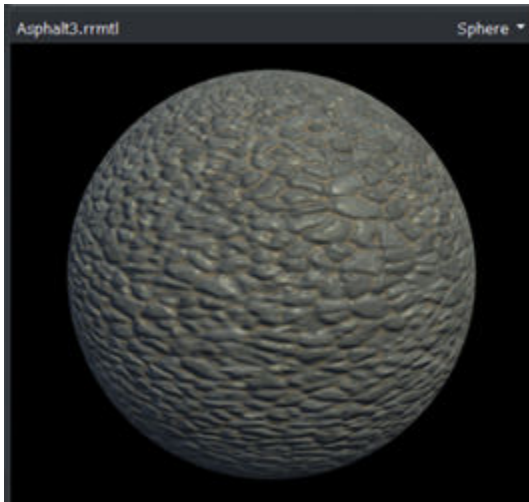
The asset viewer displays different asset types, such as 3D model assets and 2D image assets, in different ways. If the selected asset type supports a 3D display, you can move the camera by using the same controls listed in "Camera Control in RoadRunner" on page 1-44. Unlike the other render windows, you do not need to hold **Alt** to adjust the camera in the asset viewer.

Change Asset Display Type

Some types of assets support additional viewing options. For example, **Material Assets** can be displayed on different types of geometry, and **Prop Model Assets** can be displayed as a point, curve, or collection.

To change the asset display type, in the top-right corner of the asset viewer, click the current display type and select the new display type you want.
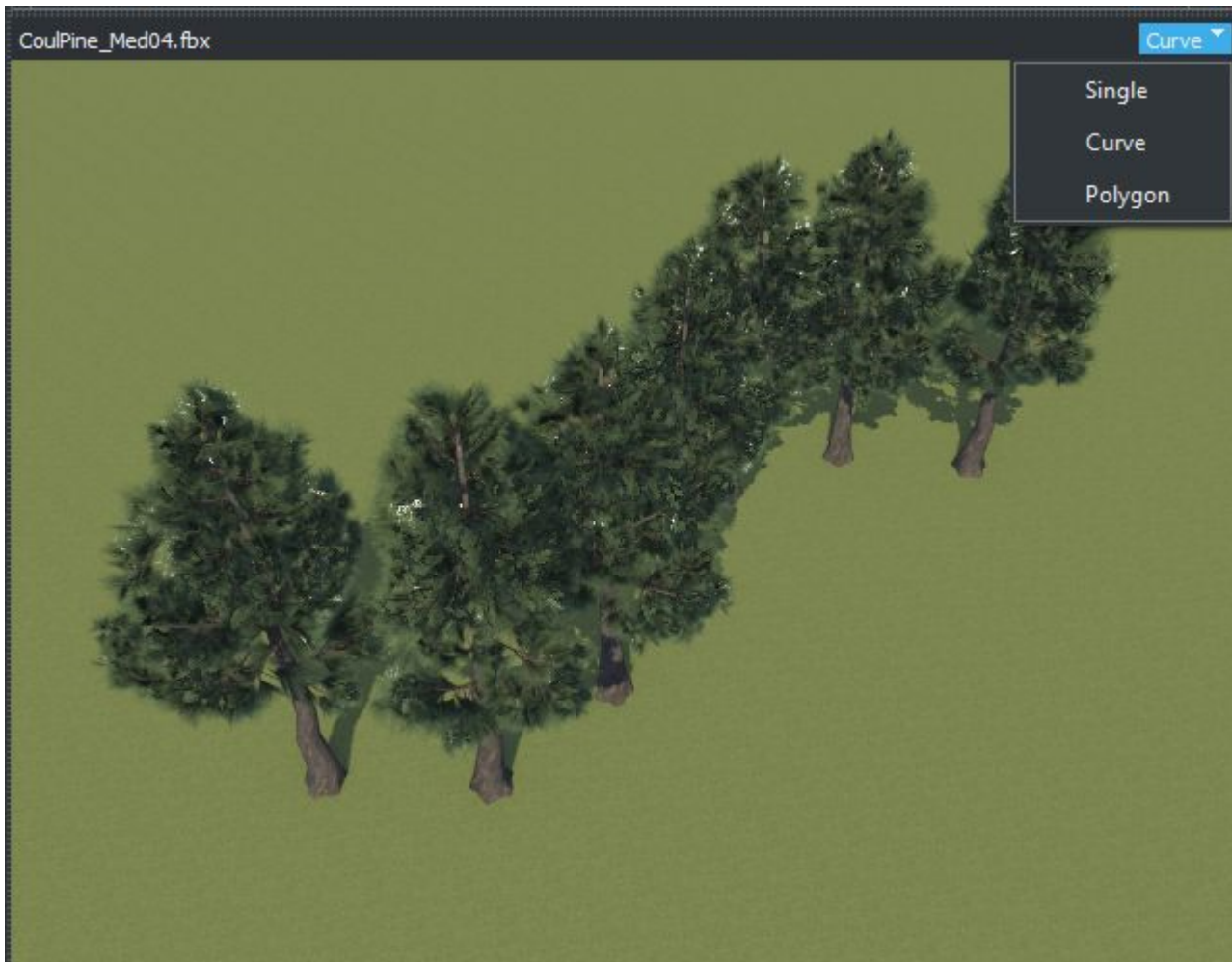
Upgrade RoadRunner Asset Library

You can upgrade the assets of a project created with a previous version of RoadRunner to the current version. You do not need to have a RoadRunner Asset Library license to upgrade the assets. Upgrading the assets:

- Removes the replaced assets from the old location.

- Keeps customer assets (assets that are not in the Asset Library) intact.

Note The file locations of all built-in assets are reset to their default locations. Make a copy of your project before upgrading if you have changed the file locations of any built-in assets.

To upgrade the RoadRunner Asset Library, follow these steps:

1   Click **Upgrade Asset Library** from the **Assets** menu.
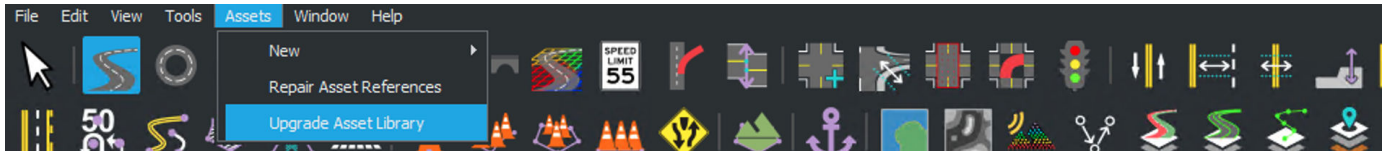
2 Select **Also upgrade assets in existing Scenes and Scenarios** if you want to upgrade the assets in the existing scene or scenario file. Then, click **Upgrade** in the Upgrade Asset Library dialog box.
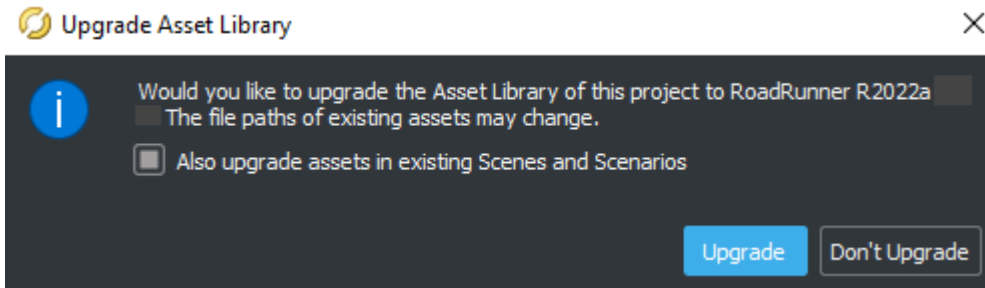
3 Once you click **Upgrade**, you can see the progress of the up-gradation process.
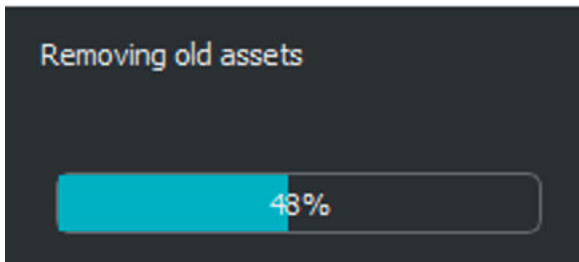
4 Once the upgrade process completes, you see a message in the dialog box, which denotes the successful completion of the upgrade. It also shows the number of assets added, the number of assets updated, and the amount of memory used by the assets. The upgrade process does not fail if no new assets are detected.
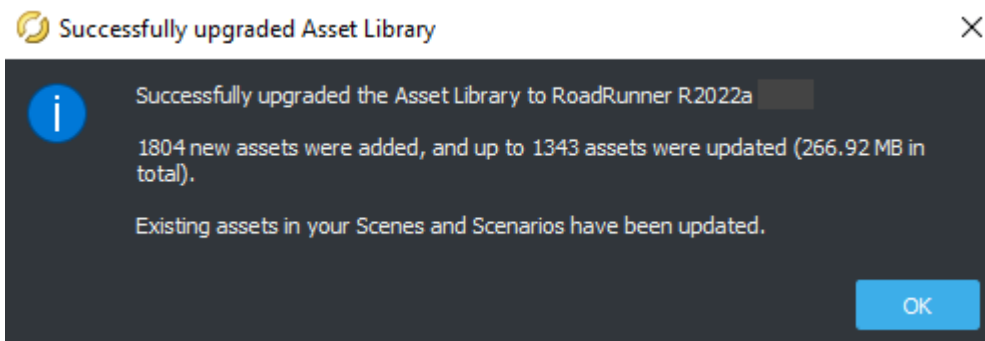
5 If you want to keep the assets in your scenes and scenarios intact, uncheck the **Also upgrade assets in existing Scenes and Scenarios** in the Upgrade Asset Library dialog box. The assets found in your scenes and scenarios that would otherwise be replaced are automatically moved to a new folder.
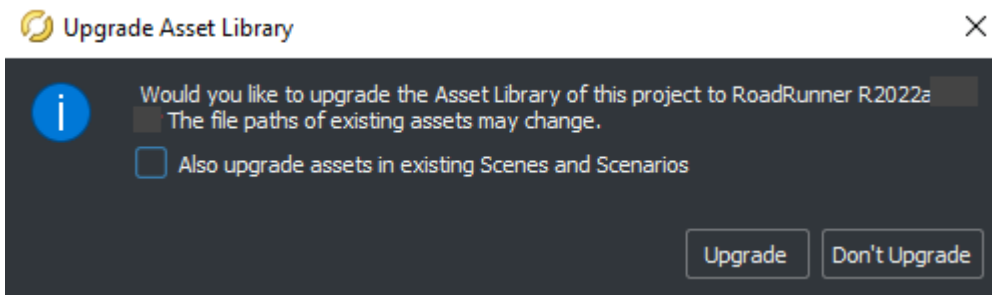
6 Once the upgrade process completes, you see a message in the dialog box, which denotes the successful completion of the upgrade. It also shows the number of assets added, the number of assets updated, and the amount of memory used by the assets. It also shows the number of existing assets that are moved to a new folder **Pre_R2022a_Assets'**
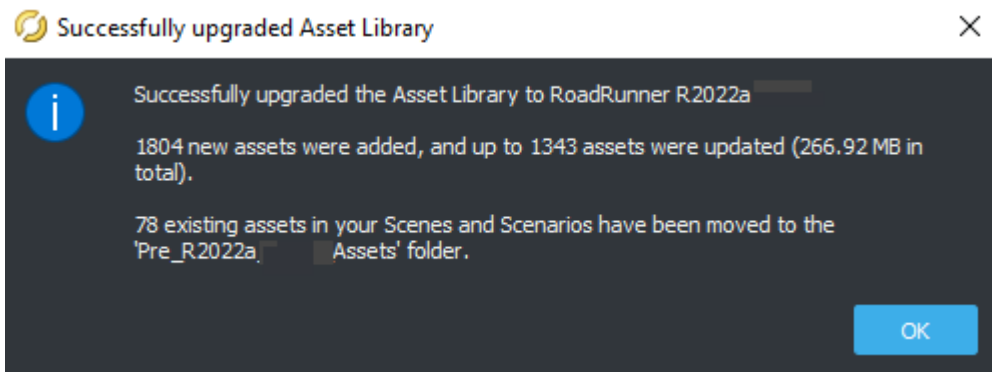
See Also

Related Examples

- "RoadRunner Asset Types" on page 2-45
- "RoadRunner Project and Scene System" on page 2-2
- "Manipulate Scene Objects" on page 2-14

Create, Import, and Modify Scene Assets

Scene Assets are collections of roads, intersections, props, and other 3D objects that are available in a RoadRunner project and can be added to a scene to quickly build a complete environment. To browse, create, and modify scene assets in a current project, use the **Library Browser**. The **Library Browser** is divided into two panes:

- The left pane displays the directory structure within the `Assets` folder, enabling you to quickly navigate the folder hierarchy.
- The right pane displays the contents of the currently selected folder. When you select an asset from this pane, you can view its attributes from the RoadRunner **Attributes** pane and preview it in the asset viewer.

Create Template Asset of Entire Scene

1 Open an existing scene or create a new scene in RoadRunner. Do not select any objects within the main scene window.

2 Create a new folder named `Templates` in the `Assets` folder. If a `Templates` folder already exists, you can select that folder instead.

3 From the **Assets** menu, select **New > Scene Template** to automatically add the current scene as a template asset into the folder and the RoadRunner Asset Library of the current project.

4 Rename the scene with an appropriate name.
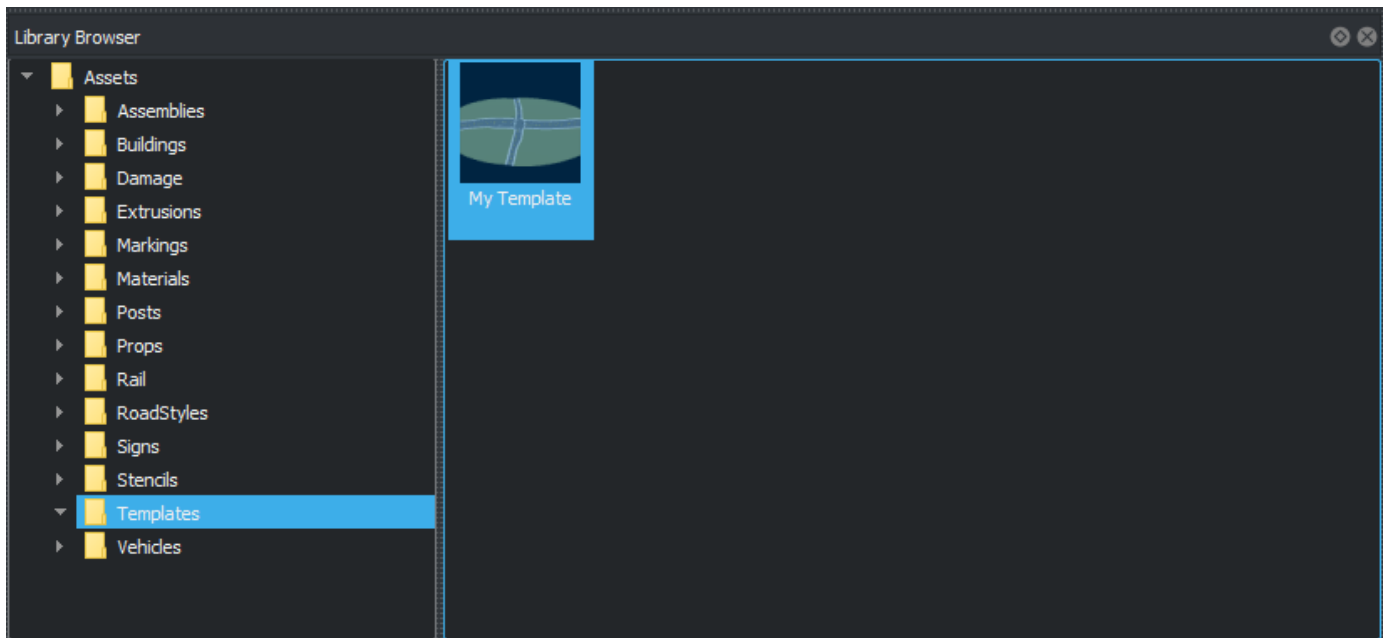
Using this method, you can create a variety of templates from existing scenes.

Create Template Asset from Selection

1 Open an existing scene or create a new scene in RoadRunner.

2   Using the **Selection** tool, select a portion of the 3D scene from which to create a scene template.
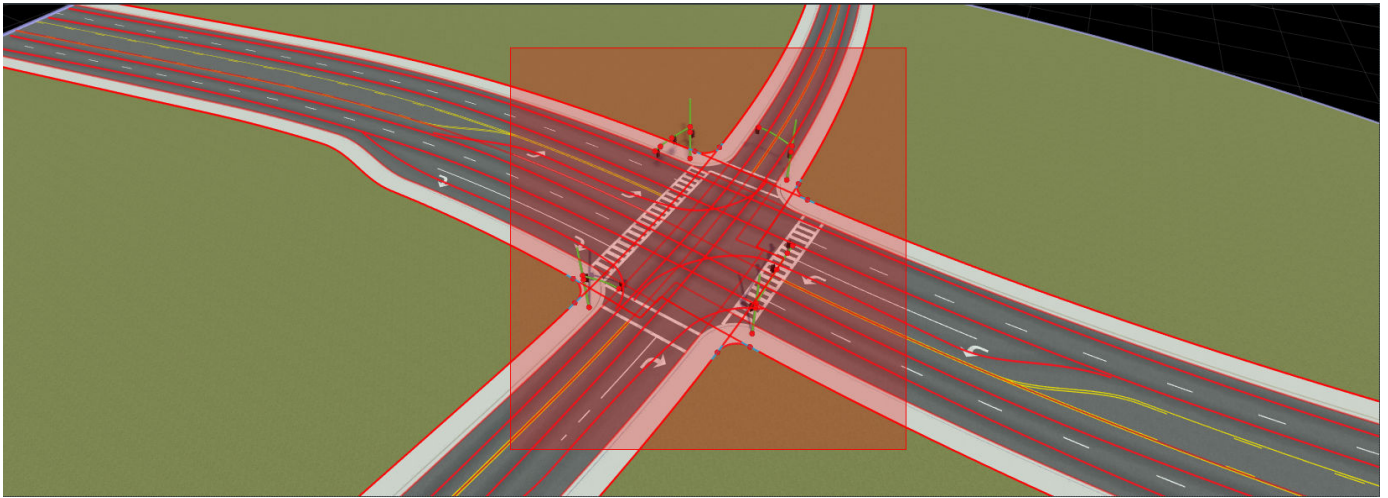
3   Create a new folder named `Templates` in the `Assets` folder. If a `Templates` folder already exists, you can select that folder instead.

4   From the **Assets** menu, select **New > Scene Template** to automatically add the current scene as a template asset into the folder and in the RoadRunner Asset Library in the current project.

5   Name the scene template.
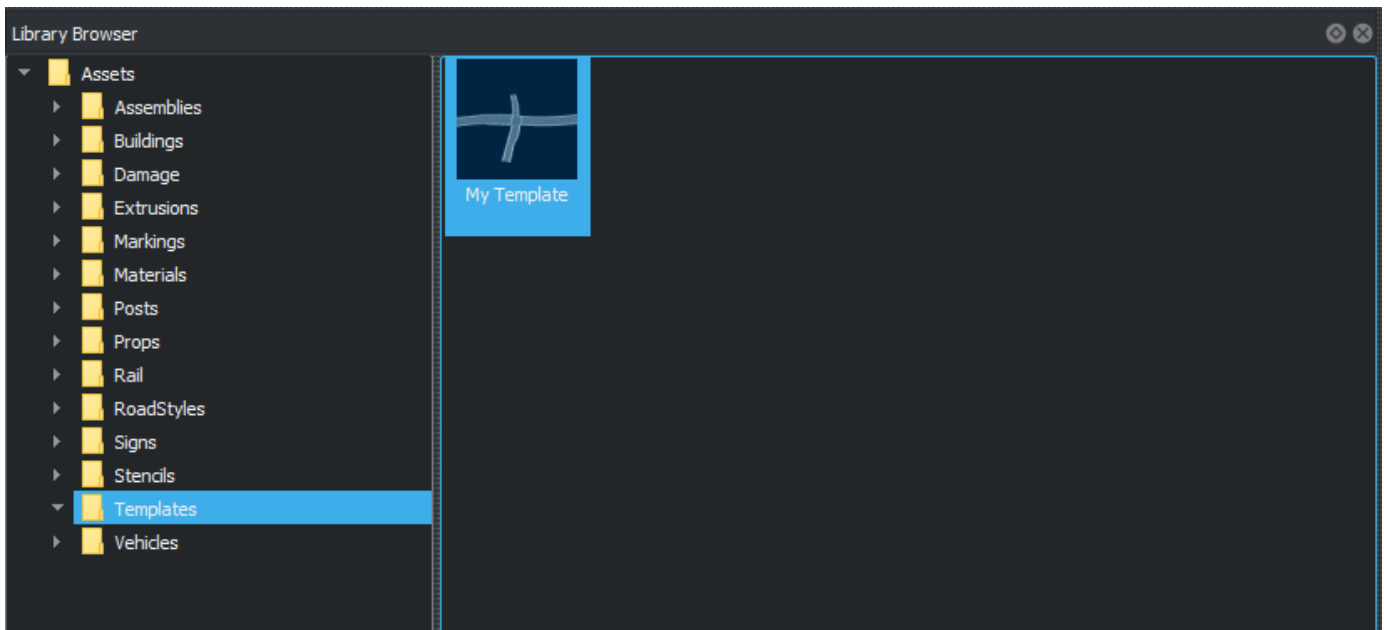
Add Template Asset to a Scene by Dragging

1   Point to the new template in the **Library Browser**. Click and hold to grab the template.

2   Drag the selected scene from the **Library Browser** into the scene and position it in the desired location. The template, while selected, is visible in the 3D scene and follows the cursor.

3   Release the mouse button to add the template into the scene. The new object is automatically selected by the **Selection Tool**.

4   You can reposition the template by dragging it to a new location, or use undo to remove the template and repeat the process of adding it to the scene in a different location.

Add Template Asset to Scene Using Copy Paste

1   Open an existing scene or create a new scene in RoadRunner.

2   Select a portion of the 3D scene from which to create a scene template.

3   Press **Ctrl+C** or select **Edit > Copy**. If a scene template can be created from your selection, RoadRunner stores the selection in the active clipboard for pasting.

4   Press **Ctrl+V** or select **Edit > Paste** to paste the template in your scene. The objects in the selection paste to the exact locations from which they are copied.

5   To move the objects to a new location, press **T** or select **View > Translate** to enable the translate manipulator.

6   Click and drag one of the translation axes to move the pasted objects to a new location.

See Also

Resolve Geometry Issues

You can enable visualization of geometric issues related to triangulation. To modify geometry issue settings, use the Geometry Issue Settings dialog box under the **Tools** menu.
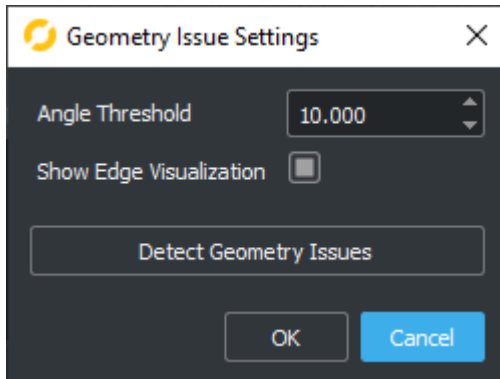
Angle Threshold

The **Angle Threshold** parameter controls the angle threshold (in degrees) at which geometric issues are displayed. This angle directly translates into the angle between adjacent faces of the road mesh. These images show the same road section with different **Angle Threshold** values used for displaying geometric issues. As the **Angle Threshold** value increases, the number of issues that are detected decreases.

Angle Threshold: 1 degree
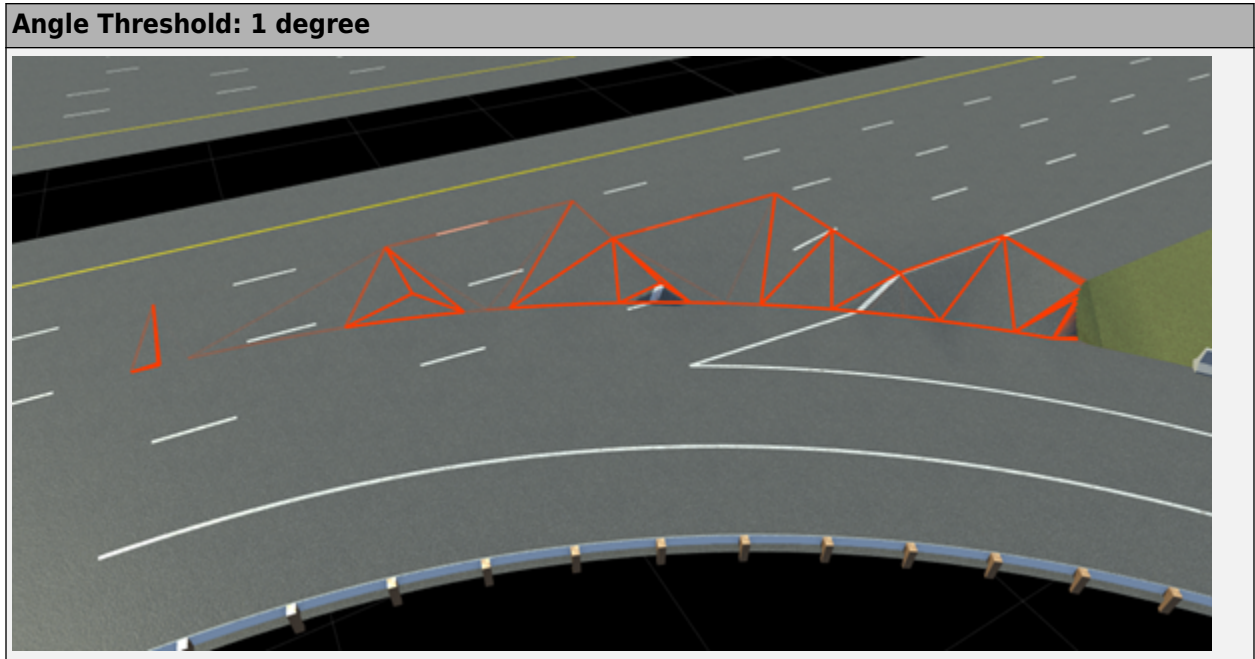
Angle Threshold: 10 degrees
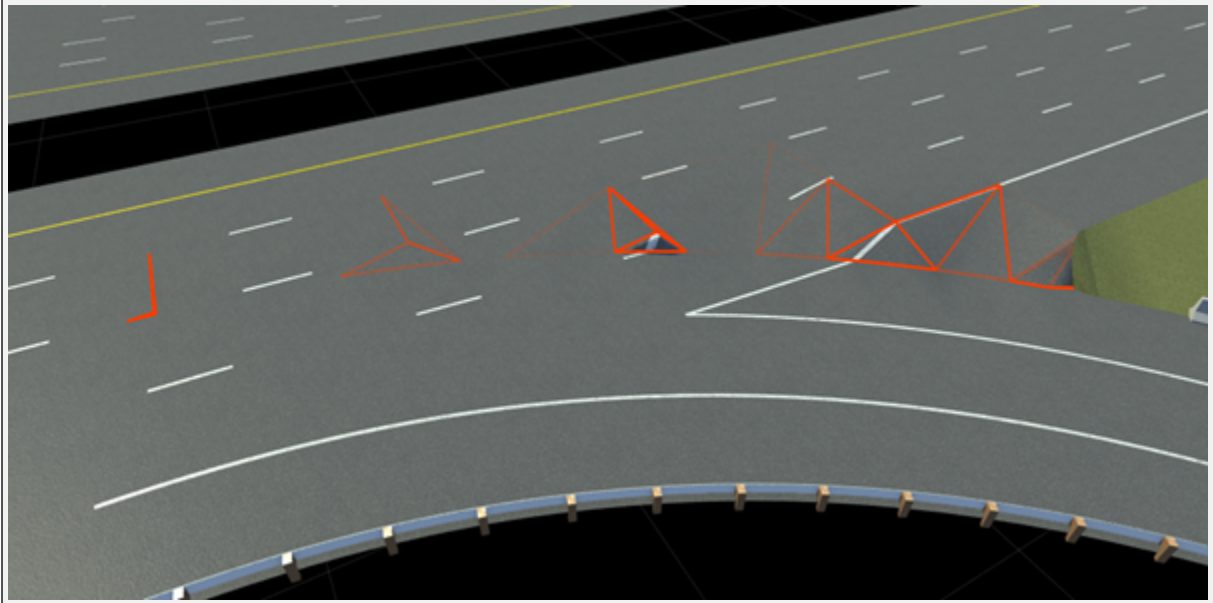
Angle Threshold: 30 degrees
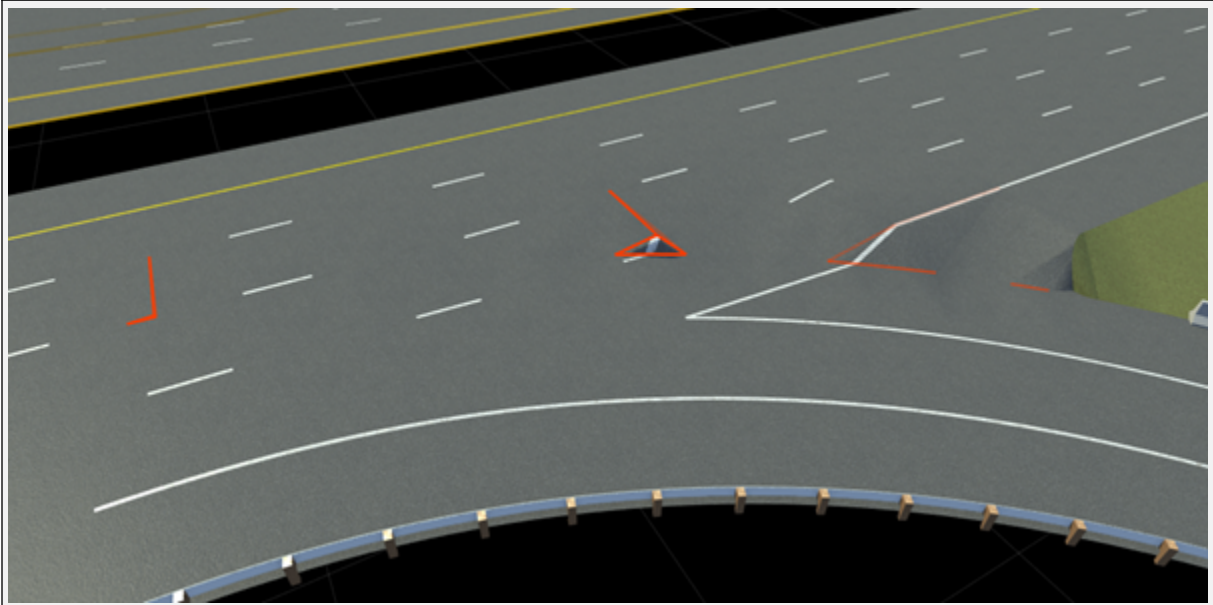
Show Edge Visualization

The **Show Edge Visualization** parameter enables visualization of the geometric issues. This toggle is applied when the dialog box is closed and corresponds directly to the **View > Geometric Issues** menu option.

Edge visualization disabled
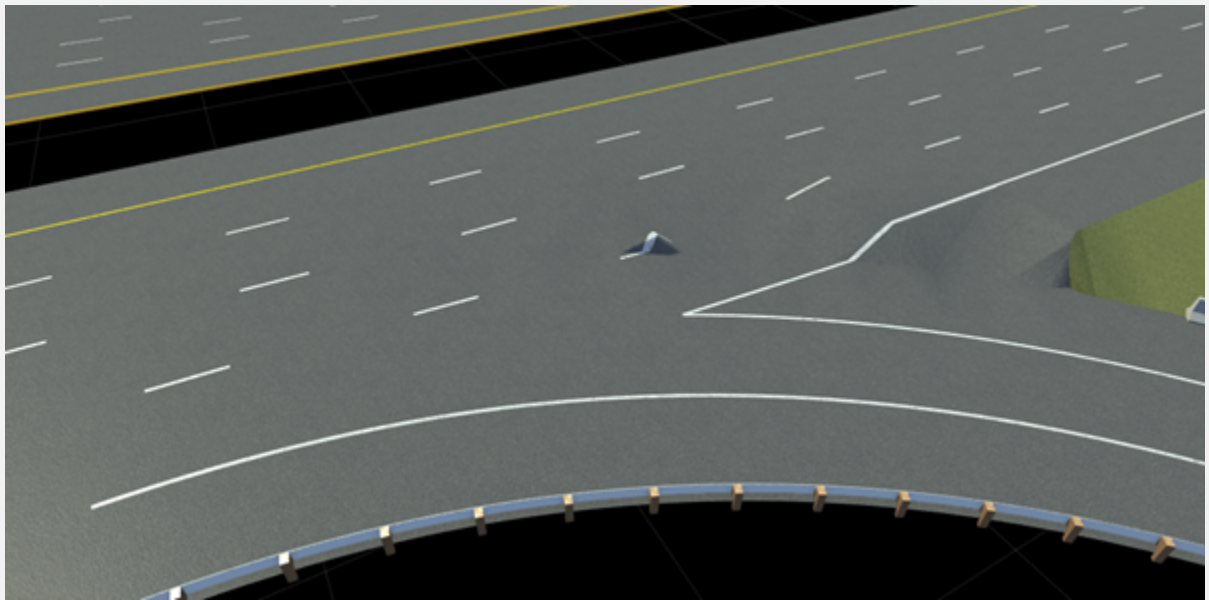
Edge visualization enabled
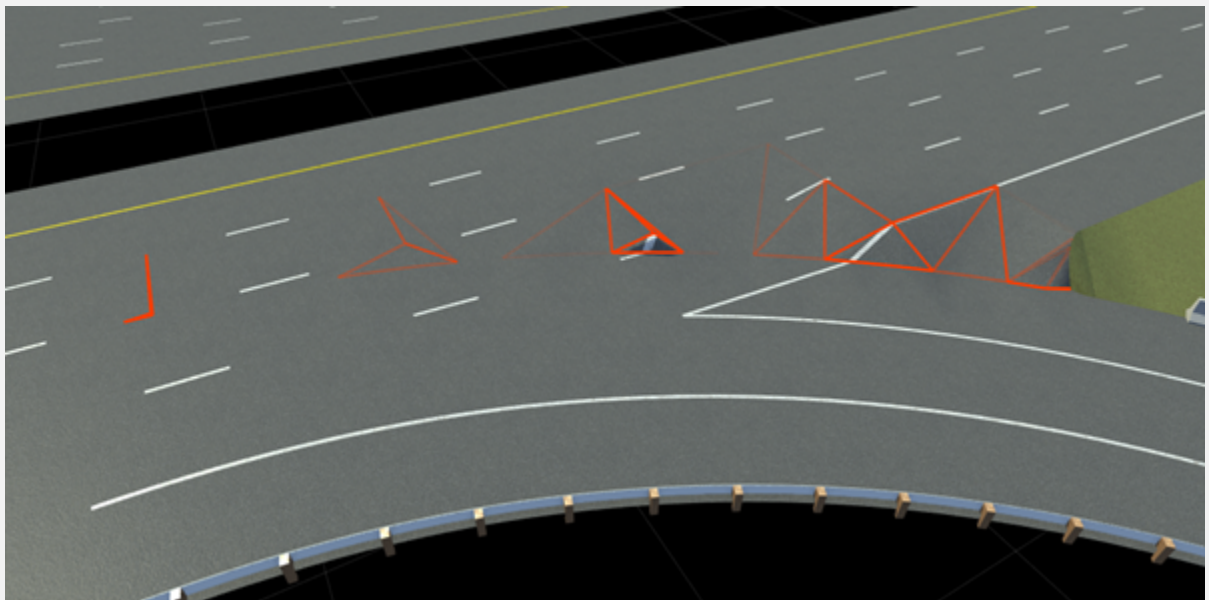
Detect Geometry Issues

Running **Detect Geometry Issues** prints the current state of the geometric issues to its own **Output** pane, and to the RoadRunner standard **Output** pane. Each issue contains its own URL that focuses the camera on each issue.

Found Geometry Issues   ✕

Output

Edge with angle 13.6431° at: (-78.9393, 16.5996, 6.73022)
Edge with angle 13.1922° at: (-70.9352, 14.8873, 6.97454)
Edge with angle 12.5162° at: (-85.3799, 24.9748, 6.21304)
Edge with angle 12.0009° at: (-69.0391, 15.9658, 6.71417)
Edge with angle 11.5995° at: (-70.2454, 16.3665, 6.64437)
Edge with angle 11.4919° at: (-81.25, 17.4688, 7.2)
Edge with angle 11.4918° at: (-81.25, 15.7188, 7.2)
Edge with angle 11.4452° at: (-71.439, 16.8032, 6.57456)
Edge with angle 11.442° at: (-74.9338, 18.3257, 6.36515)
Edge with angle 11.0295° at: (-85.008, 24.4595, 6.37663)
Edge with angle 10.6944° at: (-79.125, 14, 7.2)
Edge with angle 10.581° at: (-86.539, 22.8016, 7.16943)
Edge with angle 10.3945° at: (-86.6083, 22.7864, 7.19087)

Close

Point Editing

Some RoadRunner objects, such as prop instances, are modeled as points. This topic provides common steps to create, delete, and modify these point instances.

For general information about selecting and deleting objects, see "Design Scenes".

Create a New Point

1 Select the point tool that corresponds to the type you want to create (for example, select the **Prop Point Tool** for editing prop point instances).

2 Some tools will require an appropriate asset to be selected in the **Library Browser** before a curve can be created (for example, select **Prop Model Assets** if you are creating a prop instance).

3 Right-click to create a point. The new point is automatically assigned to the selected asset.

4 Optional: Continue holding the mouse button and drag to move the point after initial creation.

Move a Point

1 Select the point tool that corresponds to the type you want to modify (for example, select the **Prop Point Tool** for editing prop point instances).

2 Click and drag a point to move it to a new location.

Curve Editing
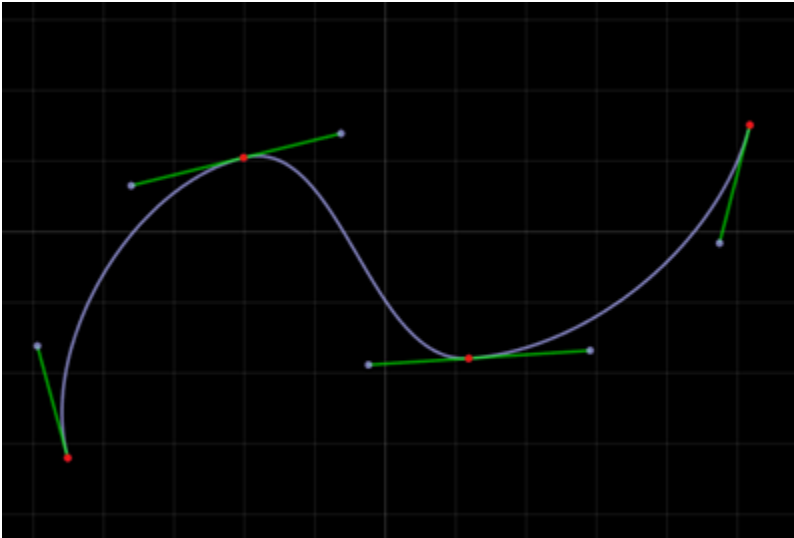
Some RoadRunner data models are built on top of curve sequences, including roads, prop curves, and marking curves. This topic provides common steps to create, delete, and modify these curve instances.

For general information about selecting and deleting objects, see "Design Scenes".

Create a New Curve

1 Select the curve tool that corresponds to the type you want to create (for example, select the **Marking Curve Tool** if you want to build a marking curve).

2 Some tools will require an appropriate asset to be selected in the **Library Browser** before a curve can be created (for example, select **Lane Marking Assets** to create marking curves).

3 Ensure that no objects are selected (for example, by selecting **Edit** and then **Deselect All** in the menu bar).

4 Right-click (and optionally drag) to create a curve with a single starting point. The new curve will automatically be assigned the selected asset.

5 Right-click (and optionally drag) to extend the curve by adding additional control points.

Extend a Curve at Its Ends by Adding Control Points

1 Select the curve tool that corresponds to the type you want to modify (for example, select the **Marking Curve Tool** to build a marking curve).

2 Select the curve you want to edit.

3 Click a control point at the end that you want to extend.

4 Right-click (and optionally drag) to add an additional control point.

Add Control Points to the Interior of a Curve

1   Select the curve tool that corresponds to the type you want to modify (for example, select the **Marking Curve Tool** to build a marking curve).

2   Select the curve you want to edit.

3   Right-click (and optionally drag) the curve at the location where you want to insert a new control point.

Move a Control Point

1   Select the curve tool that corresponds to the type you want to modify (for example, select the **Marking Curve Tool** to build a marking curve).

2   Select the curve you want to edit.

3   Click and drag a control point to move it to a new location.

Change the Tangents of a Curve

See "Tangent Editing" on page 2-70.

Polygon Editing
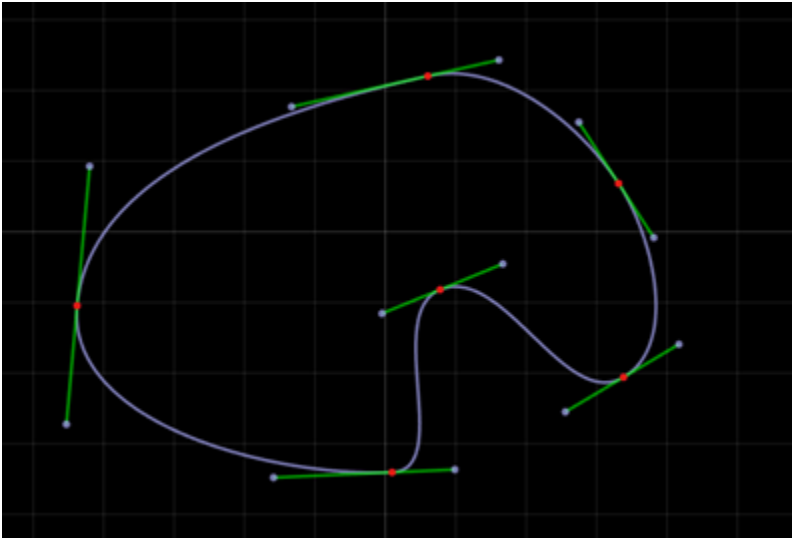
Some RoadRunner data models are polygon-based, such as prop and marking polygons. This topic provides common steps to create, delete, and modify these polygon instances.

For general information about selecting and deleting objects, see "Design Scenes".

Create a New Polygon

1   Select the polygon tool that corresponds to the type you want to create (for example, select the **Marking Polygon Tool** to build a marking polygon).
2   Some tools require an appropriate asset to be selected in the **Library Browser** before a polygon can be created (for example, select **Polygon Marking Assets** if you are creating marking polygons).
3   Ensure that no objects are selected (for example, by selecting **Edit** and then **Deselect All** from the menu bar).
4   Right-click (and optionally drag) to create a polygon with a single starting point. The new polygon will automatically be assigned the selected asset.
5   Right-click (and optionally drag) to extend the polygon and add additional control points.

Add Control Points to a Polygon

There are two ways to add control points to a polygon.

Insert a Control Point Next to an Existing Point

1   Select the polygon tool that corresponds to the type you want to modify (for example, select the **Marking Polygon Tool** to build a marking polygon).
2   Select the polygon you want to edit.
3   Select a control point next to the point you want to add.
4   Right-click (and optionally drag) to add additional control points.

Insert Control Points by Splitting a Polygon Edge

1   Select the polygon tool that corresponds to the type you want to modify (for example, select the **Marking Polygon Tool** if you want to build a marking polygon).

2   Select the polygon you want to edit.

3   Right-click (and optionally drag) the polygon edge where you want to insert a new control point.

Move a Control Point

1   Select the polygon tool that corresponds to the type you want to modify (for example, select the **Marking Polygon Tool** to build a marking polygon).

2   Select the polygon you want to edit.

3   Click and drag a control point to move it to a new location.

Change the Tangents of a Polygon

See "Tangent Editing" on page 2-70.

Tangent Editing

Some RoadRunner data models are built on top of curves and curve sequences, including roads, prop curves and polygons, marking curves and polygons, and the terrain surface graph. The control points of these curves contain tangents that can be adjusted to smooth or kink the resulting boundaries. This topic provides common steps for editing tangents and enforcing tangent continuity.

Adjust a Tangent

1 Click the parent object to expose the tangent views.
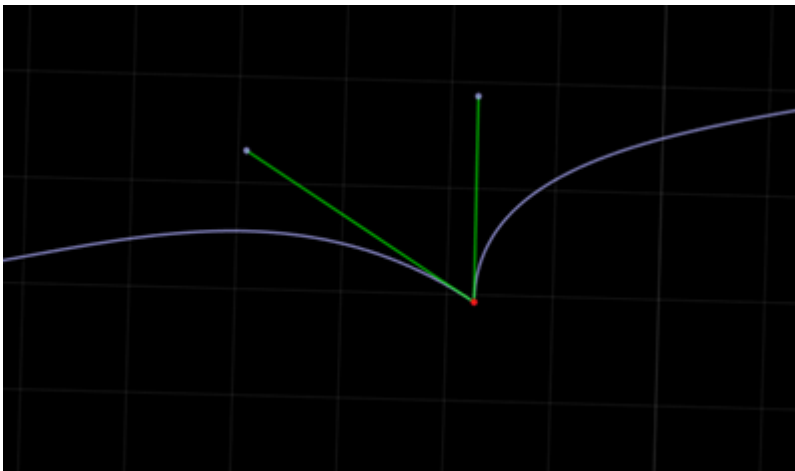
Note For some data types, such as road height profiles, the tangents are exposed once the parent is selected.

For other data types, such as curves or polygons, this might require clicking the parent object first to expose the control points.

2 Click the end point of the tangent handle.
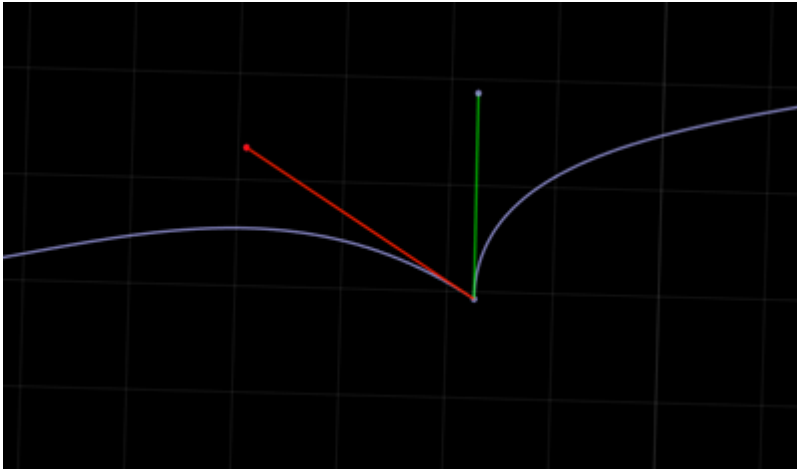
3   Click and drag to set the direction and scale of the tangent.
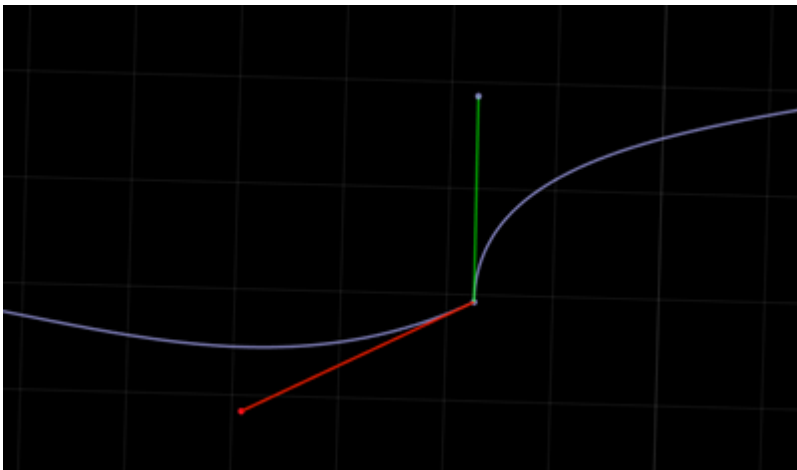
Make Tangents Continuous

To automatically enforce continuity, use the **Connect Tangents** operation:
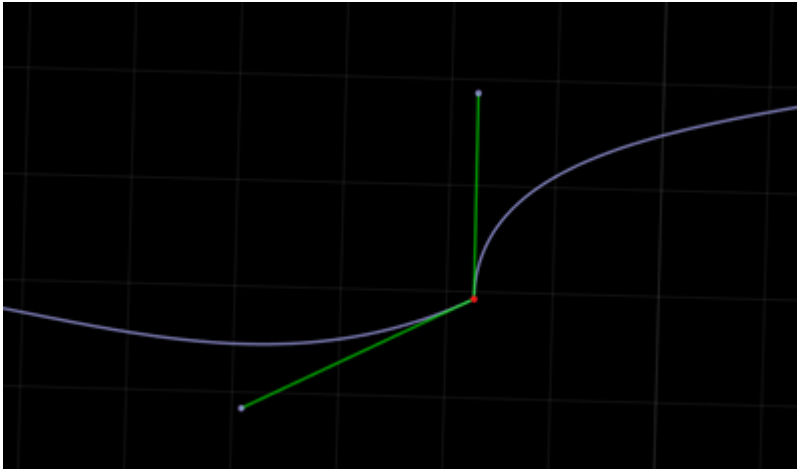
1   Click the parent object to expose the tangent views.

Note For some data types, such as road height profiles, the tangents are exposed once the parent is selected.

For other data types, such as curves or polygons, this might require clicking on the parent object first to expose the control points.

2

Click the **Connect Tangents** button.
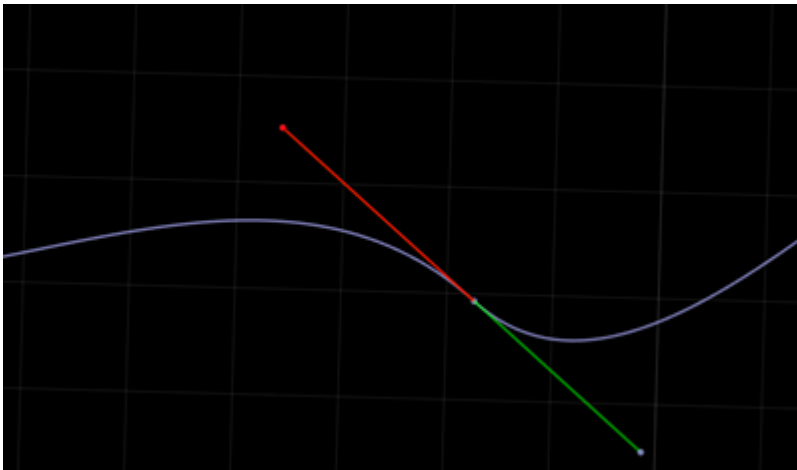
Tangents will now be enforced, even through additional edits.
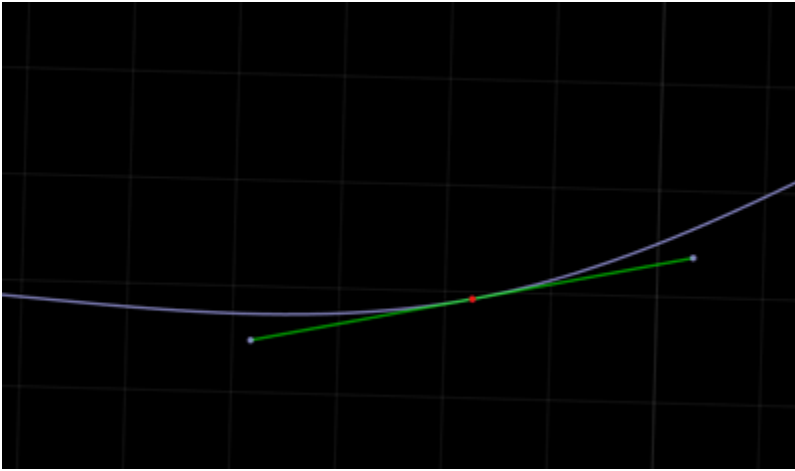
Make Tangents Discontinuous

To remove automatic continuity constraints, use the **Disconnect Tangents** operation:
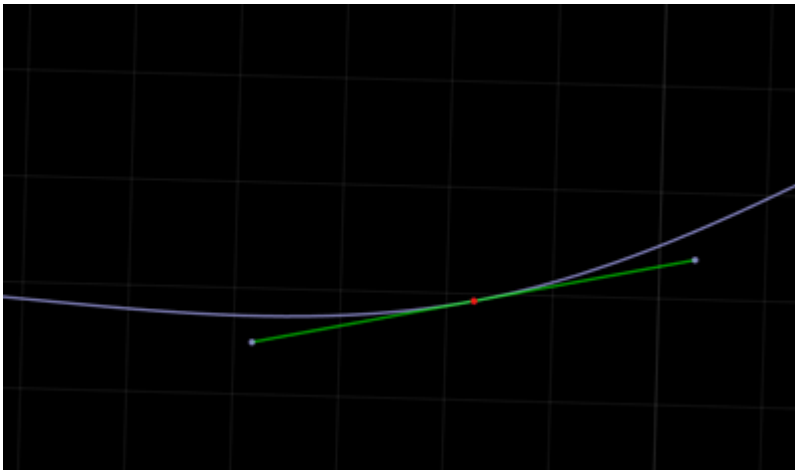
1   Click the parent object to expose the tangent views.

Note   For some data types, such as road height profiles, the tangents are exposed once the parent is selected.

For others, such as curve or polygons, this might require clicking the parent object first to expose the control points.

2

Click the **Disconnect Tangents** button.
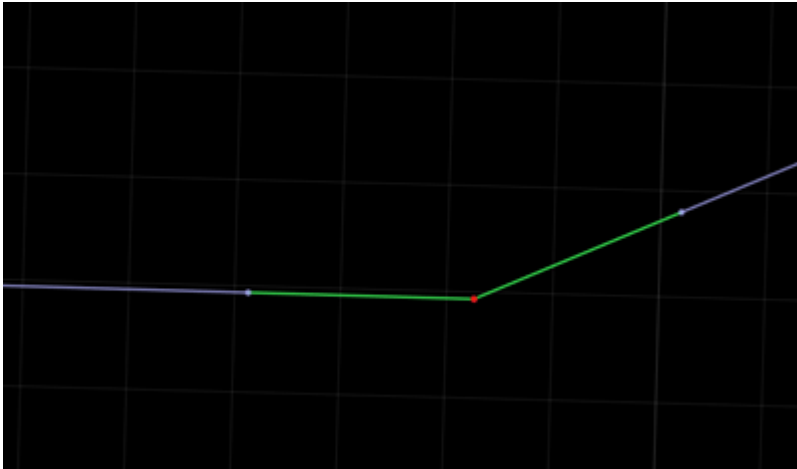
Curve Tangents

Point tangents views in RoadRunner Scenario can take on of the four states:

*   Automatic continuous
*   Automatic linear
*   Manual continuous
*   Manual linear

The default tangent state is automatic continuous. Automatic states update the tangents of the control points as their adjacent points' are updated. Manual states retain tangents and are not recomputed when the control points adjacent to it are updated. In the **Prop Curve Tool**, tangents remain automatic until the tangent is explicitly modified using the green control point handles. After that, the tangents become manual.

See Also

Related Examples

*   "Choose a RoadRunner Tool" on page 2-35

Span Editing

Various attributes are represented as parametric spans along lanes, roads, and other objects. This topic provides common steps to create, delete, and modify these span instances. Various tools use span editing concepts, such as the **Lane Marking Tool**, **Prop Span Tool**, or **Road Construction Tool**.

Span Overview

Span-based attributes are defined by the following components.

Parent Object

Span-based attributes are defined parametrically along a curve-based parent object. Typically, the parent object is either a road (as in the **Road Construction Tool**) or a lane (as in the **Lane Marking Tool**).

Span Nodes
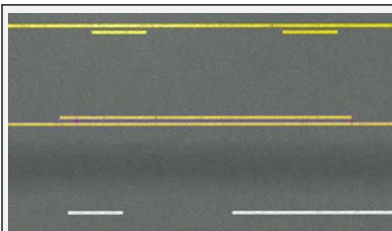
Span nodes (red circles) selected in the **Lane Marking Tool**. These nodes indicate locations where the marking type changes along the lane.	Span nodes (red lines) selected in the **Road Construction Tool**. These nodes indicate where bridges start and end along a road.

Span nodes are parametric objects along a parent curve that define where attribution changes. The visual representation of nodes differs depending on the tool (as shown in the previous images).

Span nodes can be moved along the parent curve. Nodes can also be added along a curve, and existing nodes can be deleted.

Node locations are automatically updated when the parent curve is modified (for example, when the parent road's shape is changed).

Span End Nodes
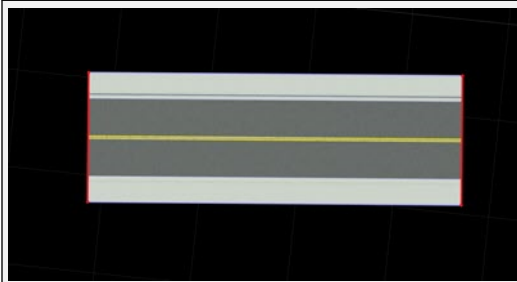
Span end nodes selected in the **Road Construction Tool**.

Span end nodes are a special type of span node that lie at the beginning and end of the parent curve. For most types of span-based attributes, these span end nodes are automatically created and cannot be deleted.

Spans
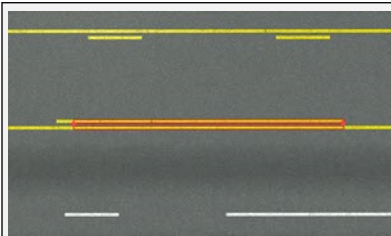
Span selected in the **Lane Marking Tool**. This span defines the lane marking type between two nodes.
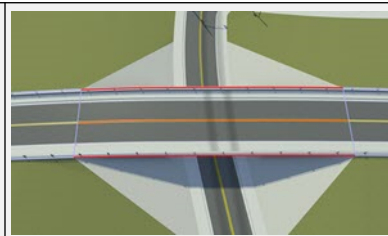
Span selected in the **Road Construction Tool**. This span defines the construction type (for example, bridge) between two nodes.

A span is a range along a parent curve bounded by two span nodes. For most types of span-based attributes, the span is automatically created between the span end nodes and cannot be deleted.

Select a Span or Span Node

The steps to select a span differ slightly depending on the tool, but the steps are typically similar to the following.

1   Select the parent object containing the span (typically either a road or a lane on a road).
2   Select the desired span or span node.

Create a New Span Node

New span nodes are created by splitting a span into two spans.

1   Select the parent object containing the span.
2   Right-click an existing span at the location where you want to insert the new node.

Note   In most cases, any attributes stored in the span are copied into the two new spans.

Edit Attributes of a Span or Span Node

1   Select a span or span node.

2   Adjust the properties in the **Attributes** pane.

Alternatively, for asset-based attributes such as in the **Lane Marking Tool** and **Prop Span Tool**, click and drag a compatible asset type from the **Library Browser** to the span or span node.

Note   Some span-based attributes only store data on the spans, others only store data on the span nodes, and some store data on both.

Span nodes always have a "Distance" attribute that defines the distance of the node along the parent curve.

Move a Span Node

1   Select the parent object containing the span.

2   Click and drag the node along the parent curve.

Alternatively:

1   Select the parent object containing the span.

2   Select the span node.

3   In the **Attributes** pane, adjust the **Distance** attribute.

Note   Most span nodes cannot be moved past another node, and must remain a minimum distance from surrounding nodes.

Delete a Span Node

1   Select the parent object containing the span.

2   Select the span node.

3   Delete the span node.

Tips for Deleting Nodes

- In most cases, the span end nodes (the nodes at the end of the parent curve) cannot be deleted.
- Deleting a span node combines the two attached spans into a single span. In most cases, the single span receives the attributes of the longer span. The shorter span is removed.

Region Graph Editing
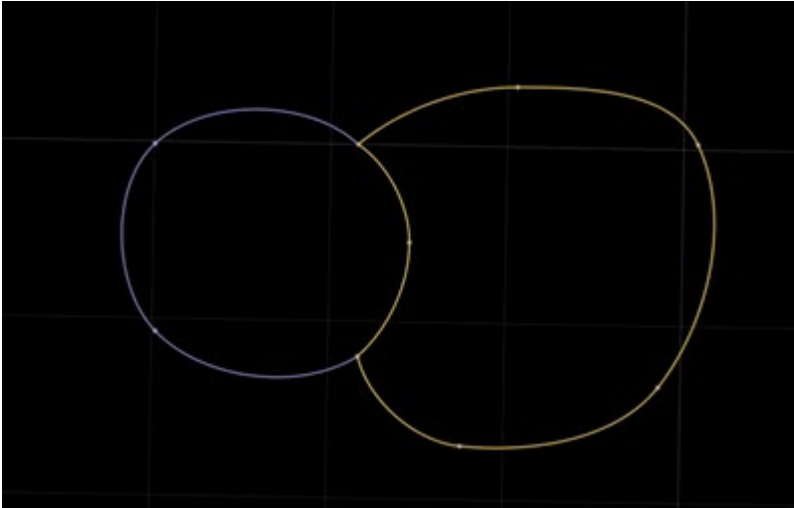
Some RoadRunner data models are built on top of graphs of curve-bounded regions. This topic provides common steps to create, delete, and modify these region graphs.

Many of the editing concepts for region graphs are similar to the concepts for "Design Scenes". For example, you can create, edit, and delete curve-based graph edges that behave like most curves in RoadRunner.

Region graphs differ in two regards:

- Edge Connectivity — Graph edge curves can be connected end-to-end.
- Regions — Whenever a closed loop of graph edges is formed, a region is created in the interior.

Refer to the "Design Scenes" page for general information about selecting and deleting objects.

Create a Graph Edge Curve

1   Select the graph region tool that corresponds to the type you want to create (for example, select the **Surface Tool** if you want to edit surfaces).

2   If you want to start the new edge at an existing node, select the existing node. Otherwise:

   a   Ensure that no objects are selected (for example, by using the **Edit >** > **Deselect All** option in the menu bar).

   b   Right-click (and optionally drag) to create an initial graph node.

3   Optional: Move the pointer over an existing node if you want to end the curve at that node.

4   Right-click (and optionally drag) to create a second graph node and a graph edge curve in between.

Split a Graph Edge Curve

1   Select the graph region tool that corresponds to the type you want to create (for example, select the **Surface Tool** to edit surfaces).

2   Right-click a graph edge curve to split it into two curves.

Move a Graph Node

1   Select the graph region tool that corresponds to the type you want to create (for example, select the **Surface Tool** to edit surfaces).

2   Click and drag the graph node you want to move.

Change the Tangents of a Graph Edge Curve

See "Tangent Editing" on page 2-70.

Create a Region

Regions are automatically created whenever a closed loop of graph edges is formed:

Sequence of connected graph edge curves (no region)	Region is automatically created when the open ends are connected.

Split a Region

Each closed loop of graph edges automatically forms a region, so one region can be split into two by forming a path of graph edge curves between two of the points on the region exterior:
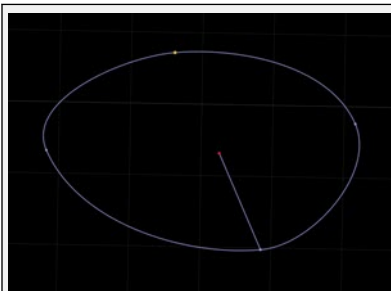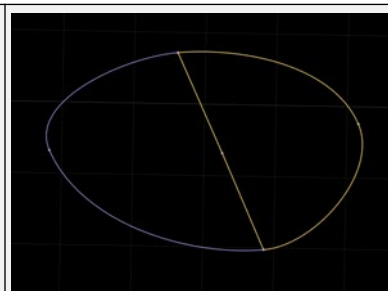
Single region	Region is automatically split into two regions when two closed loops are formed.

Regions With Holes

Only the **Surface Tool** fully supports holes.

Merge Multiple RoadRunner Scenes

In RoadRunner, scenes can represent anything from a small area, such as a single intersection, to a large area, such as a portion of a city. A scene can contain multiple roads, intersections, road markings, props, terrain sections, and various other scene aspects.

In RoadRunner, you can create many scenes within the same project, and the scenes can share assets within the project. A scene file contains an area that includes objects such as roads, surfaces, props, and other scene aspects. Individual scenes are saved as `.rrscene` files, typically in the `Scenes` folder of a project. For a more complete understanding of creating projects and scenes, see "RoadRunner Project and Scene System" on page 2-2.

Because large scenes can take several minutes to load, and working on such scenes single-handedly can become cumbersome and time-consuming, you can improve your efficiency by dividing a large scene into multiple smaller scenes. Once work on the smaller scenes is complete, RoadRunner enables you to merge the separate scene files into a single scene file. When you merge a scene directly from a scene file into a currently open scene, RoadRunner automatically adjusts the location of the incoming scene to match the current scene. This functionality enables multiple users, working together on a large scene, to can plan out how to divide a scene ahead of time.

When merging scenes, RoadRunner considers the geolocation of the scenes. In RoadRunner, the scenes can have a latitude-longitude pair set as their world origin. If the scene has a latitude-longitude pair set, then RoadRunner takes that into account and positions the roads, props, and surfaces of the incoming scene relative to that latitude-longitude pair. For more details on the local coordinate system and georeferencing in RoadRunner, see "Coordinate Space and Georeferencing" on page 2-10.

Merge Two Non-Geolocated Scenes

To merge two scenes that do not have specified world origins, follow these steps:
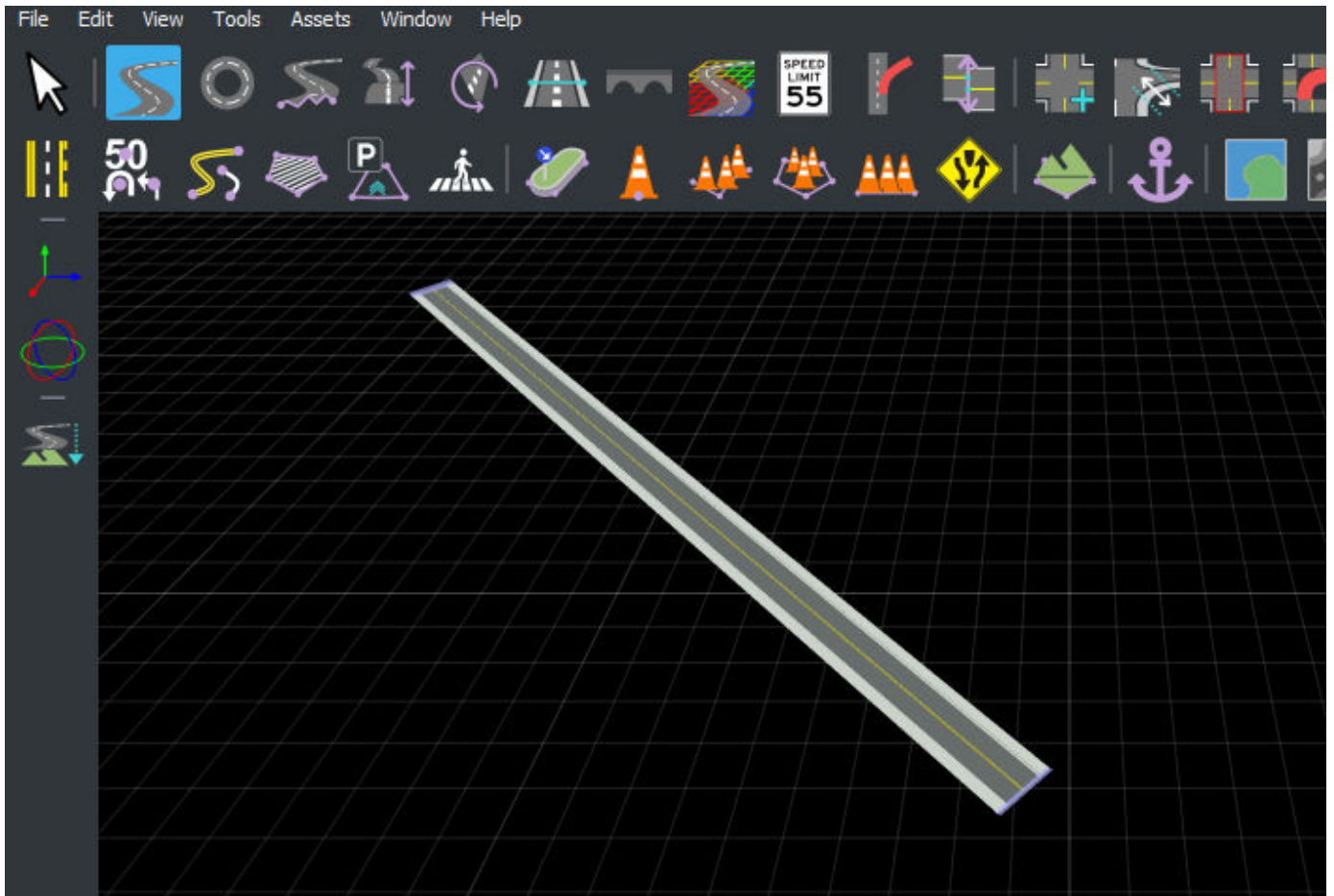
1   Open an existing scene in RoadRunner.

2   From the **File** menu, select **Open and Merge Scene File**. Alternatively, you can press **Ctrl +Shift+O**.
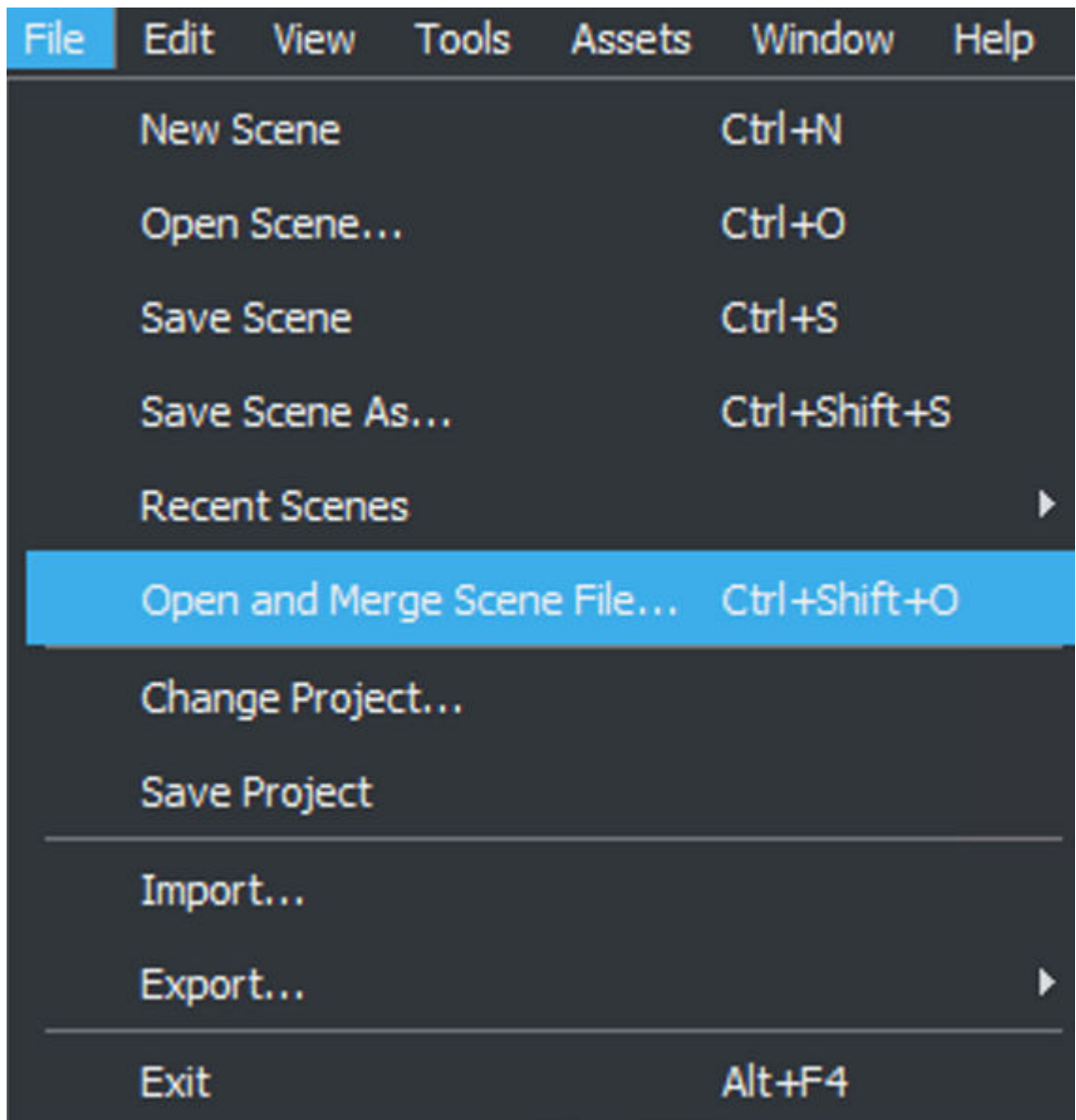
3 Select the scene you want to merge into your current scene, and click **Open**.

4 RoadRunner places the selected scene in the current scene.
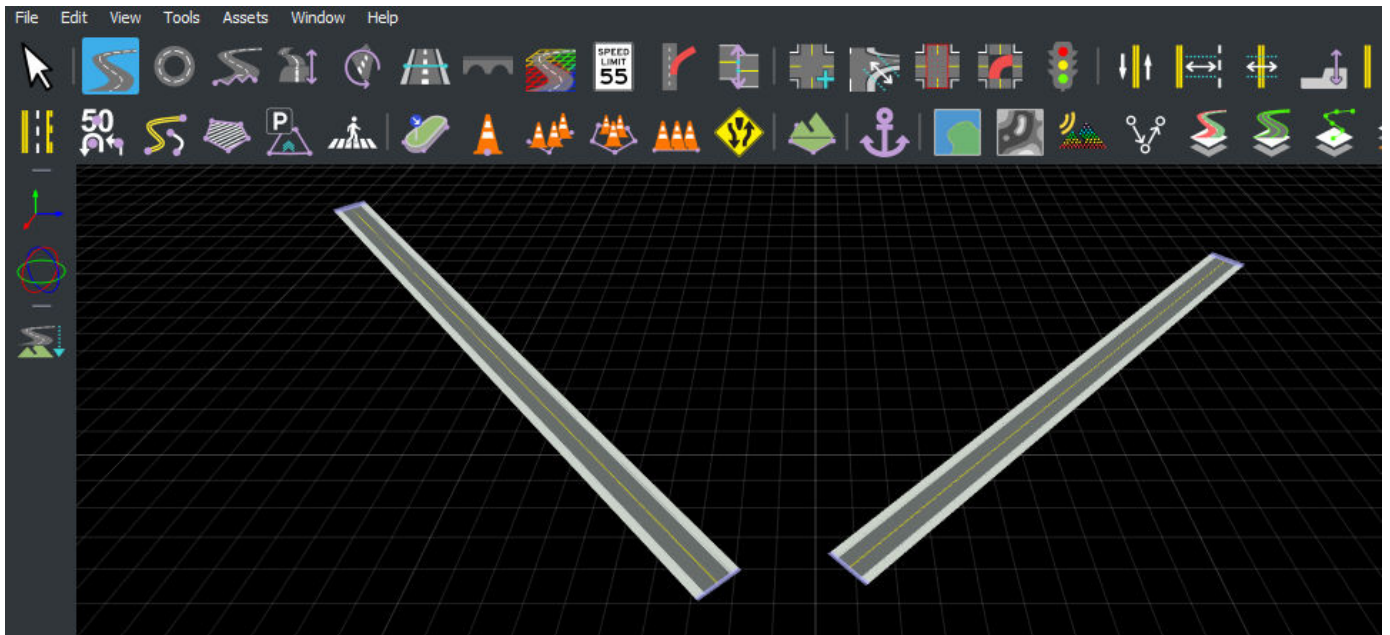
Merge Two Geolocated Scenes

To merge two scenes that have specified world origins, follow these steps:

1. Open an existing scene in RoadRunner. If the scene does not already have a specified world origin, set one by using the **World Settings Tool**. To view the world origin details for this scene,
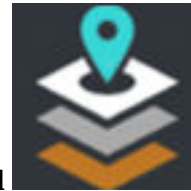
   on the RoadRunner toolbar, click the **World Settings Tool** ![icon]. The **Attributes** pane and the editing canvas show the latitude and longitude values for the world origin of this scene.
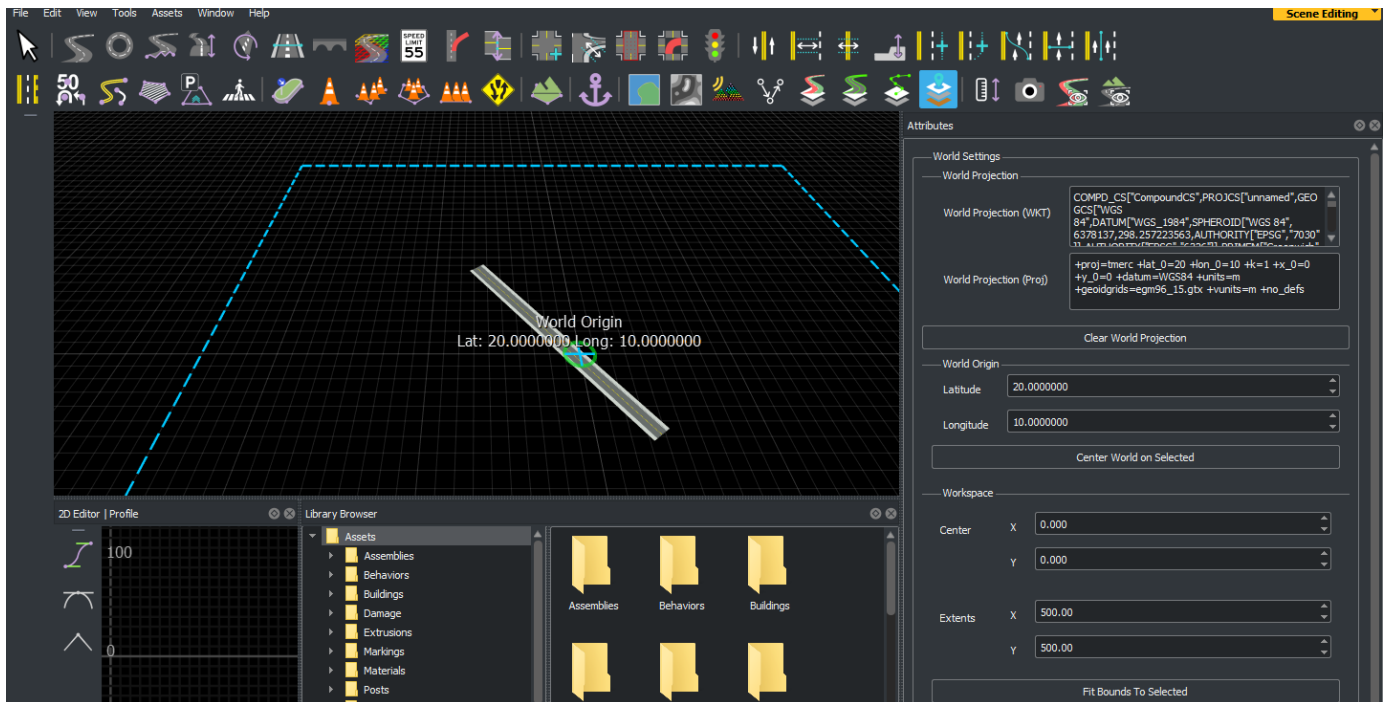
2 From the **File** menu, select **Open and Merge Scene File**. Alternatively, you can press **Ctrl +Shift+O**.
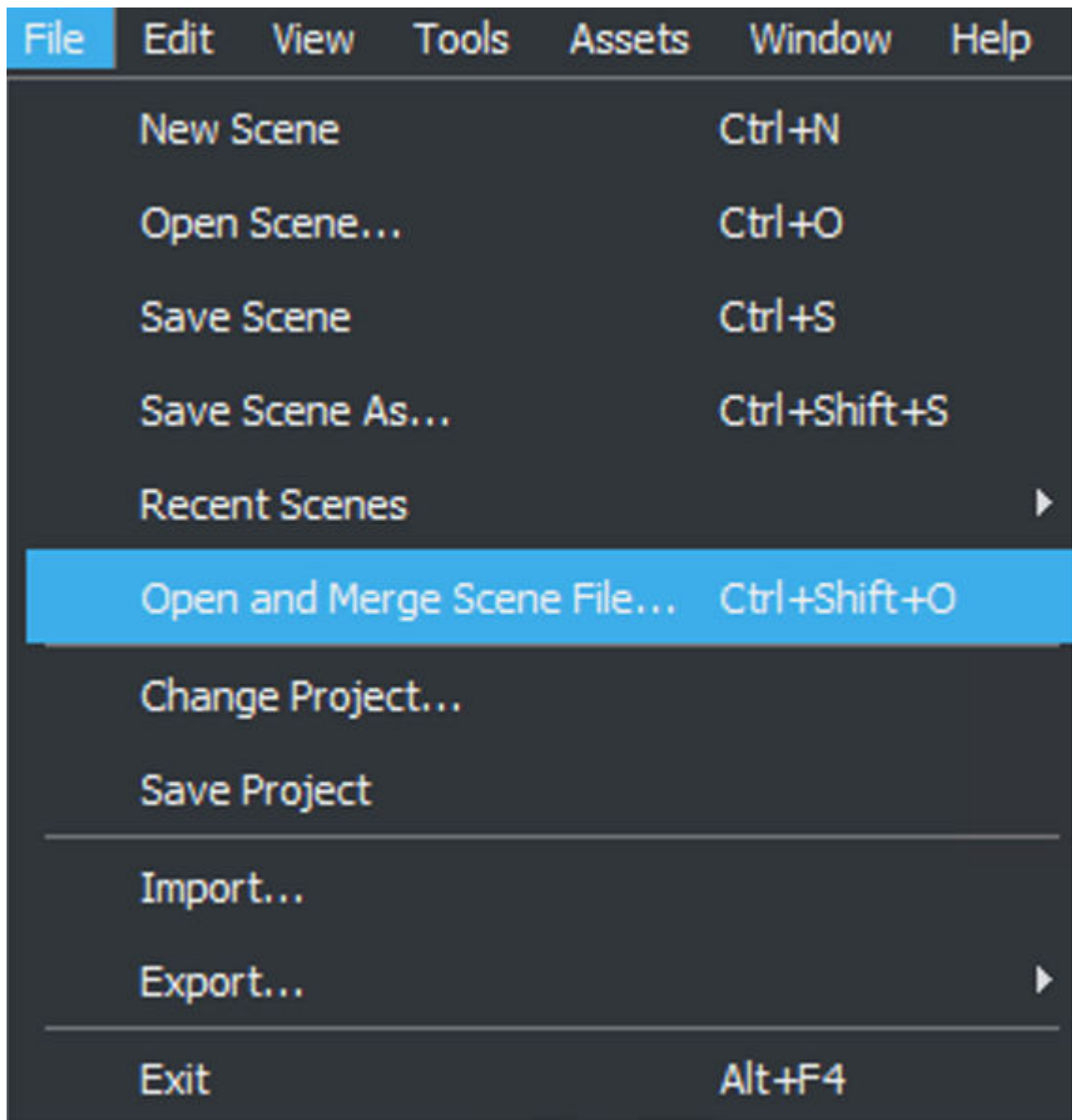
3 Select the other geolocated scene you want to merge into your current scene, and click **Open**. The editing canvas shows the selected scene to be merged along with its world origin details.
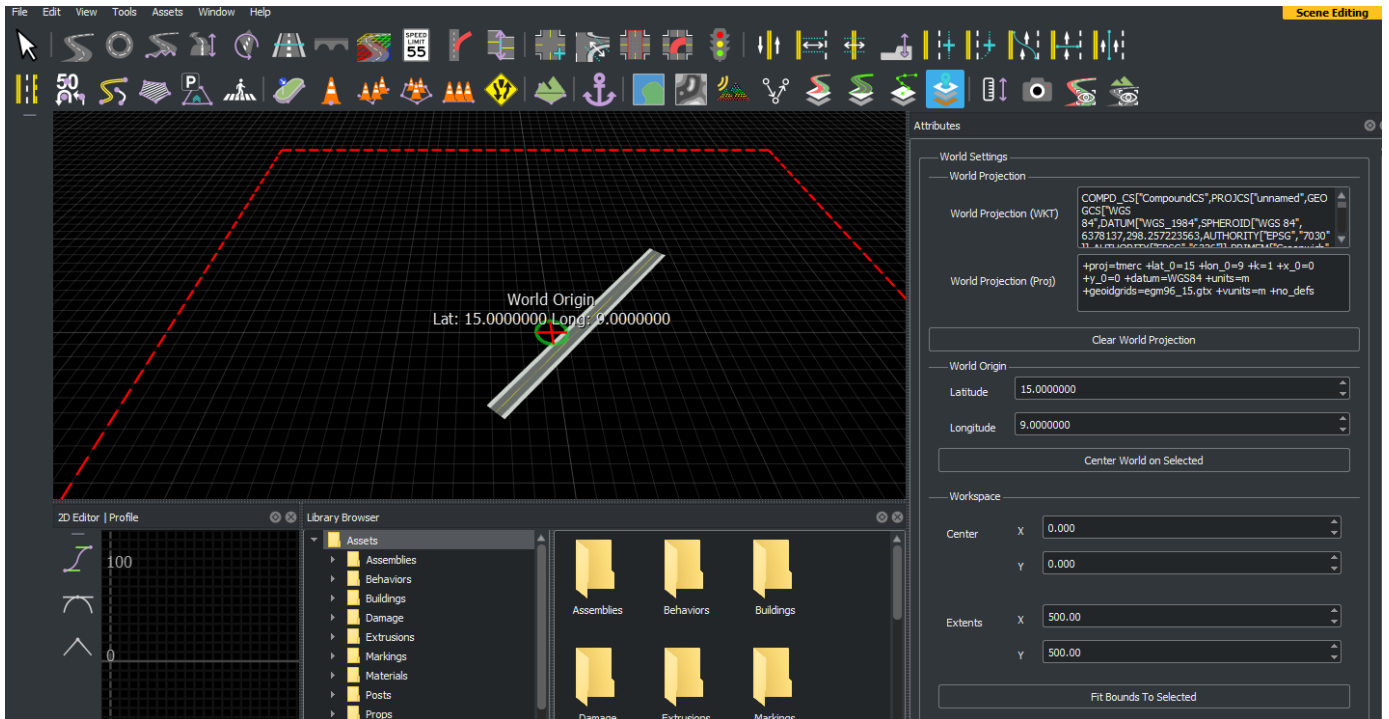
4 If the world origins of both the scenes are close enough to one another, RoadRunner merges the scenes and locates them appropriately, using the world origin of the current scene as the base.
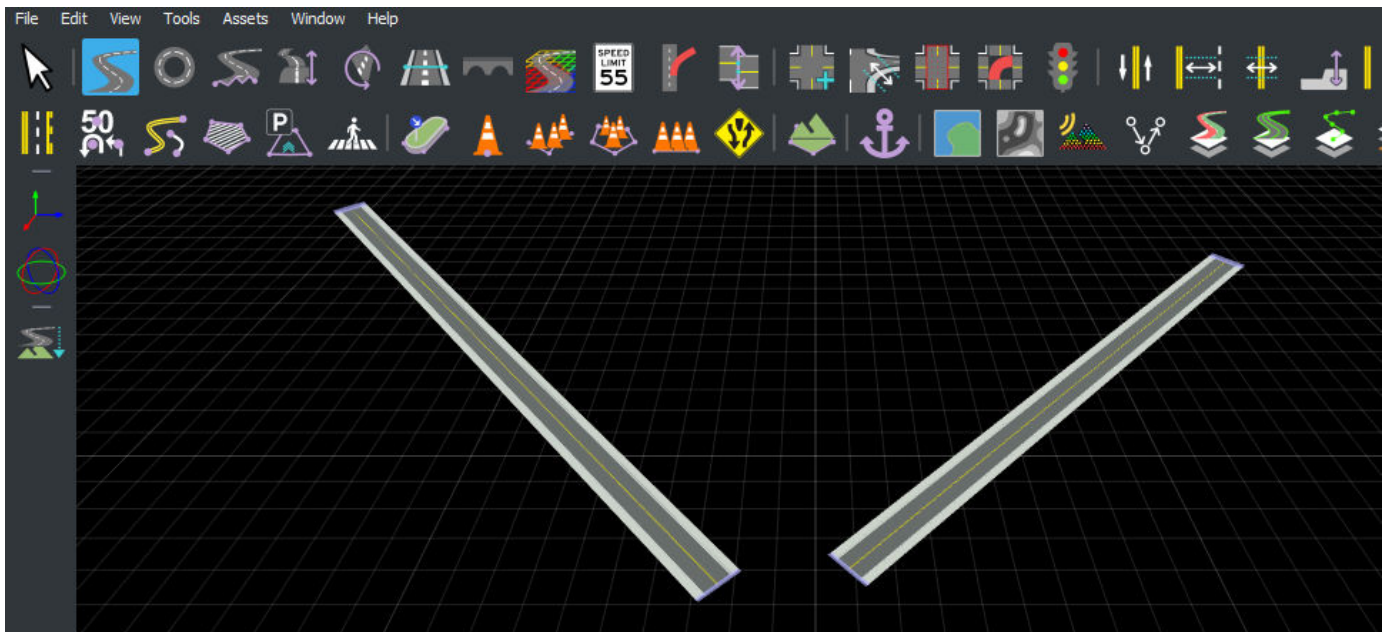
5 If the world origin of the incoming scene is located too far away from the world origin of the current scene, a dialog prompts you to either cancel the operation, or ignore the incoming world origin.
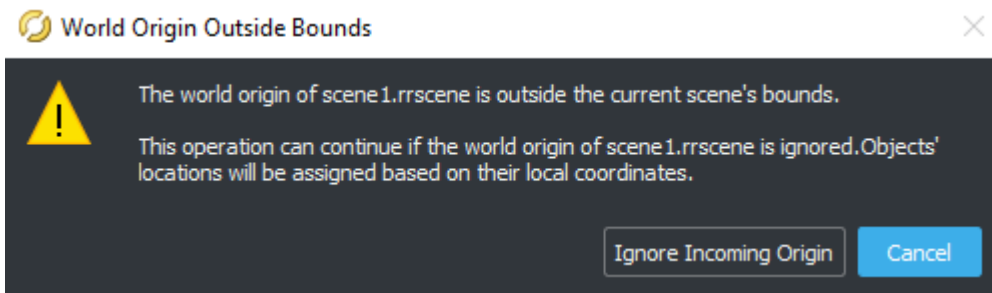
6 If you select **Ignore Incoming Origin**, RoadRunner merges the incoming scene into the current scene based on the local coordinates of the incoming scene. If you select **Cancel**, RoadRunner stops the merge operation.
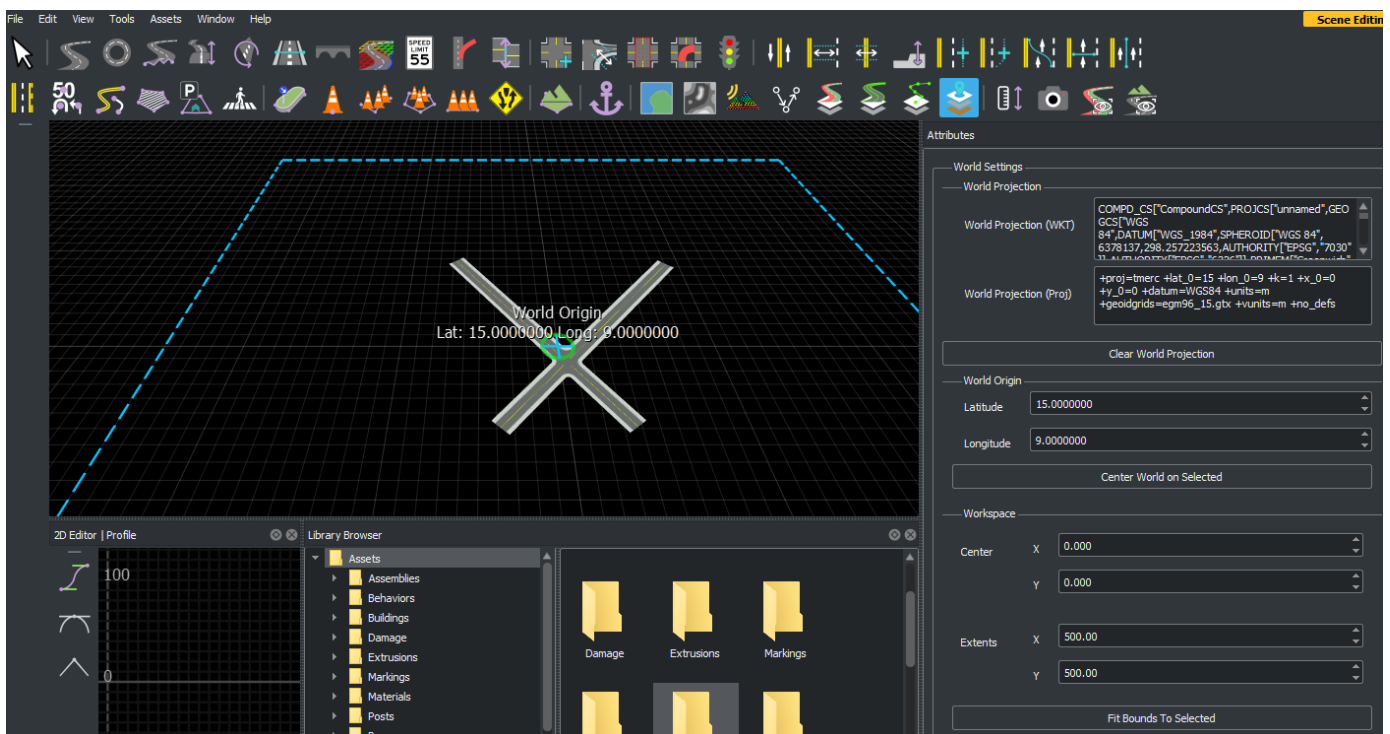
Merge Geolocated Scene to Non-Gelocated Scene

To merge a geolocated scene into a non-geolocated scene, follow these steps:

1 Open an existing scene in RoadRunner. Ensure that the scene does not have a specified world origin. To view the world origin details for this scene, in the RoadRunner toolbar, click the **World**
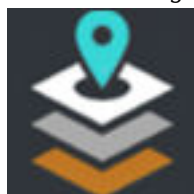
Settings Tool . The **Attributes** pane and the editing canvas show whether the scene has latitude and longitude values set for the world origin.
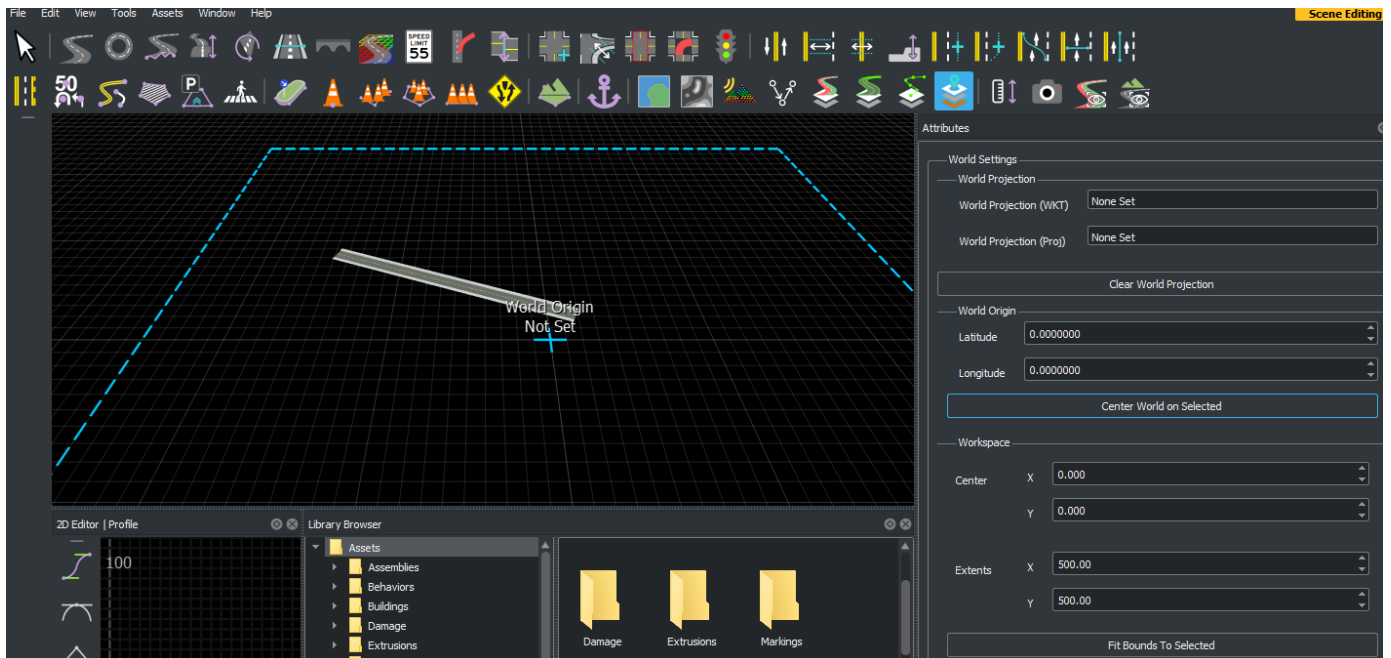
2 From the **File** menu, select **Open and Merge Scene File**. Alternatively, you can press **Ctrl +Shift+O**.
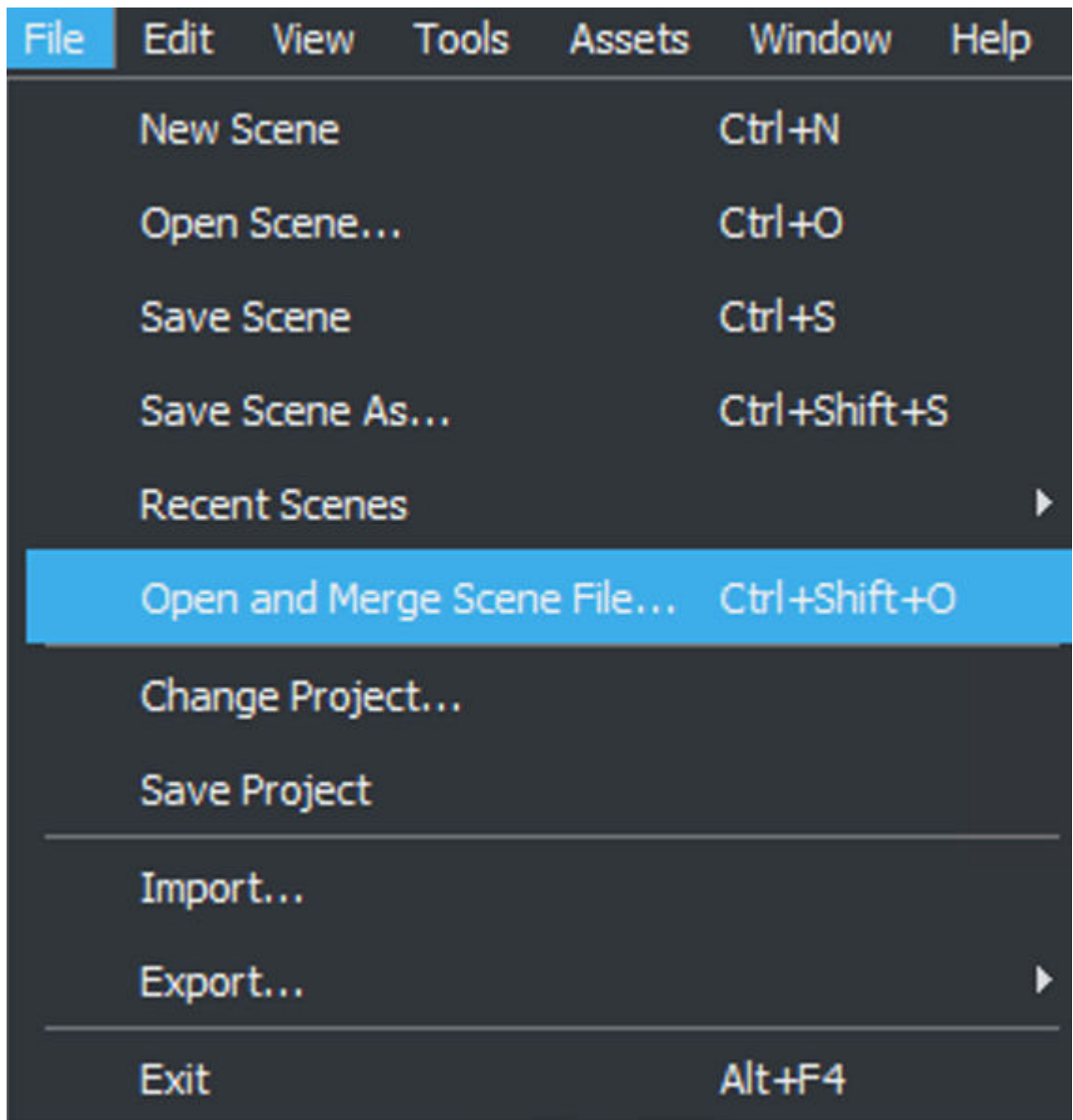
3   Select the other scene you want to merge into your current scene, and click **Open**. The editing canvas shows the selected scene to be merged. Verify that the scene has a specified world origin by using the **World Settings Tool**. The **Attributes** pane and the editing canvas show whether the latitude and longitude values of the world origin for this scene have been specified.
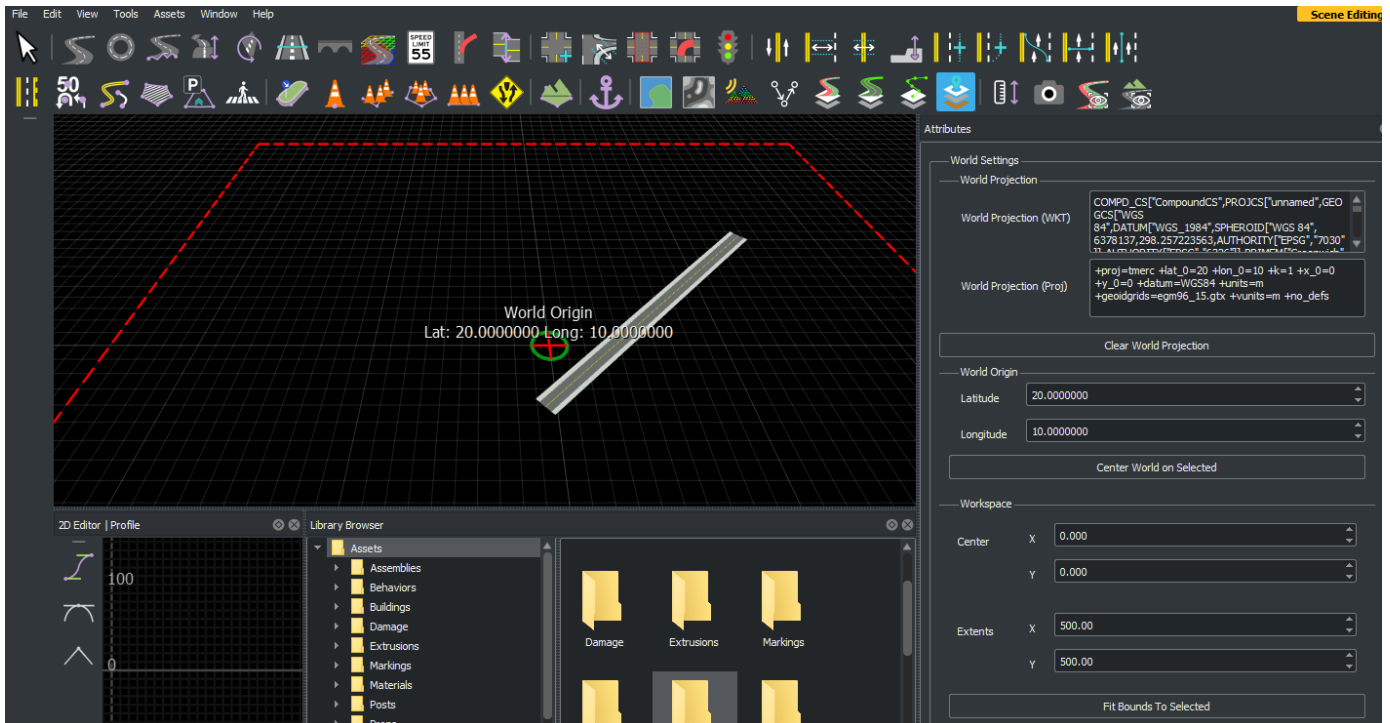
4   A dialog box prompts you to either ignore the incoming world origin, set the current scene to use the incoming world origin, or cancel the operation.
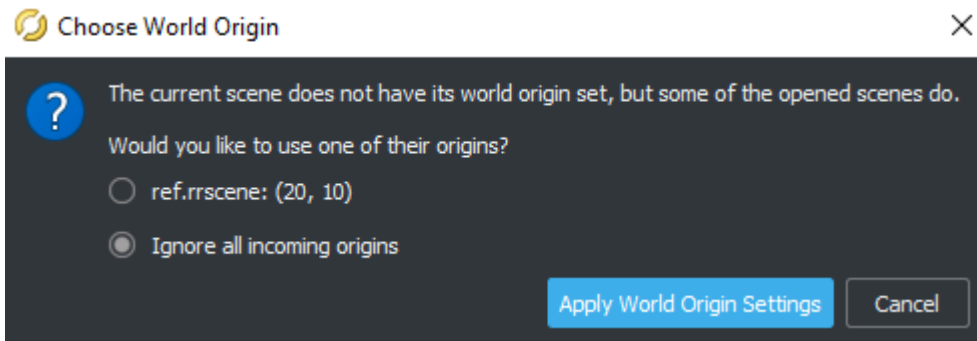
5   If you select **Ignore all incoming origins** and click **Apply World Origin Settings**, RoadRunner merges the incoming scene into the existing scene using only local coordinates. The resulting merged scene does not have a specified world origin.

6   If you select the world origin of the incoming scene file and click **Apply World Origin Settings**,
    RoadRunner merges the incoming scene into the existing scene, and assigns the world origin of
    the incoming scene to the resulting merged scene.

Limitations

- RoadRunner does not enable you to merge scenes from different projects by default. To merge a scene located in a different project, you must first copy the scene to the same location relative to the `.rrproj` file.

See Also

"Create Simple RoadRunner Scene" on page 1-19

Graphics and Startup Issues

System Requirements

RoadRunner is primarily a 3D graphics application. It requires a graphics card with support for OpenGL version 3.2 or higher.

Check that your system meets the minimum RoadRunner System Requirements on page 1-3. RoadRunner might still work with some lower specification system version, but you might experience poor performance.

Graphics Drivers

If you are experiencing rendering issues or crashes on startup, check to make sure that you are running the latest graphics drivers for your system. Some computer manufacturers install custom or unstable drivers. Windows Update has also been known to automatically install problematic graphics drivers.

Downloading drivers directly from your graphics card manufacturer is recommended. Common graphics card manufacturers include:

- NVIDIA (drivers download page)
- AMD (drivers download page)
- Intel (drivers download page)

On Linux, if your graphics card manufacturer does not supply drivers for Linux, try updating your Mesa graphics drivers.

If you are unsure which type of graphics card you have, you can often determine that by visiting your computer manufacturer's web page and searching for drivers for your system. This search usually involves entering a model (or serial) number or installing an autodetection application.

On Windows, you can determine which graphics cards you have installed by following these steps:

1   Click the **Start** button and type Run.

2   Type dxdiag and press **Enter**.

3   Inspect the **Name** and **Manufacturer** on the **Display** tabs. You might have multiple graphics cards installed, so inspect each listed **Display** tab.
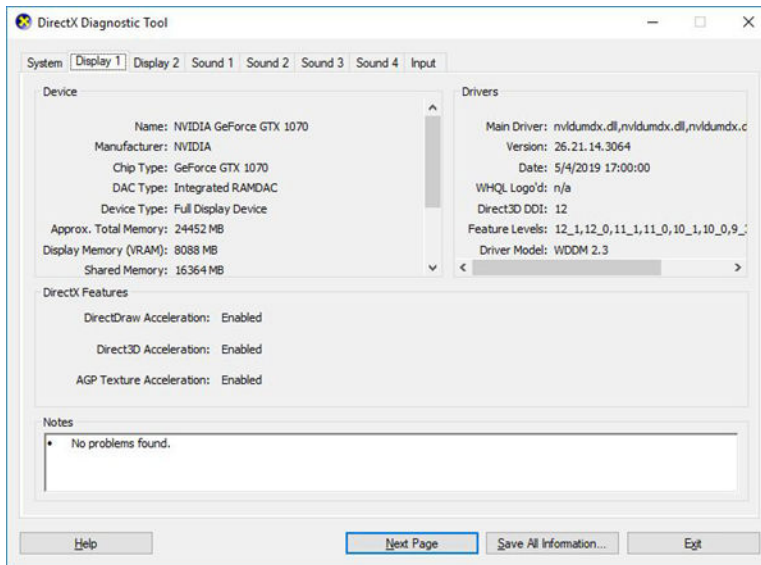
Laptops

RoadRunner can run on laptops with sufficiently powerful graphics cards. Newer laptops with low-powered graphics cards (such as Intel embedded graphics chips) might still be able to run RoadRunner acceptably.

To conserve battery life, modern laptops often have two graphics cards, for example, higher-powered NVIDIA graphics combined with lower-powered Intel embedded graphics. In these cases, RoadRunner requests to use the higher-power graphics card, but there is no guarantee that your system will obey that request.

If your laptop has multiple graphics cards (often true if your laptop advertises an NVIDIA or AMD graphics cards), then it is recommended that you check that your system provides the higher-power graphics card to RoadRunner.

The steps for this process differ depending on the laptop manufacturer, graphics driver, and other factors. For additional help, see these links:

- `https://www.techadvisor.com/article/727646/how-to-set-a-default-graphics-card.html`
- `https://www.nvidia.com/content/Control-Panel-Help/vLatest/en-us/mergedProjects/nv3d/Manage_3D_Settings_(reference).htm` (see "Program Settings" section; for RoadRunner, select the **high-performance NVIDIA processor** option)

Remote Desktops

If you are using RoadRunner while connected to a remote desktop, you might encounter performance issues. To resolve these issues, consider using one of these options.

NVIDIA Remote Desktop Acceleration

If you are using Microsoft® Remote Desktop on newer NVIDIA GeForce drivers, follow the procedure described in the "Accelerate Windows Remote Desktop" section of `https://developer.nvidia.com/physx-sdk`.

Chrome Remote Desktop

Chrome™ Remote Desktop from Google® provides a solution for remote access that supports newer OpenGL applications. For details, see `https://remotedesktop.google.com/`.

Video Card Connection

On desktops, you might also want to check that your monitor is connected to a video card, rather than to a low-powered graphics card (also called "onboard" or "integrated" graphics). Trace your monitor cable to make sure that it is not plugged into the built-in graphics port, but rather into a video card in one of the expansion slots.

Further Support

If the previous information does not resolve your issue, report your issue to MathWorks Technical Support. It is helpful to include your log files. For more details, see "Obtain RoadRunner Log Files" on page 2-98.

Obtain RoadRunner Log Files

While debugging RoadRunner issues, MathWorks Technical Support might request your RoadRunner log files. Log files are all messages, warnings, and errors related to RoadRunner that are printed to the **Output** pane. This image shows a sample **Output** pane with logged information.
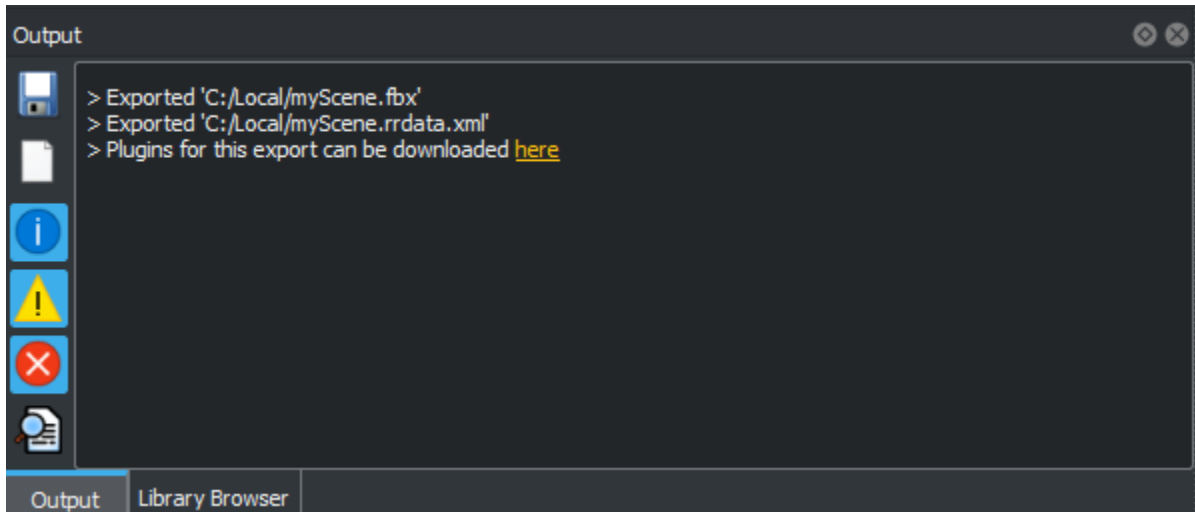
This information can be helpful for debugging certain problems, especially problems related to loading asset files.

Locate Log Folder

To obtain the log from the current RoadRunner session, from the RoadRunner menu, select **Tools > Debug > Open Log Folder**

To locate the log folder that contains log files for all RoadRunner sessions, in Windows, click **Start**, and then type:

```
%appdata%\MathWorks\RoadRunner\<release version>\Logs
```

To locate log files in Linux, navigate to this folder:

```
~/.local/share/MathWorks/RoadRunner/<release version>/Logs
```

Provide Log File Contents to MathWorks Technical Support

1   Locate the log folder (see previous section).
2   Zip (or tar) the contents of the log folder.
3   Attach the zip file to a new or existing ticket for MathWorks Technical Support.

See Also

Related Examples

- "Graphics and Startup Issues" on page 2-94

Import Data

Importing ASAM OpenDRIVE Files

RoadRunner can visualize and import OpenDRIVE 1.4, OpenDRIVE 1.5, and ASAM OpenDRIVE 1.6 data, converting the data to the internal road format during import. The ASAM OpenDRIVE data is visualized before import. The import option is designed to enable editing of ASAM OpenDRIVE files.

Import ASAM OpenDRIVE File and Build Scene

1 In RoadRunner, add the ASAM OpenDRIVE file to a folder in the **Library Browser**.

2 Drag the ASAM OpenDRIVE file from the **Library Browser** into the scene. This action switches to the **OpenDRIVE Viewer Tool**.

3 In the 3D scene, click the imported file to select it.

4 In the **Attributes** Pane, click **Set Custom Projection** or select an appropriate **Projection Mode** from the dropdown menu to correctly position the latitude and longitude of ASAM OpenDRIVE data in the RoadRunner scene.

The **Set Custom Projection** option opens a dialog box to configure the **Tranverse Mercator** projection. This option is recommended when the projection details for the imported ASAM OpenDRIVE data are unknown. Alternatively, you can select one of these available **Projection Modes** from the dropdown menu.

Projection Mode	Description
No Projection	This projection mode extracts the X, Y coordinates of the points from the ASAM OpenDRIVE data directly and places them on the RoadRunner scene. This mode is not recommended but can be used to import synthetic scenes (i.e., scenes that are not based on any real-world reference).
Translate Only	This projection mode imports the ASAM OpenDRIVE data such that the `latitude` and `longitude` of 0, 0 position in the imported data is positioned in the corresponding `latitude` and `longitude` position in the RoadRunner scene. ASAM OpenDRIVE `<geometry>` remains unaffected so the curvature information is preserved. This projection mode is recommended for synthetic scenes where the `<geometry>` information needs to be preserved accurately. It is also recommended for real-world scenes where the projection details for the ASAM OpenDRIVE data are unknown but the `latitude` and `longitude` of 0, 0 position can be determined.

Projection Mode	Description
Full Projection	In this projection mode, the `latitude` and `longitude` of any point in the imported ASAM OpenDRIVE data correspond to the `latitude` and `longitude` of the same point in the RoadRunner scene. The ASAM OpenDRIVE data is reprojected to exist in RoadRunner's local **Tranverse Mercator** projection. This projection mode is highly recommended for scenes based on real-world data or ASAM OpenDRIVE files containing projection information in the `header`. It is not needed for synthetic scenes.

5   In the toolbar on the left, click the **Build Scene** button.

6   In the Import ASAM OpenDRIVE dialog box, select the import options, such as which objects, signals, and markings to build in the scene.

The OpenDRIVE importer uses a configuration XML file to map ASAM OpenDRIVE `<object>`, `<signal>`, and `<marking>` entries to RoadRunner assets. This configuration file is also used to define the correlation during export. For more details, see "Convert Asset Data Between RoadRunner and ASAM OpenDRIVE" on page 5-16.

The table shows the import options that you can select.

Option	Description
Import signals	Select this option to map all `<signal>` entries to signals or signs.
Import props	Select this option to map all `<object>` entries to props or markings.
Use hOffset relative to orientation	Select this attribute to import the `<hOffset>` (heading offset) values of `<signal>` entries as being relative to `<orientation>`, which is the direction of travel of the road that the signal applies to. By default, the heading offset is relative to the heading of the road, regardless of its direction of travel

7   Click **Import**. The Import ASAM OpenDRIVE Results dialog box displays any warnings and errors that occurred during import. Close this dialog and inspect the imported scene by clicking to a different tool.

Explicit Lane Direction Priority

This section states the priority for how lane direction is set is when importing ASAM OpenDRIVE files to RoadRunner. In decreasing order of priority:

1. If an ASAM OpenDRIVE driving lane has its `travelDir` attribute set to `forward`, `backward`, or `bidirectional`, use that as the travel direction for that lane. Note that travelDir is a custom RoadRunner attribute that is unlikely to be found in OpenDRIVE files from 3rd party vendors.

2. If an ASAM OpenDRIVE road has its `rule` attribute set to RHT (right-handed traffic) or LHT(left-handed traffic), set the travel direction of the lanes appropriately for that traffic rule (e.g. RHT would indicate left lanes backward, right lanes forward).

3. Otherwise, set the lane direction to `undirected`.

Limitations

General

- If no `zOffset` is defined for a `<signal>` element in ASAM OpenDRIVE, RoadRunner places the imported sign or signal directly on the road surface.

- RoadRunner road plan geometry requires that segments (line, arc, clothoid, or parametric cubic) be continuous. During import, RoadRunner might define some ASAM OpenDRIVE data as a series of `<line>` segments that are mapped to a continuous curve.

- During import, RoadRunner converts the ASAM OpenDRIVE geometry type `<poly3>` to `<paramPoly3>`.

- RoadRunner roads cannot connect to themselves. Import ignores ASAM OpenDRIVE connections that connect a road to itself.

Specific to OpenDRIVE 1.5 / ASAM OpenDRIVE 1.6

- RoadRunner does not import the virtual junctions and virtual connections specified in ASAM OpenDRIVE.

- If segments of serially connected roads are overlapping in ASAM OpenDRIVE, then RoadRunner ignores the overlapping segments of the roads and imports the roads as disjoint roads.

- RoadRunner ignores the lateral offset `<sway>` specified for `roadMark` entries in ASAM OpenDRIVE. The imported road markings follow the lane border as if no `<sway>` is specified.

- RoadRunner ignores the object markings and object borders described in ASAM OpenDRIVE, and does not import the `<markings>` and `<border>` elements specified within an `<object>` element.

- RoadRunner does not import the `id`, `fillType`, `outer`, and `laneType` attributes for an object outline described in ASAM OpenDRIVE.

- If there are multiple `<outline>` entries for an `<object>` element, RoadRunner exports only the last `<outline>` entry and ignores the rest.

See Also
OpenDRIVE Viewer Tool | OpenDRIVE Export Preview Tool

More About

Decompress LAZ Files

RoadRunner software does not support some LAZ files, resulting in this error: `"The LAZ schema is not recognized"`. To resolve this issue, you can decompress the LAZ file into an LAS file.

Decompression Process

1 To get started, on Windows, get the latest version of LASzip (found here). To get started on Linux, build an executable for your operating system.

2 For both operating systems, run the LASzip executable.

> **Tip** LASzip can sometimes have strange behavior when clicking or selecting in the interface. To fix this issue, try maximizing the LASzip window or increasing the window size.

3 Click **browse**.
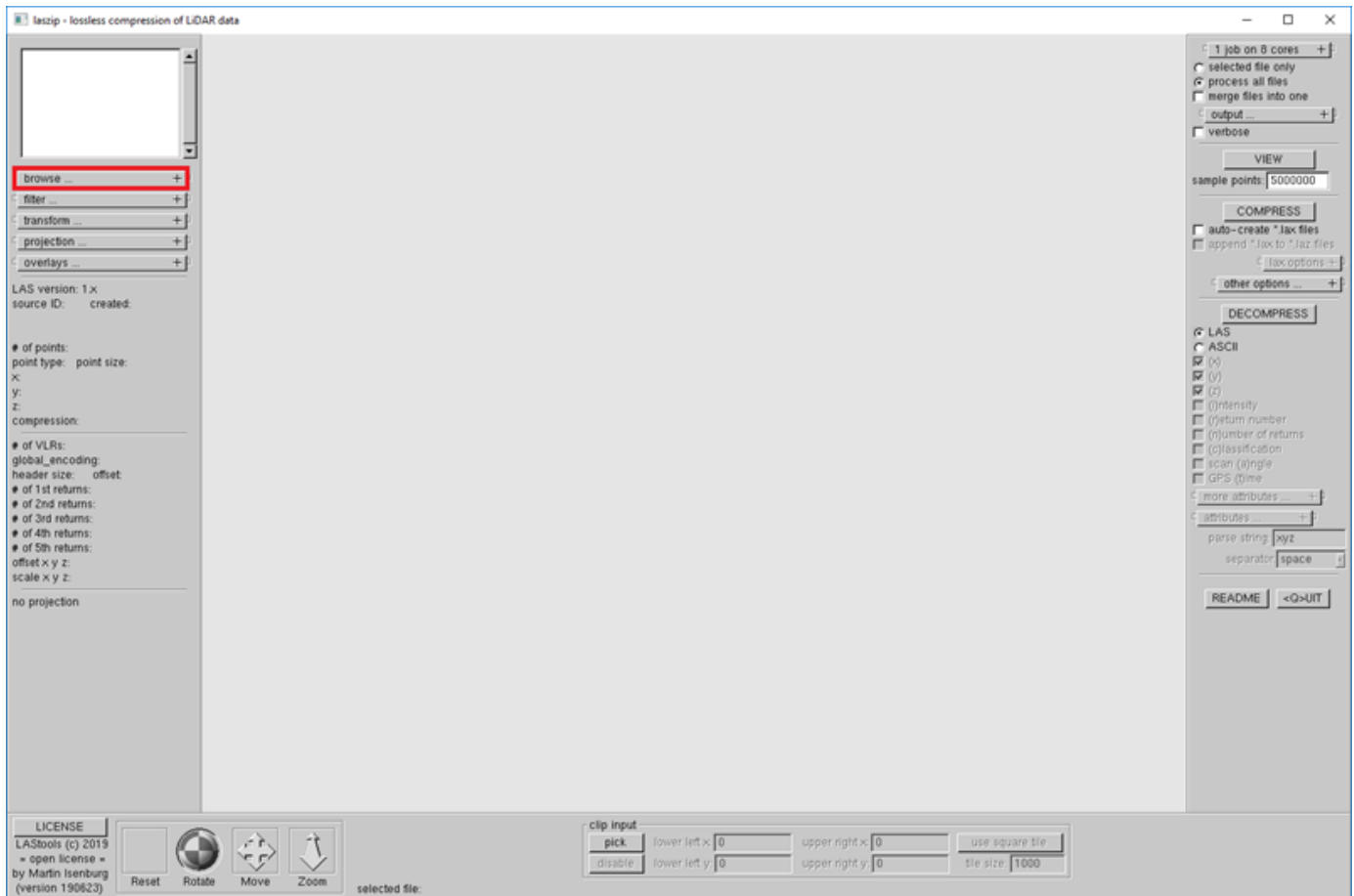
4 Find the desired LAZ file on your system. You can go to a specific directory using the **directory** field and clicking **go**.

> **Tip** To add multiple LAZ files, you can use the **wildcard** field to specify which types of files to add. Then click **add**. This action adds all the files fitting that wildcard in the currently browsed directory.
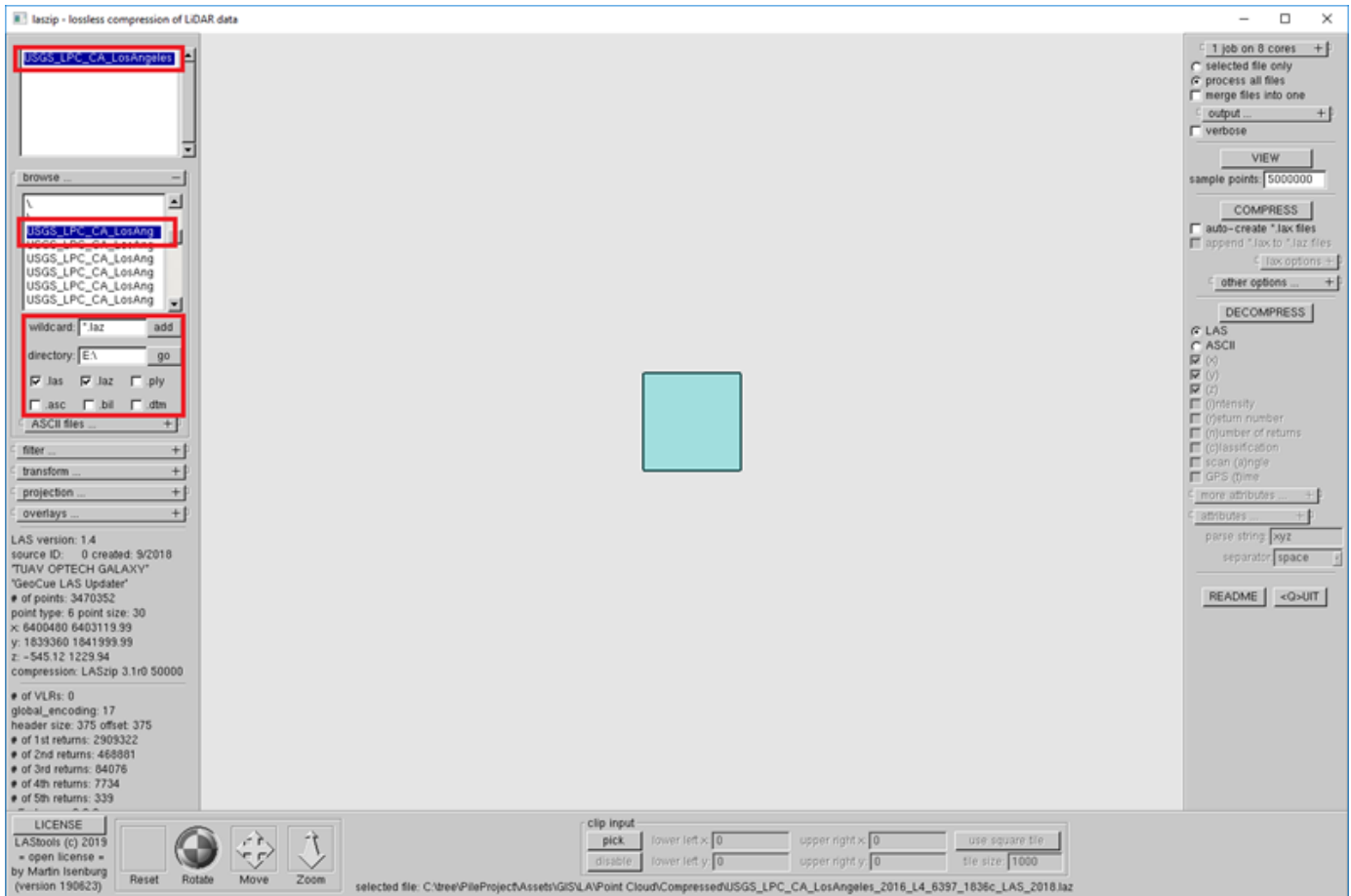
6 Adjust settings for output and processor usage in the upper-right corner of the window. Then click **Decompress**.
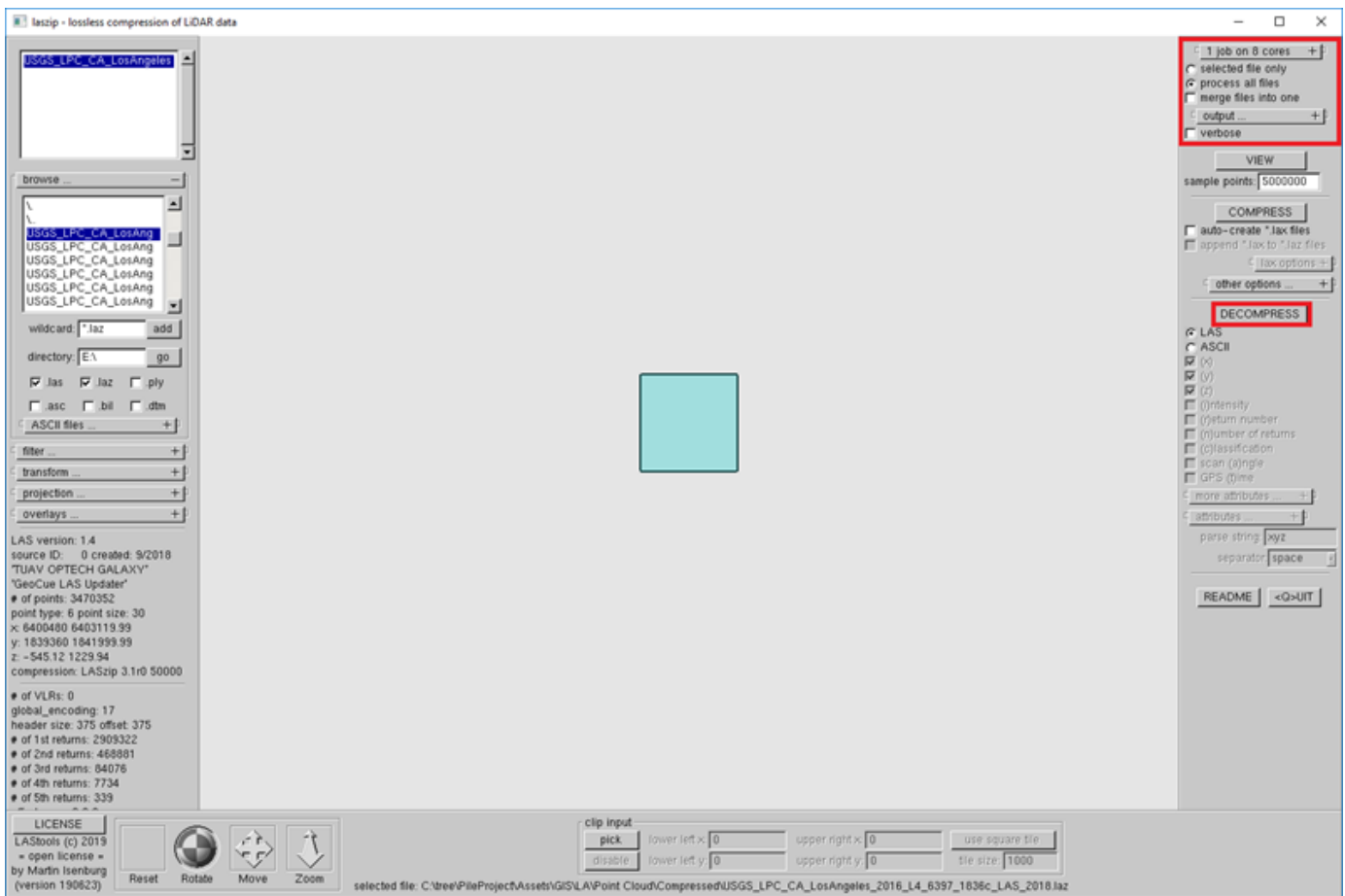
7 Click **Start** to run the decompression.
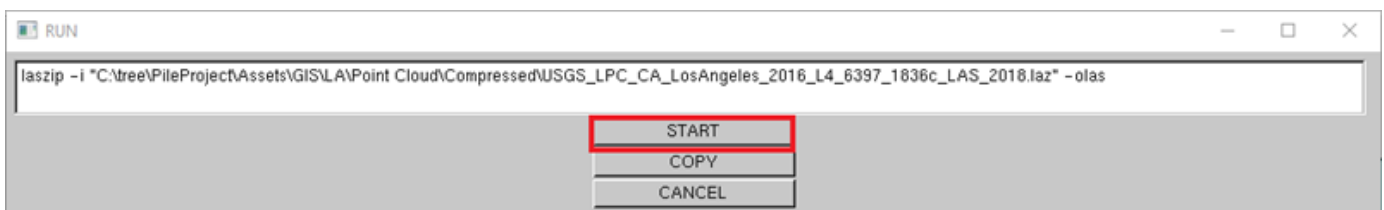
8 Once the RUN window closes, the decompression is complete. Assuming default settings, you can find your LAS files beside your LAZ files in the same directory.

Download GIS Data for Use in RoadRunner

When designing a scene based on a real-world location, you can use imported geographic information system (GIS) data as a visual reference. The process for importing the data into RoadRunner is the same as importing any other data, regardless of where the data was obtained.

RoadRunner supports several different formats. For lists of supported formats for each type of GIS data, refer to the GIS asset documentation:

- **Elevation Map Assets**
- **Aerial Image Assets**
- **Vector Data Assets**
- **Point Cloud Assets**

The most common file formats are:

- Raster data (satellite imagery and elevation), such as GeoTIFF and JPEG 2000
- Point cloud data, such as lidar data in LAZ and LAS formats

Choose USGS Interface for Downloading GIS Data

The U.S. Geological Survey (USGS) provides freely available GIS data for much of the United States. Coverage and quality varies depending on the data type and location.

This table shows the USGS interfaces from which you can access GIS data.

Interface Link	Interface Name
`https://apps.nationalmap.gov/downloader/#/`	The National Map (preferred)
`https://apps.nationalmap.gov/viewer/`	The National Map - Advanced Viewer
`https://earthexplorer.usgs.gov/`	EarthExplorer

Each USGS interface has a different user interface for selecting locations and data sets and can contain different data sets. When finding data for a specific project, best practice is to check all of the USGS interfaces.

Download GIS Data

This example shows how to download data from The National Map.

1 Find your area of interest by using the map interface.
2 In the **Datasets** pane, select one or more data sources. After you select a data source, you can click **Show Availability** to display the coverage of that data source. The table shows the recommended data sources to use for each type of GIS data.

Type of GIS Data	Recommended Data Source
Elevation data	**Elevation Products** One meter or 1/9 arc-second is recommended, but those options are not available in some locations.
Lidar data	**Elevation Source Data** Select the lidar point cloud (LPC) option.
Imagery data	**Imagery - NAIP Plus**

3 Click **Search Products** and, in the returned results, select the download link in a result to download that data.

4 Drag the downloaded data into the RoadRunner **Library Browser** to use as a visual reference when creating scenes. Some data might require downloading multiple tiles to cover your area of interest. For more details, see "Create Roads Around Imported GIS Assets" on page 1-57.

See Also

World Settings Tool | Aerial Imagery Tool | Elevation Map Tool | Point Cloud Tool | Vector Data Tool

Related Examples

- "Create Roads Around Imported GIS Assets" on page 1-57
- "Decompress LAZ Files" on page 3-5
- "Coordinate Space and Georeferencing" on page 2-10

Importing ASAM OpenCRG Files

ASAM OpenCRG is an open standard that enables you to specify road surface data using the curved regular grid (CRG) format. You can link CRG data with road network data specified using an ASAM OpenDRIVE file. Using RoadRunner, you can import data from an ASAM OpenCRG V1.2.0 file, assign it to a road segment in the scene, and visualize road surface variations using a colormap.

Note You cannot edit data within ASAM OpenCRG files using RoadRunner.

Import ASAM OpenCRG File

You can import an ASAM OpenCRG file with or without an ASAM OpenDRIVE file. When you import an ASAM OpenCRG file with an ASAM OpenDRIVE file, RoadRunner can build a scene by integrating CRG data with the road network data specified by the ASAM OpenDRIVE file. Alternatively, you can import an ASAM OpenCRG file independently and assign the CRG data to the desired road segment in a scene.

Import ASAM OpenCRG File with ASAM OpenDRIVE File

Follow these steps to build a scene and visualize CRG data by importing an ASAM OpenDRIVE file that references one or more ASAM OpenCRG files.

1   In RoadRunner, add the ASAM OpenDRIVE file and all linked ASAM OpenCRG files to a folder in the **Library Browser**.
2   Drag the ASAM OpenDRIVE file from the **Library Browser** into the scene. This action switches to the **OpenDRIVE Viewer Tool**.

3   In the 3D scene, click the imported ASAM OpenDRIVE file to select it.
4   On the toolbar to the left of the scene editing canvas, click the **Build Scene** button.

5   Select your desired options in the Import ASAM OpenDRIVE dialog box and click **Import**.
6   To visualize the CRG data, click the **Road CRG Tool** button, select the desired road, and select the desired road span. RoadRunner displays the CRG data using a colormap overlaid on the road. The **Attributes** pane shows the linked CRG file and the attributes of the CRG data, as specified in the imported ASAM OpenDRIVE file.

Import ASAM OpenCRG File Independently

Follow these steps to assign CRG data to a road segment in a 3D scene by importing an ASAM OpenCRG file.

1   In RoadRunner, add the ASAM OpenCRG file to a folder in the **Library Browser**.
2   Click the **Road CRG Tool** button on the toolbar.
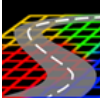
3 Select a road for which you want to define road surface data.

4 To define a span for road surface data, right-click the locations at which you want to insert span nodes.

5 Select the desired road span and drag the ASAM OpenCRG file from the **Library Browser** into the **CRG file** asset holder in the **Attributes** pane. The scene editing canvas shows the CRG data using a colormap overlaid on the road.

You can repeat these steps to assign CRG data from the same or a different ASAM OpenCRG file to other road segments in the scene.

See Also
Road CRG Tool | **Synthetic OpenCRG Assets** | **OpenDRIVE Viewer Tool**

More About
- "Export to ASAM OpenCRG" on page 5-48
- "Importing ASAM OpenDRIVE Files" on page 3-2
- "Export to ASAM OpenDRIVE" on page 5-24

External Websites
- ASAM OpenCRG

Build Roads by Using Zenrin Japan Map API 3.0 (Itsumo NAVI API 3.0) Data

This example shows how to build roads in RoadRunner by using Zenrin Japan Map API 3.0 (Itsumo NAVI API 3.0) [1] data for an area in Koto City, Tokyo, Japan that contains roads and few overpasses.

Choose Area of Interest

Specify the area of interest by using the **World Settings Tool**.

In the **Attributes** pane, in the **World Origin** section, specify the **Latitude** as 35.6380286 degrees and the **Longitude** as 139.7958767 degrees. In the **Workspace** section, under **Extents**, specify both **X** and **Y** as 500 meters. Apply your changes by selecting **Apply World Changes**.

Import and Explore Data

Open the **SD Map Viewer Tool** from the toolbar by clicking the **SD Map Viewer Tool** button.

Then, import the Zenrin Japan Map API 3.0 (Itsumo NAVI API 3.0) data by clicking the **Import Data For Area** button on the toolbar to the left of the scene editing canvas.

Before you import the data, you must enter valid Zenrin Japan Map API 3.0 (Itsumo NAVI API 3.0) access credentials in the Zenrin Map Credentials dialog box.
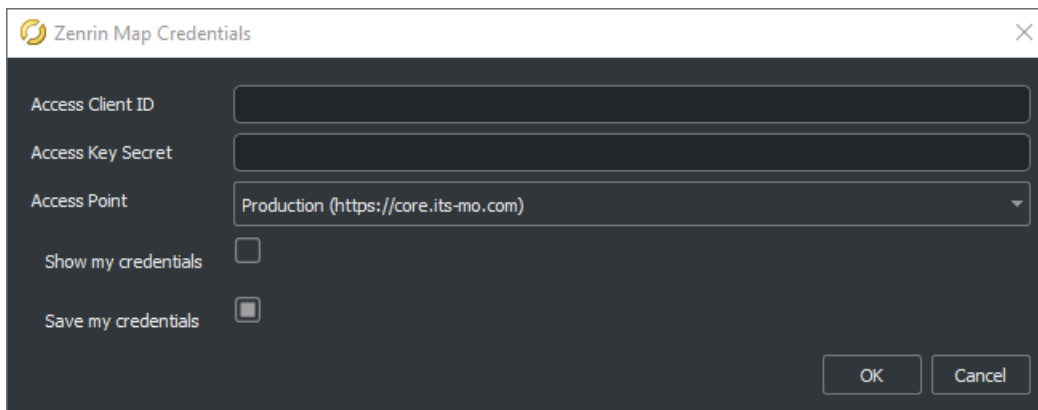
- **Access Client ID** — Specify your Zenrin client ID.
- **Access Key Secret** — Specify your Zenrin secret key.
- **Access Point** — Select the appropriate option for your Zenrin license:

  - Production (https://core.its-mo.com) — Use this option when you have a permanent license agreement to access Zenrin Japan Map API 3.0 (Itsumo NAVI API 3.0).
  - Verification (https://test.core.its-mo.com) — Use this option when you have an evaluation license agreement to access Zenrin Japan Map API 3.0 (Itsumo NAVI API 3.0).

---

1   To gain access to the Zenrin Japan Map API 3.0 (Itsumo NAVI API 3.0) service and get the required credentials (a client ID and secret key), you must enter into a separate agreement with ZENRIN DataCom CO., LTD.

- **Show my credentials** — Select this parameter to see the explicit values you enter in the **Access Client ID** and **Access Key Secret** fields. By default, this parameter is not selected and your input is hidden.

- **Save my credentials** — Select this parameter to save the specified credentials for future RoadRunner sessions. The credentials remain saved until you delete them. If you do not want to save the credentials for future sessions, clear this parameter. In this case, RoadRunner saves the credentials for only the rest of this RoadRunner session on your machine. By default, this parameter is not selected.

  You can delete your saved credentials by clearing the **Access Client ID** and **Access Key Secret** boxes and clicking **OK**. RoadRunner displays an error message indicating an issue with either your credentials or your connection to the download server. Ignore the error message, and click **Cancel** to close the dialog box.

Once you have specified your access credentials, click **OK**.

The **SD Map Viewer Tool** imports SD Map data that intersects your workspace, converts the data into a preview called an **SD Map**, and displays the **SD Map** in the scene editing canvas. The **SD Map** displays the nodes and links of the road data.

Explore the imported data by selecting links and nodes. You can view their properties on the **Attributes** pane. The type of road element selected in the SD Map scene editing canvas determines the available properties.

Simple Link

- **Id** — Unique identification number for the selected link.
- **Skip During Build** — Specifies whether to add or skip this link during the build process. If you select this attribute, the **SD Map** represents this link as a dashed line, and the link is ignored in the build process. To include the link in the build process, which displays it as a solid line, clear this attribute.

  The **SD Map Viewer Tool** imports the actual links with the **Skip During Build** attribute disabled, displaying them as solid lines.

  SD Map data can contain unsupported links for the same area. By default, the **SD Map Viewer Tool** imports these unsupported links with the **Skip During Build** attribute enabled, displaying them in the **SD Map** as dashed lines.. You can see a list of detected unsupported link IDs in the **Output** window. Select these IDs to navigate to and view the links in the **SD Map**.

  Note You can click and drag to select multiple links within a rectangular region of interest. You can also hold **Shift** and click additional links, to add them to the selection. You can control the **Skip During Build** attribute collectively, for all selected links in the **Attributes** pane.

- **Road Width (in meters)** — **Min** and **Max** road width.

- **Number of Lanes** — Number of forward and backward lanes. **Forward** and **Backward** attributes shows the **Min** and **Max** values if the map data specifies it.
- **Travel Direction** — Direction of travel for the road segment as forward, backward, or bidirectional.

Each link has several control points and each **Control Point** contains **Position** attribute specifying its (*X*,*Y*, *Z*) location.

Simple Node

- **Id** — Unique identification number for the selected node.
- **Connecting Links** — Displays all the links connected to the selected node. Each connected link is labelled with its associated ID and orientation.

For details on the programmatic use of these parameters while building roads, see "Road Width and Number of Lane Calculations".

Build Roads

You can build roads for the imported data using one or more of these processes.
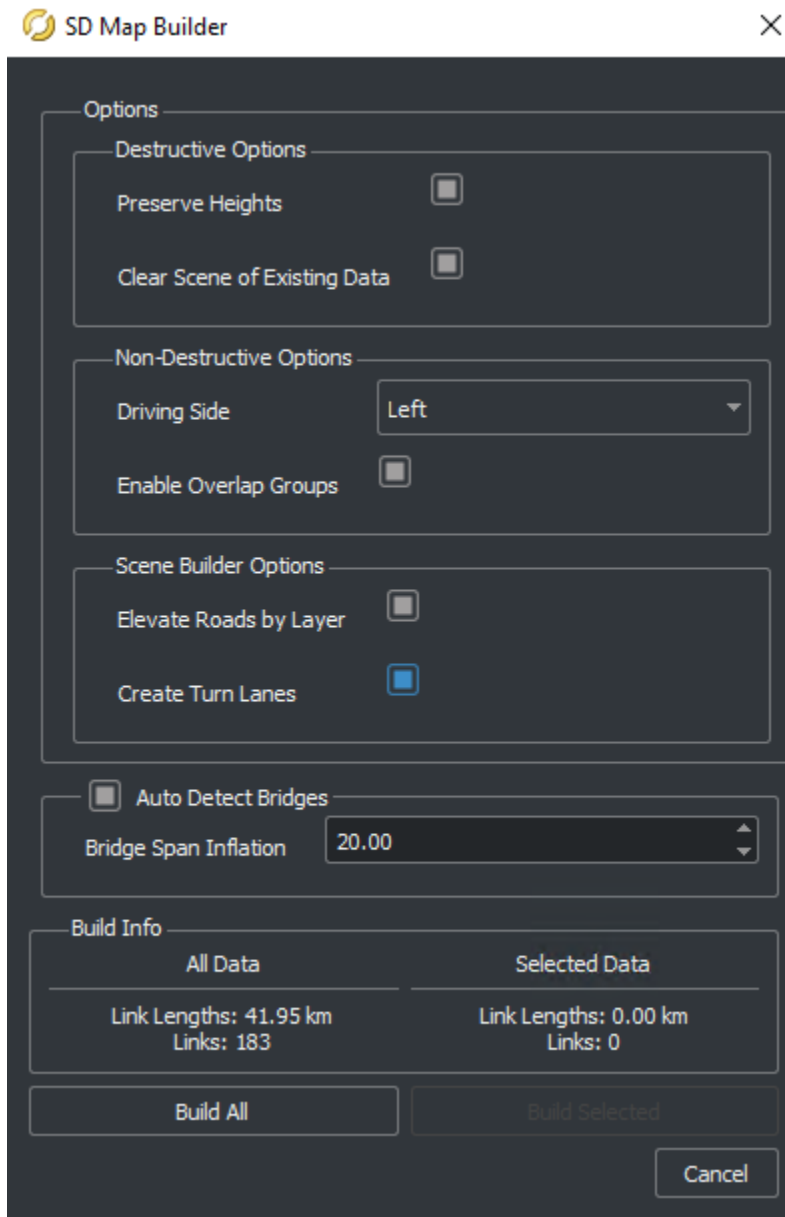
- All data — Build all of the imported data.
- Select links — Click and drag to select links within a rectangular region of interest.

You can also delete selected links to avoid building them.

For this example, do not select any of the links.

Then, click the **Build Roads** button on the toolbar to the left of the scene editing canvas.

In the Simple Map Builder dialog box, you can view and modify these options:

- **Preserve Heights** — By default, the **SD Map Viewer Tool** preserves the heights of the imported roads. To remove height information, clear this option.

- **Clear Scene of Existing Data** — By default, the **SD Map Viewer Tool** removes already built roads from your scene when you use it to build a scene. To keep the existing roads in the scene, clear this option.

- **Driving Side** — By default, **SD Map Viewer Tool** considers left side of the road as forward direction of driving. To consider right side of the road as forward direction of driving, select `Right` from the drop down list.

- **Enable Overlap Groups** — By default, the **SD Map Viewer Tool** enforces junction location information to create junctions. To create junctions using geometric overlaps, clear this option.

- **Auto Detect Bridges** — By default, the **SD Map Viewer Tool** creates bridges at road intersections when the roads have different elevations. The tool extends the bridges by 20 meters on either side of the intersection. You can change the amount of extension by changing the **Bridge Span Inflation** value. To prevent the tool from creating bridges, clear this option. For more information, see **Road Construction Tool**.

- **Build Info** — Displays the link length and number of links in all imported data, as well as in the selected subset of roads in the scene.

Note The **Enable Roads by Layer** and **Create Turn Lanes** options are not applicable. Setting or clearing these options, does not affect the roads build from Zenrin Japan Map API 3.0 (Itsumo NAVI API 3.0) data.

Build roads in the entire scene by clicking **Build All**. If you want to build only a subset of the roads in the scene, select the links to include in the scene and click **Build Selected**.

After you build roads, you can modify the scene in RoadRunner. You can also export the scene to ASAM OpenDRIVE file. For more information, see "Export to ASAM OpenDRIVE" on page 5-24.

If RoadRunner detects lane marking overlaps when building roads, then might display this message in the SD Map Builder Results dialog box:

```
>WARNING: Lane marking overlaps detected. Adjust road centers at these
locations
```
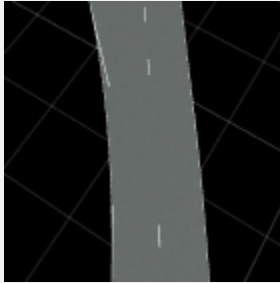
To resolve this issue, open the **Road Plan Tool**, click-navigate to the overlap locations, and the adjust road centers.
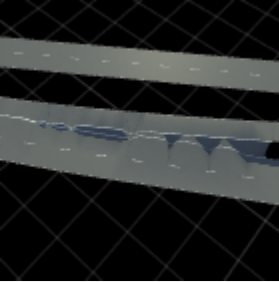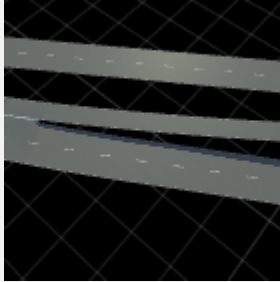
Troubleshoot Import and Build Issues

Depending on your area of interest, you might encounter issues when the **SD Map Viewer Tool** imports data and builds roads. Some issues might be due to missing or inaccurate SD Map data.

Gap

Issue	Solution
The built road contains gaps between roads at intersections. 	Open the Custom Junction Tool, navigate to the affected junction, and increase the ray **Distance** in the **Attributes** pane.

Steep Road Meshes

Issue	Solution
The built road contains steep rises or falls in the road meshes at road junctions. 	Open the Corner Tool, navigate to the affected junction, and reduce the **Corner Radius** in the **Attributes** pane.

Roads Under Terrain

Issue	Solution
The built scene contains roads under the terrain.	Open the Surface Tool, navigate to the affected location, and manually adjust the terrain.

See Also
SD Map Viewer Tool | **Custom Junction Tool** | **Corner Tool** | **Surface Tool**

More About
- "Export to ASAM OpenDRIVE" on page 5-24
- "Build Roads Using OpenStreetMap Data" on page 3-37
- "Importing ASAM OpenDRIVE Files" on page 3-2
- "Import ASAM OpenCRG File" on page 3-10

External Websites
- ZENRIN DataCom CO., LTD.

Import Custom Data Using RoadRunner HD Map

This example shows how to write C++ code to generate RoadRunner HD Map file to import custom data into RoadRunner.

Introduction

RoadRunner HD Map is a road data model for representing high-definition (HD) map data in a RoadRunner scene. This model defines a simple data structure to represent road layouts using lanes, lane boundaries, lane markings, and junctions. The model also defines data structure for other scene elements such as barriers, signs, and different types of static objects. The data in this model is serialized using protocol buffers and saved in a binary format with the `.rrhd` extension. The RoadRunner HD Map binary file consists of these data sections:

- Varint Header Size — Specifies the size of the header message. For more information, see Base 128 Varints in the Google Protocol Buffers guide.
- Header Message — Specifies information about the file author, map projection, and spatial bounds of the geometric data.
- RoadRunner HD Map Message — Specifies properties for lanes, lane boundaries, lane markings, and junctions.

You can create a RoadRunner HD Map binary file for your custom HD map data using protocol buffer schema files. These schema files describe the RoadRunner HD Map data model. Protocol buffers support various programming languages, so you can choose which language to use when implementing a RoadRunner HD Map. This example uses C++.

Compile Protocol Buffers for RoadRunner HD Map

Your local installation of RoadRunner contains the protocol buffer schema files at this location, referred to as *RoadRunnerProtoPath* in this example.

RRInstallFolder\bin*platform*\Proto

- *RRInstallFolder* is your local RoadRunner installation folder.
- *platform* is the folder name for your OS platform.

Inside the `Proto` folder, the top-level `mathworks` folder contains the hierarchy of protobuf files. This table shows the protobuf files that you need to compile and their locations.

Protobuf Files to Compile	Protobuf File Locations
hd_map_header.proto	*RoadRunnerProtoPath*\mathworks\scenario\scene\hd
hd_map.proto	
hd_lanes.proto	
hd_lane_markings.proto	
hd_junctions.proto	
hd_barriers.proto	
hd_signs.proto	

Protobuf Files to Compile	Protobuf File Locations
`hd_static_objects.proto`	
`common_attributes.proto`	
`geometry.proto`	*RoadRunnerProtoPath*\mathworks\scenario\common

To compile the protocol buffers using C++, follow these steps:

1 Download and install the protocol compiler `protoc` (minimum version 3.8.0) and the protobuf C++ runtime. Follow the instructions at `https://github.com/protocolbuffers/protobuf`.

The installation creates these folders at the install location:

- `bin` — Contains the `protoc` compiler.
- `include` — Contains additional header files and protobuf files required for compilation.
- `lib` — Contains linking libraries.

2 Run the compiler. Specify file paths for these folders:

- `include` — Folder created during previous step.
- *RoadRunnerProtoPath* — Folder that contains the RoadRunner HD Map protobuf files.
- *DestinationPath* — Destination directory for the generated code.
- Absolute file path to the source protobuf file.

For example, use this command to compile `hd_map_header.proto`, where *DestinationPath* specifies the destination directory.

```
protoc -I="..\include" -I="RoadRunnerProtoPath" --cpp_out="DestinationPath" ^
"RoadRunnerProtoPath\mathworks\scenario\scene\hd\hd_map_header.proto"
```

3 Verify that the compiler creates these files for the `.proto` file:

- Header file with the extension `.pb.h`.
- Implementation file with the extension `.pb.cc`.

4 Repeat steps 2 and 3 to compile the other protobuf files.

Prepare Custom Data

In practice, you must explore the data of the custom scene you are importing to determine its lane attributes. This example uses synthetic data to create a scene consisting of two longitudinally connected lane segments. Each lane has two lane boundaries. This table shows the attributes of these lanes and lane boundaries.

Attribute	Value for Lane 1	Value for Lane 2
Lane ID	`Lane1`	`Lane2`
Geometry	`[(0.782,-1.56)` `(50.78,23.43)]`	`[(50.78,23.43)` `(100.78,48.43)]`
Type	`Driving`	`Driving`
Travel Direction	`Forward`	`Forward`

Attribute	Value for Lane 1	Value for Lane 2
Predecessor ID	-	`Lane1`
Predecessor Alignment	-	`Forward`
Successor ID	`Lane2`	-
Successor Alignment	`Forward`	-
Left Lane Boundary ID	`LaneBoundary1`	`LaneBoundary4`
Left Lane Boundary Geometry	`[(0,0) (50,25)]`	`[(50,25) (100,50)]`
Right Lane Boundary ID	`LaneBoundary2`	`LaneBoundary3`
Right Lane Boundary Geometry	`[(1.565,-3.13)` `(51.565,21.864)]`	`[(51.565,21.864)` `(101.565,46.869)]`

The example also shows how to represent data for other scene elements such as barriers, signs, trees and buildings.

Create RoadRunner HD Map Binary File from Custom Data

Write C++ code to add custom data to the generated RoadRunner HD Map protocol buffer classes, after compilation.

C++ Code to Copy

```cpp
/*
Copyright 2021-2022 The MathWorks, Inc.
An example code for creating RoadRunner HD Map binary file.
*/
#include "mathworks/scenario/scene/hd/hd_map.pb.h"
#include "mathworks/scenario/scene/hd/hd_map_header.pb.h"
#include <google/protobuf/util/delimited_message_util.h>

#include <iostream>
#include <fstream>
#include <string>

using namespace std;
using namespace mathworks::scenario::scene::hdmap;

void WriteToRRHD(const string &filepath,
    const google::protobuf::MessageLite &headerMessage,
    const google::protobuf::MessageLite &HDMap);

int main(int argc, char **argv)
{
    HDMap myModel;

    // Add Lane1 to the Model
    Lane* lane1 = myModel.add_lanes();

    lane1->set_id("Lane1");
    auto lane1Geometry = lane1->mutable_geometry();

    // Start Point of Lane1
    auto startPoint_lane1 = lane1Geometry->add_values();
    startPoint_lane1->set_x(0.782);
    startPoint_lane1->set_y(-1.56);

    // End point of Lane1
    auto endPoint_lane1 = lane1Geometry->add_values();
    endPoint_lane1->set_x(50.78);
    endPoint_lane1->set_y(23.43);

    // Set type and direction for Lane1
    lane1->set_lane_type(LaneType::LANE_TYPE_DRIVING);
    lane1->set_travel_dir(TravelDir::TRAVEL_DIR_FORWARD);

    // Add Lane2 to the Model
    Lane* lane2 = myModel.add_lanes();

    lane2->set_id("Lane2");
    auto lane2Geometry = lane2->mutable_geometry();

    // Start Point of Lane2
    auto startPoint_lane2 = lane2Geometry->add_values();
    startPoint_lane2->set_x(50.78);
    startPoint_lane2->set_y(23.43);

    // End point of Lane2
    auto endPoint_lane2 = lane2Geometry->add_values();
    endPoint_lane2->set_x(100.78);
    endPoint_lane2->set_y(48.43);

    // Set type and direction for Lane2
    lane2->set_lane_type(LaneType::LANE_TYPE_DRIVING);
    lane2->set_travel_dir(TravelDir::TRAVEL_DIR_FORWARD);

    // Add connectivity information of Lane1 and Lane2
    auto predecessor = lane2->add_predecessors();
    auto predecessor_ref = predecessor->mutable_reference();
    predecessor_ref->set_id("Lane1");
    predecessor->set_alignment(Alignment::ALIGNMENT_FORWARD);

    auto successor = lane1->add_successors();
```

```
auto successor_ref = successor->mutable_reference();
successor_ref->set_id("Lane2");
successor->set_alignment(Alignment::ALIGNMENT_FORWARD);

// Add lane boundaries to the model
auto laneBoundary1 = myModel.add_lane_boundaries();
laneBoundary1->set_id("LaneBoundary1");
auto lb1Geometry = laneBoundary1->mutable_geometry();

// Define the start point of LaneBoundary1
auto startPoint_lb1 = lb1Geometry->add_values();
startPoint_lb1->set_x(0);
startPoint_lb1->set_y(0);

// Define the end point of LaneBoundary1
auto endPoint_lb1 = lb1Geometry->add_values();
endPoint_lb1->set_x(50);
endPoint_lb1->set_y(25);

auto laneBoundary2 = myModel.add_lane_boundaries();
laneBoundary2->set_id("LaneBoundary2");
auto lb2Geometry = laneBoundary2->mutable_geometry();

// Define the start point of LaneBoundary2
auto startPoint_lb2 = lb2Geometry->add_values();
startPoint_lb2->set_x(1.565);
startPoint_lb2->set_y(-3.13);

// Define the end point of LaneBoundary2
auto endPoint_lb2 = lb2Geometry->add_values();
endPoint_lb2->set_x(51.565);
endPoint_lb2->set_y(21.864);

auto laneBoundary3 = myModel.add_lane_boundaries();
laneBoundary3->set_id("LaneBoundary3");
auto lb3Geometry = laneBoundary3->mutable_geometry();

// Define the start point of LaneBoundary3
auto startPoint_lb3 = lb3Geometry->add_values();
startPoint_lb3->set_x(51.565);
startPoint_lb3->set_y(21.864);

// Define the end point of LaneBoundary3
auto endPoint_lb3 = lb3Geometry->add_values();
endPoint_lb3->set_x(101.565);
endPoint_lb3->set_y(46.869);

auto laneBoundary4 = myModel.add_lane_boundaries();
laneBoundary4->set_id("LaneBoundary4");
auto lb4Geometry = laneBoundary4->mutable_geometry();

// Define the start point of LaneBoundary4
auto startPoint_lb4 = lb4Geometry->add_values();
startPoint_lb4->set_x(50);
startPoint_lb4->set_y(25);

// Define the end point of LaneBoundary4
auto endPoint_lb4 = lb4Geometry->add_values();
endPoint_lb4->set_x(100);
endPoint_lb4->set_y(50);

// Connect lane boundaries to lane
auto referenceBoundary1 = lane1->mutable_left_lane_boundary();
auto referenceBoundary1_ref = referenceBoundary1->mutable_reference();
referenceBoundary1_ref->set_id("LaneBoundary1");
referenceBoundary1->set_alignment(Alignment::ALIGNMENT_FORWARD);

auto referenceBoundary2 = lane1->mutable_right_lane_boundary();
auto referenceBoundary2_ref = referenceBoundary2->mutable_reference();
referenceBoundary2_ref->set_id("LaneBoundary2");
referenceBoundary2->set_alignment(Alignment::ALIGNMENT_FORWARD);
```

```
auto referenceBoundary3 = lane2->mutable_right_lane_boundary();
auto referenceBoundary3_ref = referenceBoundary3->mutable_reference();
referenceBoundary3_ref->set_id("LaneBoundary3");
referenceBoundary3->set_alignment(Alignment::ALIGNMENT_FORWARD);

auto referenceBoundary4 = lane2->mutable_left_lane_boundary();
auto referenceBoundary4_ref = referenceBoundary4->mutable_reference();
referenceBoundary4_ref->set_id("LaneBoundary4");
referenceBoundary4->set_alignment(Alignment::ALIGNMENT_FORWARD);

// Add barrier type to the model
auto barrierType1 = myModel.add_barrier_types();
auto barrierTypeID = "BarrierType1";
barrierType1->set_id(barrierTypeID);
auto extrusionPath1 = barrierType1->mutable_extrusion_path();
extrusionPath1->set_asset_path("Assets/Extrusions/GuardRail.rrext");

// Add barrier to the model
auto barrier1 = myModel.add_barriers();
barrier1->set_id("Barrier1");
auto barrier1Geometry = barrier1->mutable_geometry();

// Define the geometry points of barrier
auto barrier1_point1 = barrier1Geometry->add_values();
barrier1_point1->set_x(1.565);
barrier1_point1->set_y(-3.13);
auto barrier1_point2 = barrier1Geometry->add_values();
barrier1_point2->set_x(51.565);
barrier1_point2->set_y(21.864);
auto barrier1_point3 = barrier1Geometry->add_values();
barrier1_point3->set_x(101.565);
barrier1_point3->set_y(46.869);
auto barrier1_ref = barrier1->mutable_barrier_type_ref();
barrier1_ref->set_id(barrierTypeID);
barrier1->flip_laterally();

// Add sign type to the model
auto signType1 = myModel.add_sign_types();
auto signTypeID = "SignType1";
signType1->set_id(signTypeID);
auto signType1_asset_path = signType1->mutable_asset_path();
signType1_asset_path->set_asset_path("Assets/Signs/US/Regulatory Signs/Sign_R2-1(30).svg");

// Add sign to the model
auto sign1 = myModel.add_signs();
sign1->set_id("Sign1");
auto sign1Geometry = sign1->mutable_geometry();
auto sign1center = sign1Geometry->mutable_center();
sign1center->set_x(8.62);
sign1center->set_y(-3.70);
sign1center->set_z(2);
auto sign1dimension = sign1Geometry->mutable_dimension();
sign1dimension->set_length(0);
sign1dimension->set_width(0.5);
sign1dimension->set_height(0.5);
auto sign1geoOrientation = sign1Geometry->mutable_geo_orientation();
auto sign1geoAngle = sign1geoOrientation->mutable_geo_angle();
sign1geoAngle->set_roll(0);
sign1geoAngle->set_pitch(0);
sign1geoAngle->set_heading(-3383);
auto sign1TypeRef = sign1->mutable_sign_type_ref();
sign1TypeRef->set_id(signTypeID);

// Add static objects to the model
// Define pole type of static object
auto staticObjectType1 = myModel.add_static_object_types();
auto staticObjectTypeID1 = "StaticObjectType1";
staticObjectType1->set_id(staticObjectTypeID1);
auto staticObjectType1_asset_path = staticObjectType1->mutable_asset_path();
staticObjectType1_asset_path->set_asset_path("Assets/Props/Signals/WoodPost_10ft.fbx");

// Add pole to the model
```

```cpp
        pair<double, double> pole = { 8.689,-3.693 };
        auto so1 = myModel.add_static_objects();
        so1->set_id("Pole1");
        auto so1Geometry = so1->mutable_geometry();
        auto so1center = so1Geometry->mutable_center();
        so1center->set_x(pole.first);
        so1center->set_y(pole.second);
        so1center->set_z(1.4);
        auto so1dimension = so1Geometry->mutable_dimension();
        so1dimension->set_length(0.10 / 2.0);
        so1dimension->set_width(0.10/2.0);
        so1dimension->set_height(3.3/2.0);
        auto so1geoOrientation = so1Geometry->mutable_geo_orientation();
        auto so1geoAngle = so1geoOrientation->mutable_geo_angle();
        so1geoAngle->set_roll(0);
        so1geoAngle->set_pitch(0);
        so1geoAngle->set_heading(0);
        auto so1TypeRef = so1->mutable_object_type_ref();
        so1TypeRef->set_id(staticObjectTypeID1);

        // Define tree type of static object
        auto staticObjectType2 = myModel.add_static_object_types();
        auto staticObjectTypeID2 = "StaticObjectType2";
        staticObjectType2->set_id(staticObjectTypeID2);
        auto staticObjectType2_asset_path = staticObjectType2->mutable_asset_path();
        staticObjectType2_asset_path->set_asset_path("Assets/Props/Trees/Eucalyptus_Sm01.fbx");

        // Add trees to the model
        double treePositions[15][2] = {
            { 20.6743,2.2571 },
            { 32.7286,8.2143 },
            { 44.7829,14.1714 },
            { 56.8371,20.1286 },
            { 68.8914,26.0857 },
            { 80.9457,32.0429 },
            { 93,38 },
            { 87,48 },
            { 5,8 },
            { 16.7143,13.7857 },
            { 28.4286,19.5714 },
            { 40.1429,25.3571 },
            { 51.8571,31.1429 },
            { 63.5714,36.9286 },
            { 75.2857,42.7143 }
        };

        for (int i = 0; i < 15; i++)
        {
            auto so2 = myModel.add_static_objects();
            so2->set_id("Tree" + std::to_string(i));
            auto so2Geometry = so2->mutable_geometry();
            auto so2center = so2Geometry->mutable_center();
            so2center->set_x(treePositions[i][0]);
            so2center->set_y(treePositions[i][1]);
            so2center->set_z(1.4);
            auto so2dimension = so2Geometry->mutable_dimension();
            so2dimension->set_length(1);
            so2dimension->set_width(1);
            so2dimension->set_height(1.5);
            auto so2geoOrientation = so2Geometry->mutable_geo_orientation();
            auto so2geoAngle = so2geoOrientation->mutable_geo_angle();
            so2geoAngle->set_roll(0);
            so2geoAngle->set_pitch(0);
            so2geoAngle->set_heading(0);
            auto so2TypeRef = so2->mutable_object_type_ref();
            so2TypeRef->set_id(staticObjectTypeID2);
        }

        // Define building type of static object
        auto staticObjectType3 = myModel.add_static_object_types();
        auto staticObjectTypeID3 = "StaticObjectType3";
        staticObjectType3->set_id(staticObjectTypeID3);
```

```
    auto staticObjectType3_asset_path = staticObjectType3->mutable_asset_path();
    staticObjectType3_asset_path->set_asset_path("Assets/Buildings/Downtown_30mX30m_6storey.fbx");

    // Add building to the model
    pair<double, double> building = { 31.670,50.01 };
    auto so3 = myModel.add_static_objects();
    so3->set_id("Building1");
    auto so3Geometry = so3->mutable_geometry();
    auto so3center = so3Geometry->mutable_center();
    so3center->set_x(building.first);
    so3center->set_y(building.second);
    so3center->set_z(13.3915);
    auto so3dimension = so3Geometry->mutable_dimension();
    so3dimension->set_length(31.5804 / 2.0);
    so3dimension->set_width(30.6722 / 2.0);
    so3dimension->set_height(38.7831 / 2.0);
    auto so3geoOrientation = so3Geometry->mutable_geo_orientation();
    auto so3geoAngle = so3geoOrientation->mutable_geo_angle();
    so3geoAngle->set_roll(0);
    so3geoAngle->set_pitch(0);
    so3geoAngle->set_heading(0.4697);
    auto so3TypeRef = so3->mutable_object_type_ref();
    so3TypeRef->set_id(staticObjectTypeID3);

    Header headerMessage;

    // Set the author of the file
    headerMessage.set_author("Author Name");

    // Set the projection of the file
    auto proj = headerMessage.mutable_projection();
    proj->set_projection("+proj=tmerc +datum=WGS84");

    auto geoBounds = headerMessage.mutable_geographic_boundary();
    auto bounds = geoBounds->mutable_bounds();

    // Set the maximum bounds
    auto boundsMax = bounds->mutable_max();
    boundsMax->set_x(107.57);
    boundsMax->set_y(119.04);
    boundsMax->set_z(0);

    // Set the minimum bounds
    auto boundsMin = bounds->mutable_min();
    boundsMin->set_x(-3.700);
    boundsMin->set_y(-18.830);
    boundsMin->set_z(0);


    // Specify the output file name
    string filepath = "example.rrhd";

    // Write the RRHD file to disk
    WriteToRRHD(filepath, headerMessage, myModel);

    return 0;
}

void WriteToRRHD(const string &filepath,
    const google::protobuf::MessageLite &headerMessage,
    const google::protobuf::MessageLite &HDMap)
{
    // Open the file as output binary
    fstream fileStream(filepath, ios::out | ios::binary);

    // Write the delimited header message to the buffer
    if (!google::protobuf::util::SerializeDelimitedToOstream(
        headerMessage, &fileStream))
        cerr << "Error writing the header message" << endl;

    // Write the RoadRunner HD Map message to buffer
    if (!HDMap.SerializeToOstream(&fileStream))
```

```
        cerr << "Error writing the RoadRunner HD Map message" << endl;
}
```

The table describes the code in this example.

Code and Description

Include the required header files. The first two header files are generated when you compile the protocol buffer files for RoadRunner HD Map. The third file is generated when you import protocol buffers into your project.

Include standard C++ libraries and use the `mathworks::scenario::scene::hdmap` name space for RoadRunner HD Map.

```
#include "mathworks/scenario/scene/hd/hd_map.pb.h"
#include "mathworks/scenario/scene/hd/hd_map_header.pb.h"
#include <google/protobuf/util/delimited_message_util.h>

#include <iostream>
#include <fstream>
#include <string>

using namespace std;
using namespace mathworks::scenario::scene::hdmap;
```

Declare the `WriteToRRHD` function that writes delimited, serialized data to the output binary file.

```
void WriteToRRHD(const string &filepath,
 const google::protobuf::MessageLite &headerMessage,
 const google::protobuf::MessageLite &HDMap);
```

Create an instance of RoadRunner HD Map.

```
int main(int argc, char **argv)
{
 HDMap myModel;
```

Add this information for two lanes to the model:

- Lane ID
- Coordinates defining lane geometry
- Lane type
- Driving direction

```
// Add Lane1 to the model
Lane* lane1 = myModel.add_lanes();

lane1->set_id("Lane1");
auto lane1Geometry = lane1->mutable_geometry();

// Start Point of Lane1
auto startPoint_lane1 = lane1Geometry->add_values();
startPoint_lane1->set_x(0.782);
startPoint_lane1->set_y(-1.56);

// End point of Lane1
auto endPoint_lane1 = lane1Geometry->add_values();
endPoint_lane1->set_x(50.78);
endPoint_lane1->set_y(23.43);

// Set type and direction for Lane1
lane1->set_lane_type(LaneType::LANE_TYPE_DRIVING);
lane1->set_travel_dir(TravelDir::TRAVEL_DIR_FORWARD);

// Add Lane2 to the Model
Lane* lane2 = myModel.add_lanes();

lane2->set_id("Lane2");
auto lane2Geometry = lane2->mutable_geometry();

// Start Point of Lane2
```

Code and Description

```
auto startPoint_lane2 = lane2Geometry->add_values();
startPoint_lane2->set_x(50.78);
startPoint_lane2->set_y(23.43);

// End point of Lane2
auto endPoint_lane2 = lane2Geometry->add_values();
endPoint_lane2->set_x(100.78);
endPoint_lane2->set_y(48.43);

// Set type and direction for Lane2
lane2->set_lane_type(LaneType::LANE_TYPE_DRIVING);
lane2->set_travel_dir(TravelDir::TRAVEL_DIR_FORWARD);
```

Specify alignment between the lanes by defining information about their predecessor and successor relationship.

```
// Add connectivity information of Lane1 and Lane2
auto predecessor = lane2->add_predecessors();
auto predecessor_ref = predecessor->mutable_reference();
predecessor_ref->set_id("Lane1");
predecessor->set_alignment(Alignment::ALIGNMENT_FORWARD);

auto successor = lane1->add_successors();
auto successor_ref = successor->mutable_reference();
successor_ref->set_id("Lane2");
successor->set_alignment(Alignment::ALIGNMENT_FORWARD);
```

Add four lane boundaries to the model and define the geometry for each lane boundary.

```
// Add lane boundaries to the model
auto laneBoundary1 = myModel.add_lane_boundaries();
laneBoundary1->set_id("LaneBoundary1");
auto lb1Geometry = laneBoundary1->mutable_geometry();

// Define the start point of LaneBoundary1
auto startPoint_lb1 = lb1Geometry->add_values();
startPoint_lb1->set_x(0);
startPoint_lb1->set_y(0);

// Define the end point of LaneBoundary1
auto endPoint_lb1 = lb1Geometry->add_values();
endPoint_lb1->set_x(50);
endPoint_lb1->set_y(25);

auto laneBoundary2 = myModel.add_lane_boundaries();
laneBoundary2->set_id("LaneBoundary2");
auto lb2Geometry = laneBoundary2->mutable_geometry();

// Define the start point of LaneBoundary2
auto startPoint_lb2 = lb2Geometry->add_values();
startPoint_lb2->set_x(1.565);
startPoint_lb2->set_y(-3.13);

// Define the end point of LaneBoundary2
auto endPoint_lb2 = lb2Geometry->add_values();
endPoint_lb2->set_x(51.565);
endPoint_lb2->set_y(21.864);

auto laneBoundary3 = myModel.add_lane_boundaries();
laneBoundary3->set_id("LaneBoundary3");
auto lb3Geometry = laneBoundary3->mutable_geometry();

// Define the start point of LaneBoundary3
auto startPoint_lb3 = lb3Geometry->add_values();
startPoint_lb3->set_x(51.565);
startPoint_lb3->set_y(21.864);

// Define the end point of LaneBoundary3
auto endPoint_lb3 = lb3Geometry->add_values();
```

Code and Description

```
endPoint_lb3->set_x(101.565);
endPoint_lb3->set_y(46.869);

auto laneBoundary4 = myModel.add_lane_boundaries();
laneBoundary4->set_id("LaneBoundary4");
auto lb4Geometry = laneBoundary4->mutable_geometry();

// Define the start point of LaneBoundary4
auto startPoint_lb4 = lb4Geometry->add_values();
startPoint_lb4->set_x(50);
startPoint_lb4->set_y(25);

// Define the end point of LaneBoundary4
auto endPoint_lb4 = lb4Geometry->add_values();
endPoint_lb4->set_x(100);
endPoint_lb4->set_y(50);
```

Link the lane boundaries to the lanes. Define the left and right lane boundaries for each lane, and specify alignment between lanes and lane boundaries.

```
// Connect lane boundaries to lane
auto referenceBoundary1 = lane1->mutable_left_lane_boundary();
auto referenceBoundary1_ref = referenceBoundary1->mutable_reference();
referenceBoundary1_ref->set_id("LaneBoundary1");
referenceBoundary1->set_alignment(Alignment::ALIGNMENT_FORWARD);

auto referenceBoundary2 = lane1->mutable_right_lane_boundary();
auto referenceBoundary2_ref = referenceBoundary2->mutable_reference();
referenceBoundary2_ref->set_id("LaneBoundary2");
referenceBoundary2->set_alignment(Alignment::ALIGNMENT_FORWARD);

auto referenceBoundary3 = lane2->mutable_right_lane_boundary();
auto referenceBoundary3_ref = referenceBoundary3->mutable_reference();
referenceBoundary3_ref->set_id("LaneBoundary3");
referenceBoundary3->set_alignment(Alignment::ALIGNMENT_FORWARD);

auto referenceBoundary4 = lane2->mutable_left_lane_boundary();
auto referenceBoundary4_ref = referenceBoundary4->mutable_reference();
referenceBoundary4_ref->set_id("LaneBoundary4");
referenceBoundary4->set_alignment(Alignment::ALIGNMENT_FORWARD);
```

Specify type of barrier, add barrier to the model and define the properties of the barrier.

```
// Add barrier type to the model
auto barrierType1 = myModel.add_barrier_types();
auto barrierTypeID = "BarrierType1";
barrierType1->set_id(barrierTypeID);
auto extrusionPath1 = barrierType1->mutable_extrusion_path();
extrusionPath1->set_asset_path("Assets/Extrusions/GuardRail.rrext");

// Add barrier to the model
auto barrier1 = myModel.add_barriers();
barrier1->set_id("Barrier1");
auto barrier1Geometry = barrier1->mutable_geometry();

// Define the geometry points of barrier
auto barrier1_point1 = barrier1Geometry->add_values();
barrier1_point1->set_x(1.565);
barrier1_point1->set_y(-3.13);
auto barrier1_point2 = barrier1Geometry->add_values();
barrier1_point2->set_x(51.565);
barrier1_point2->set_y(21.864);
auto barrier1_point3 = barrier1Geometry->add_values();
barrier1_point3->set_x(101.565);
barrier1_point3->set_y(46.869);
auto barrier1_ref = barrier1->mutable_barrier_type_ref();
barrier1_ref->set_id(barrierTypeID);
barrier1->flip_laterally();
```

Code and Description

Specify type of sign, add sign to the model and define the properties of the sign.

```
// Add sign type to the model
auto signType1 = myModel.add_sign_types();
auto signTypeID = "SignType1";
signType1->set_id(signTypeID);
auto signType1_asset_path = signType1->mutable_asset_path();
signType1_asset_path->set_asset_path("Assets/Signs/US/Regulatory Signs/Sign_R2-1(30).svg");

// Add sign to the model
auto sign1 = myModel.add_signs();
sign1->set_id("Sign1");
auto sign1Geometry = sign1->mutable_geometry();
auto sign1center = sign1Geometry->mutable_center();
sign1center->set_x(8.62);
sign1center->set_y(-3.70);
sign1center->set_z(2);
auto sign1dimension = sign1Geometry->mutable_dimension();
sign1dimension->set_length(0);
sign1dimension->set_width(0.5);
sign1dimension->set_height(0.5);
auto sign1geoOrientation = sign1Geometry->mutable_geo_orientation();
auto sign1geoAngle = sign1geoOrientation->mutable_geo_angle();
sign1geoAngle->set_roll(0);
sign1geoAngle->set_pitch(0);
sign1geoAngle->set_heading(-3383);
auto sign1TypeRef = sign1->mutable_sign_type_ref();
sign1TypeRef->set_id(signTypeID);
```

Specify pole type of static object, add pole to the model, and define the properties of the pole.

```
// Add static objects to the model
// Define pole type of static object
auto staticObjectType1 = myModel.add_static_object_types();
auto staticObjectTypeID1 = "StaticObjectType1";
staticObjectType1->set_id(staticObjectTypeID1);
auto staticObjectType1_asset_path = staticObjectType1->mutable_asset_path();
staticObjectType1_asset_path->set_asset_path("Assets/Props/Signals/WoodPost_10ft.fbx");

// Add pole to the model
pair<double, double> pole = { 8.689,-3.693 };
auto so1 = myModel.add_static_objects();
so1->set_id("Pole1");
auto so1Geometry = so1->mutable_geometry();
auto so1center = so1Geometry->mutable_center();
so1center->set_x(pole.first);
so1center->set_y(pole.second);
so1center->set_z(1.4);
auto so1dimension = so1Geometry->mutable_dimension();
so1dimension->set_length(0.10 / 2.0);
so1dimension->set_width(0.10/2.0);
so1dimension->set_height(3.3/2.0);
auto so1geoOrientation = so1Geometry->mutable_geo_orientation();
auto so1geoAngle = so1geoOrientation->mutable_geo_angle();
so1geoAngle->set_roll(0);
so1geoAngle->set_pitch(0);
so1geoAngle->set_heading(0);
auto so1TypeRef = so1->mutable_object_type_ref();
so1TypeRef->set_id(staticObjectTypeID1);
```

Specify tree type of static object, add trees to the model and define the properties of the trees.

```
// Define tree type of static object
auto staticObjectType2 = myModel.add_static_object_types();
auto staticObjectTypeID2 = "StaticObjectType2";
staticObjectType2->set_id(staticObjectTypeID2);
auto staticObjectType2_asset_path = staticObjectType2->mutable_asset_path();
staticObjectType2_asset_path->set_asset_path("Assets/Props/Trees/Eucalyptus_Sm01.fbx");

// Add trees to the model
```

Code and Description

```
double treePositions[15][2] = {
    { 20.6743,2.2571 },
    { 32.7286,8.2143 },
    { 44.7829,14.1714 },
    { 56.8371,20.1286 },
    { 68.8914,26.0857 },
    { 80.9457,32.0429 },
    { 93,38 },
    { 87,48 },
    { 5,8 },
    { 16.7143,13.7857 },
    { 28.4286,19.5714 },
    { 40.1429,25.3571 },
    { 51.8571,31.1429 },
    { 63.5714,36.9286 },
    { 75.2857,42.7143 }
};

for (int i = 0; i < 15; i++)
{
    auto so2 = myModel.add_static_objects();
    so2->set_id("Tree" + std::to_string(i));
    auto so2Geometry = so2->mutable_geometry();
    auto so2center = so2Geometry->mutable_center();
    so2center->set_x(treePositions[i][0]);
    so2center->set_y(treePositions[i][1]);
    so2center->set_z(1.4);
    auto so2dimension = so2Geometry->mutable_dimension();
    so2dimension->set_length(1);
    so2dimension->set_width(1);
    so2dimension->set_height(1.5);
    auto so2geoOrientation = so2Geometry->mutable_geo_orientation();
    auto so2geoAngle = so2geoOrientation->mutable_geo_angle();
    so2geoAngle->set_roll(0);
    so2geoAngle->set_pitch(0);
    so2geoAngle->set_heading(0);
    auto so2TypeRef = so2->mutable_object_type_ref();
    so2TypeRef->set_id(staticObjectTypeID2);
}
```

Specify building type of static object, add building to the model and define the properties of the building.

```
// Define building type of static object
auto staticObjectType3 = myModel.add_static_object_types();
auto staticObjectTypeID3 = "StaticObjectType3";
staticObjectType3->set_id(staticObjectTypeID3);
auto staticObjectType3_asset_path = staticObjectType3->mutable_asset_path();
staticObjectType3_asset_path->set_asset_path("Assets/Buildings/Downtown_30mX30m_6storey.fbx");

// Add building to the model
pair<double, double> building = { 31.670,50.01 };
auto so3 = myModel.add_static_objects();
so3->set_id("Building1");
auto so3Geometry = so3->mutable_geometry();
auto so3center = so3Geometry->mutable_center();
so3center->set_x(building.first);
so3center->set_y(building.second);
so3center->set_z(13.3915);
auto so3dimension = so3Geometry->mutable_dimension();
so3dimension->set_length(31.5804 / 2.0);
so3dimension->set_width(30.6722 / 2.0);
so3dimension->set_height(38.7831 / 2.0);
auto so3geoOrientation = so3Geometry->mutable_geo_orientation();
auto so3geoAngle = so3geoOrientation->mutable_geo_angle();
so3geoAngle->set_roll(0);
so3geoAngle->set_pitch(0);
so3geoAngle->set_heading(0.4697);
```

Code and Description

```
auto so3TypeRef = so3->mutable_object_type_ref();
so3TypeRef->set_id(staticObjectTypeID3);
```

Define the header message by specifying metadata about the file author, projection of the file and spatial bounds of the geometric data.

```
Header headerMessage;

// Set the author of the file
headerMessage.set_author("Author Name");

// Set the projection of the file
auto proj = headerMessage.mutable_projection();
proj->set_projection("+proj=tmerc +datum=WGS84");

auto geoBounds = headerMessage.mutable_geographic_boundary();
auto bounds = geoBounds->mutable_bounds();

// Set the maximum bounds
auto boundsMax = bounds->mutable_max();
boundsMax->set_x(107.57);
boundsMax->set_y(119.04);
boundsMax->set_z(0);

// Set the minimum bounds
auto boundsMin = bounds->mutable_min();
boundsMin->set_x(-3.700);
boundsMin->set_y(-18.830);
boundsMin->set_z(0);
```

Define the file path, and then generate a RoadRunner HD Map binary file using the `WriteToRRHD` function. The function writes delimited serialized, data to the output binary file.

```
// Specify the output file name
string filepath = "example.rrhd";

// Write the RRHD file to disk
WriteToRRHD(filepath, headerMessage, myModel);

return 0;
}

void WriteToRRHD(const string &filepath,
 const google::protobuf::MessageLite &headerMessage,
 const google::protobuf::MessageLite &HDMap)
{
// Open the file as output binary
fstream fileStream(filepath, ios::out | ios::binary);

// Write the delimited header message to the buffer
if (!google::protobuf::util::SerializeDelimitedToOstream(
 headerMessage, &fileStream))
 cerr << "Error writing the header message" << endl;

// Write the RoadRunner HD Map message to buffer
if (!HDMap.SerializeToOstream(&fileStream))
 cerr << "Error writing the RoadRunner HD Map message" << endl;
}
```

Import HD Map File into RoadRunner

To import the RoadRunner HD Map binary file into RoadRunner, follow these steps:

1   In RoadRunner, add the HD Map file to a folder in the **Library Browser**.

2   Select the file in the **Library Browser** to see the attributes of the file in the **Attributes** pane.

3   If the **Attributes** pane does not show the projection data, select **Set Custom Projection**, specify the latitude and longitude of the (0,0) point in the file, and select **Use Transverse Mercator At**. Then click **OK**.

4   Drag the file from the **Library Browser** into the scene. This action switches to the **Scene Builder Tool**.

The scene editing canvas shows the RoadRunner HD Map of the scene. Verify the imported data by selecting control points, lanes, and lane boundaries from the **Attributes** pane.
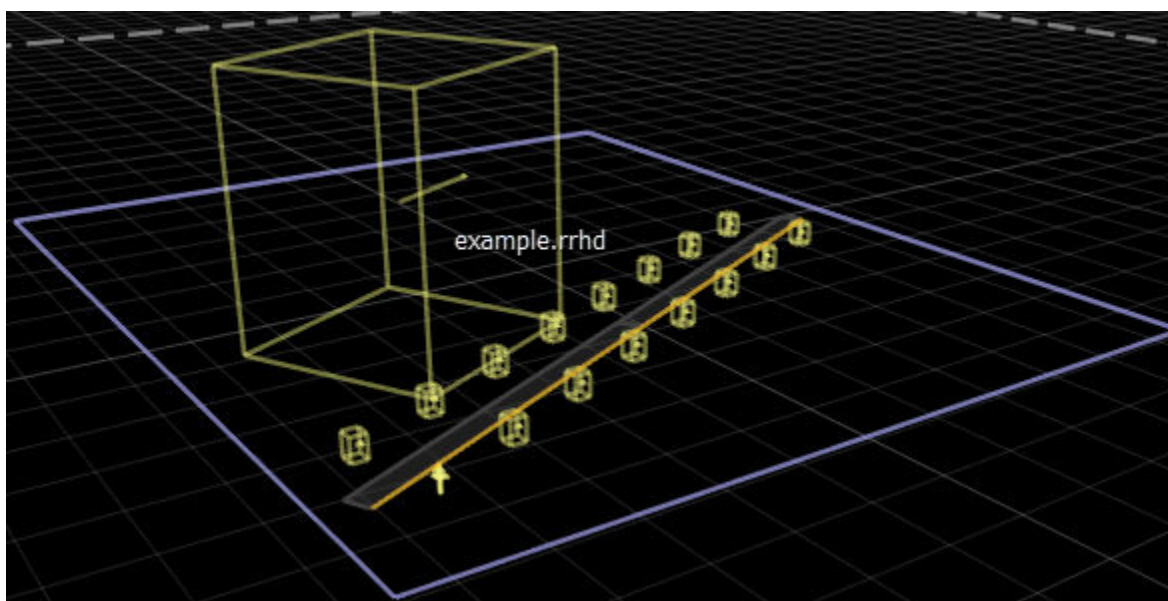
See Also

hd_map_header.proto | hd_map.proto | hd_lanes.proto | hd_lane_markings.proto | hd_junctions.proto | common_attributes.proto | geometry.proto

Related Examples

- "Compile Protocol Buffers for RoadRunner gRPC API" on page 6-19
- "Build Scenes from Custom Data Using RoadRunner HD Map"

External Websites

- Protocol Buffers

Build Roads Using OpenStreetMap Data

OpenStreetMap is a free, open-source, web map service that enables you to access crowd-sourced map data. Using RoadRunner, you can import and preview map data from an OpenStreetMap file and use it to build roads.

Import OpenStreetMap File

To import OpenStreetMap data, you must first select an OpenStreetMap file containing road geometry. To get these files, visit `openstreetmap.org`, specify a map location, manually adjust the region around this location, and export the road geometry for that region to an OpenStreetMap file with extension `.osm`. OpenStreetMap exports only the roads whose whole lengths are within the specified region. In this example, you use an OpenStreetMap file previously exported from the website.

1   Open the **SD Map Viewer Tool** from the toolbar by clicking the **SD Map Viewer Tool** button.

2   Click the **Open OpenStreetMap File** button on the toolbar to the left of scene editing canvas.

3   In the Open OpenStreetMap File dialog box, browse to this file, select it, and click **Open**.

RRInstallFolder/bin/*platform*/AssetsInstall/SampleFiles/city.osm

- *RRInstallFolder* is your local RoadRunner installation folder.
- *platform* is the folder name for your OS platform.

The file was downloaded from `https://www.openstreetmap.org`, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), `https://opendatacommons.org/licenses/odbl/`.

The **SD Map Viewer Tool** imports SD Map data that intersects your workspace, converts the data into a preview called an **SD Map**, and displays the **SD Map** in the scene editing canvas. The **SD Map** displays the nodes and links of the road data.
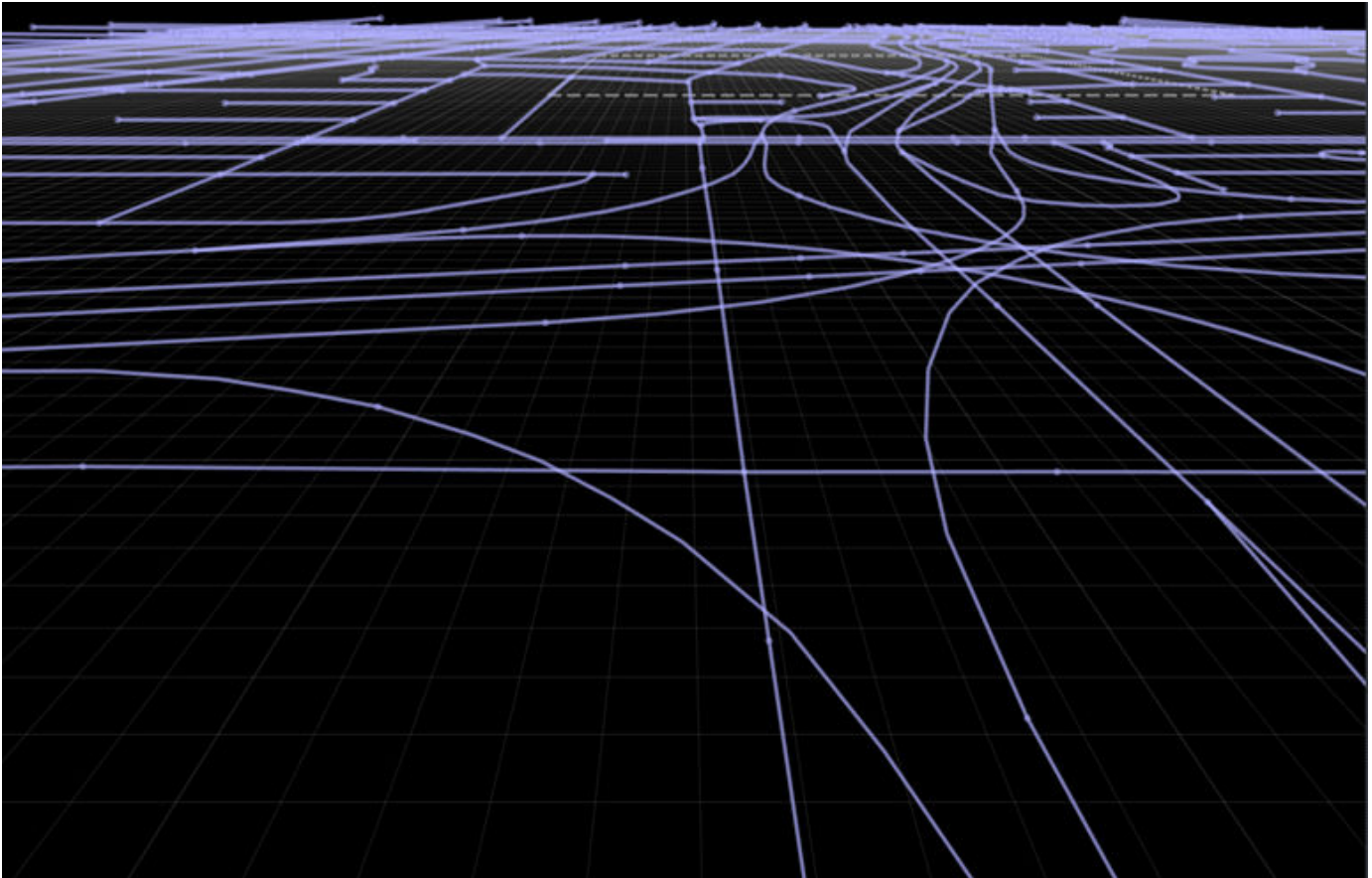
Note

- When you import an OpenStreetMap file into a new scene, the **SD Map Viewer Tool** automatically sets the world origin using the geographic bounds specified in the file. However, to successfully import an OpenStreetMap file into an existing scene that has an already specified **World Origin**, the geographic bounds specified in the file must approximately match the **World Origin** value of the existing scene.

- OpenStreetMap does not specify road junction information. When a road intersects another road, the **SD Map Viewer Tool** generates a separate link for each segment of the intersecting roads before and after the intersection. As a result, the number of generated links does not match the number of roads specified in the OpenStreetMap file.

Explore Imported Data

Explore the imported data by selecting links and nodes. You can view their attributes in the **Attributes** pane. The type of road element selected in the SD Map scene editing canvas determines the available attributes.

Simple Link

- **Id** — Unique identification number for the selected link.
- **LayerId** — Unique layer identification number for the selected link.
- **Skip During Build** — Specifies whether to add or skip this link during the build process. If you select this attribute, the **SD Map** represents this link as a dashed line, and the link is ignored in the build process. To include the link in the build process, which displays it as a solid line, clear this attribute.

  If you clear the **Skip During Build** attribute , the **SD Map Viewer Tool** imports the actual links and displays them as solid lines.

Note You can click and drag to select multiple links within a rectangular region of interest. You can also hold **Shift** and click additional links to add them to the selection. You can control the **Skip During Build** attribute collectively, for all selected links, in the **Attributes** pane.

- **Road Width (in meters)** — Width of the road corresponding to the selected link. Because OpenStreetMap does not specify the width of a lane or a road, the **SD Map Viewer Tool** sets the default lane width to 3.5 meters. Road width is the product of the lane width and the number of lanes within a road.
- **Number of Lanes** — Number of **Forward** and **Backward** lanes for the road corresponding to the selected link. If the input file does not specify the number of lanes for a road, the **SD Map Viewer Tool** estimates one lane for each travel direction. By default, one-way roads have one lane, and bidirectional roads have two total lanes, one for each travel direction.

- **Travel Direction** — Direction of travel for the road corresponding to the selected link, specified as `Forward`, `Backward`, or `Bidirectional`. If the input file does not specify the `oneway` tag for a road, the **SD Map Viewer Tool** assumes the road is bidirectional.

Each link has several control points and each **Control Point** contains a **Position** attribute specifying its (*X,Y,Z*) location.

Simple Node

- **Id**— Unique identification number for the selected node.
- **Connecting Links**— Displays all the links connected to the selected node. Each connected link is labelled with its associated ID and orientation.

Build Roads

You can build roads for the imported data using one of these processes.

- All data — Build all of the imported data.
- Select links — Click and drag to select links within a rectangular region of interest.

You can also delete selected links to avoid building them.

For this example, do not select any links. Click the **Build Roads** button on the toolbar to the left of the scene editing canvas to open the SD Map Builder dialog box.

In the SD Map Builder dialog box, you can view and modify these options:

Option	Description
Preserve Heights	OpenStreetMap files does not contain road elevation information. By default, this option is not applicable. **Dependencies** • This option is applicable only when the **Elevate Roads by Layer** option is enabled, and the imported OpenStreetMap file contains layer ID information.
Clear Scene of Existing Data	By default, the **SD Map Viewer Tool** removes already built roads from your scene when you use it to build a scene. To keep the existing roads in the scene, clear this option.
Driving Side	By default, **SD Map Viewer Tool** considers left side of the road as forward direction of driving. To consider right side of the road as forward direction of driving, select `Right` from the drop down list.
Enable Overlap Groups	By default, the **SD Map Viewer Tool** enforces junction location information to create junctions. To create junctions using geometric overlaps, clear this option.
Elevate Roads by Layer	OpenStreetMap files does not contain road elevation information and the imported files may contain overlap roads. To elevate bridges using layer information, enable this option. To get better build results of elevated bridges, enable the **Preserve Heights** and **Auto Detect Bridges** options, as well. **Dependencies** • To access this option, you must have a RoadRunner Scene Builder license. • For SD Map Builder to elevate the roads, the imported OpenStreetMap file must contain layer information.

Option	Description
Create Turn Lanes	Turn lane indicates the forward or backward direction of the road in a bi-directional way. By default, imported roads do not contain turn lane markings. To include turn lane markings using turn lane information, enable this option. **Dependencies** • To access this option, you must have a RoadRunner Scene Builder license. • For RoadRunner to include turn lane markings in its built results, the imported OpenStreetMap file must contain turn lane information.
Auto Detect Bridges	If you select this option, the **SD Map Viewer Tool** creates bridges at road intersections when the roads have different elevations. By default, the tool extends the bridges by 20 meters on either side of the intersection. You can change the length of the extension by changing the **Bridge Span Inflation** value. To prevent the tool from creating bridges, clear this option. For more information, see **Road Construction Tool**. **Dependencies** This option is applicable only when the **Elevate Roads by Layer** option is enabled, and the imported OpenStreetMap file contains layer information.
Build Info	Displays the link length and number of links in all the imported data, as well as in the selected subset of roads in the scene.

To build all roads in the scene, click **Build All**. If you want to build only a subset of the roads in the scene, select the links you want to include in the scene, and click **Build Selected**.

For this example, use the default options to build the scene.
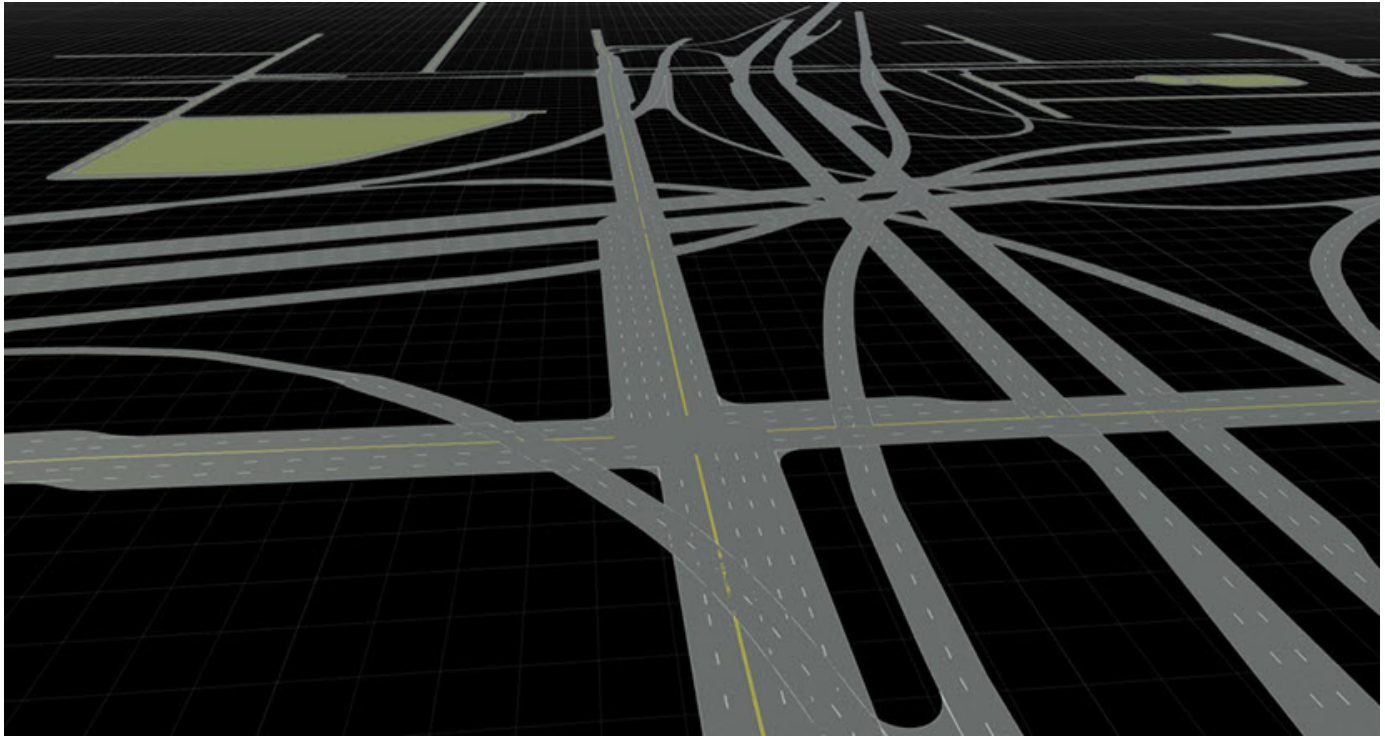
You can also build roads by enabling the **Elevate Roads by Layer** and the **Create Turn Lanes** options. Note that the `city.osm` file contains the layer and turn lane information.

Note To access the **Elevate Roads by Layer** and **Create Turn Lanes** options, you must have a RoadRunner Scene Builder license.

You can visualize the elevated bridges and overpasses at roads and junctions with turn lane markings in the built scene.
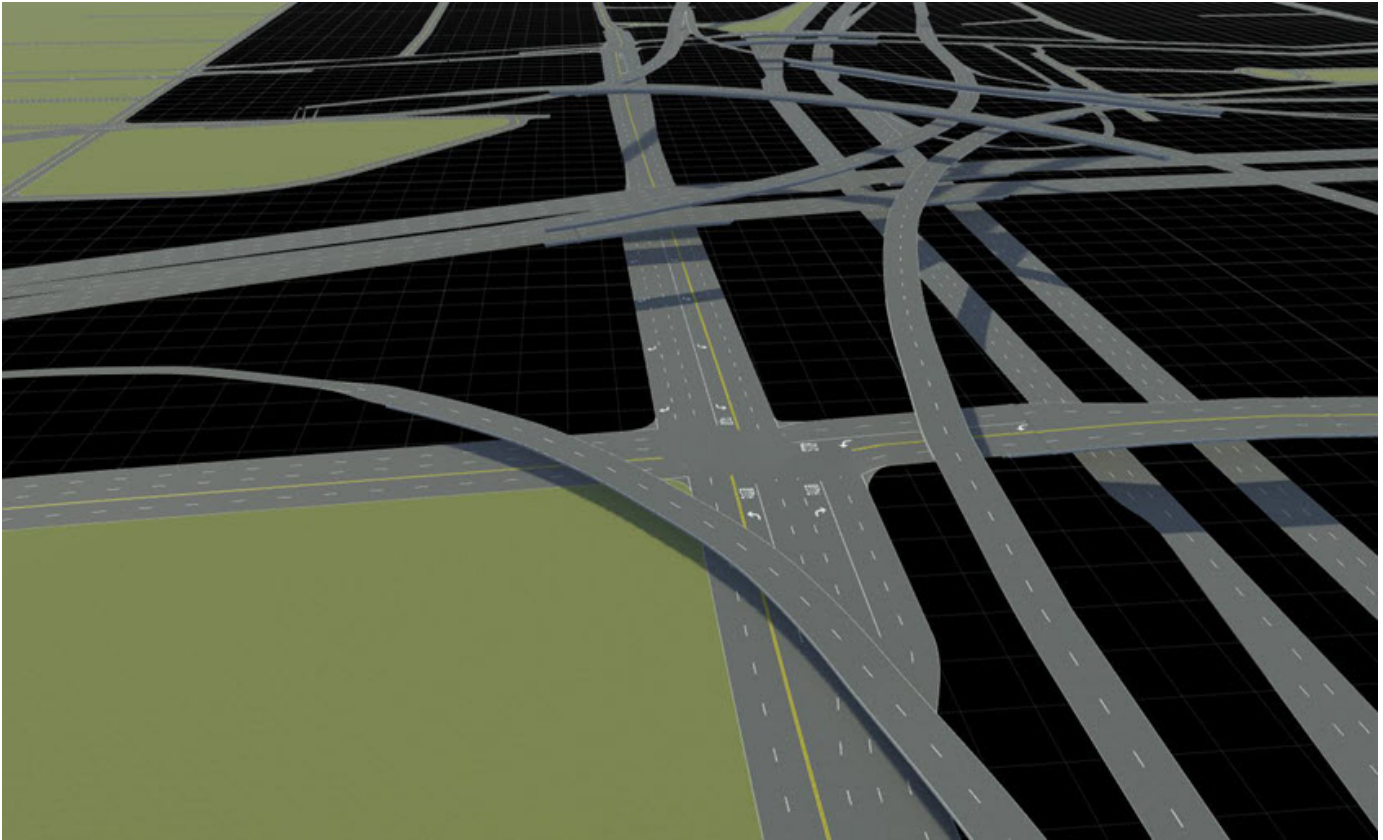
After you build roads, you can modify the scene in RoadRunner. You can also export the scene to ASAM OpenDRIVE file. For more information, see "Export to ASAM OpenDRIVE" on page 5-24.

If RoadRunner detects lane marking overlaps when building roads, then might display this message in the SD Map Builder Results dialog box:
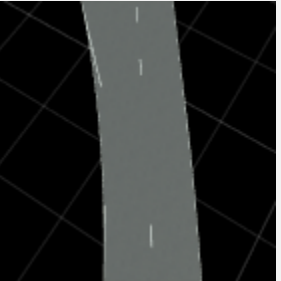
>WARNING: Lane marking overlaps detected. Adjust road centers at these locations

To resolve this issue, open the **Road Plan Tool**, click-navigate to the overlap locations, and adjust the road centers.
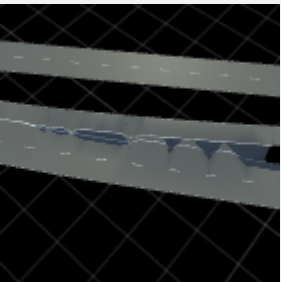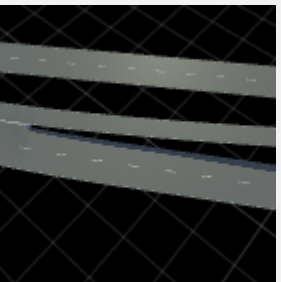
Troubleshoot Import and Build Issues

Depending on the region from which you build map data, you might encounter issues when the **SD Map Viewer Tool** imports data and builds roads. Follow these steps to fix these known issues.
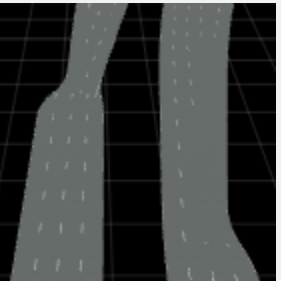
Gap

Issue	Solution
The built road contains gaps between roads at intersections.	Open the Custom Junction Tool, navigate to the affected junction, and increase the ray **Distance** in the **Attributes** pane.

Steep Road Meshes

Issue	Solution
The built road contains steep rises or falls in the road meshes at road junctions.	Open the Corner Tool, navigate to the affected junction, and reduce the **Corner Radius** in the **Attributes** pane.

Roads Under Terrain

Issue	Solution
The built scene contains roads under the terrain.	Open the Surface Tool, navigate to the affected location, and manually adjust the terrain.

Limitations

- RoadRunner does not support direct import of live map data from `openstreetmap.org`.
- Because OpenStreetMap data does not contain elevation information for roads, RoadRunner builds bridges or overpasses that cross a road as intersecting roads.
- Some issues with built roads might be due to missing or inaccurate map data in the OpenStreetMap service. To check whether data is missing or inaccurate due to the map service, view the map data on an external map viewer.

See Also

SD Map Viewer Tool | **Custom Junction Tool** | **Corner Tool** | **Surface Tool**

More About

- "Build Roads by Using Zenrin Japan Map API 3.0 (Itsumo NAVI API 3.0) Data" on page 3-12
- "Export to ASAM OpenDRIVE" on page 5-24
- "Importing ASAM OpenDRIVE Files" on page 3-2

External Websites

- `openstreetmap.org`

Design Scenes

Resolve Triangulation Issues in Junctions

In RoadRunner, a junction represents the complex intersection of multiple roadways, defining a space where multiple surfaces compete for influence over the junction's final surface representation. Even in simple intersections, roads can vary by width, length, bank angle, and elevation. Roads also vary by features such as medians, curbs, and sidewalks, which need to be gracefully clipped from the final result. The goal of RoadRunner software is to unify these overlapping regions into a single representation suitable for simulation use cases. This unification often requires triangulating the junction surface to export into various formats. This task is nontrivial and can often lead to undesirable artifacts in the final junction triangulation.

To avoid common triangulation issues, use these tips.

Adjust Road Elevations

RoadRunner continually detects overlaps with neighboring roads and automatically creates junctions for any roads which overlap within 2 vertical meters. However, given that each road is fully independent, it is possible to create intersections that vary in grade, which can cause undesirable triangulation artifacts. One way to resolve this issue is to adjust road elevations to match as closely as possible within the junction.

From within the **Road Plan Tool**, the RoadRunner 2D Profile editor displays all overlapping roads for any selected road. You can use the tool to raise or lower any road to match the height of other roads by selecting and dragging either the height profile nodes or spans. Dragging a span is equivalent to dragging the nodes on either end.
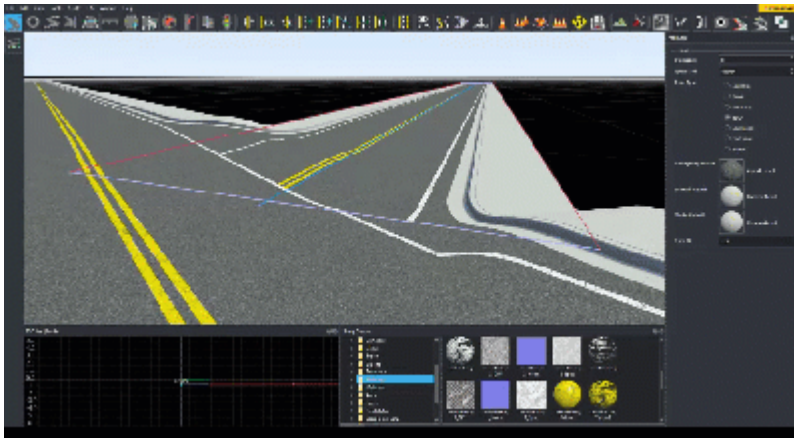
Bank Roads

When two or more roads that intersect have different slopes, the intersections might need to be banked to better align the road surfaces. The RoadRunner **Cross Section Tool** offers an interface to adjust road bank at lane boundary locations. To use this tool, select the road you want to edit, select a cross section, and adjust the banking by using the 2D Cross section editor window.
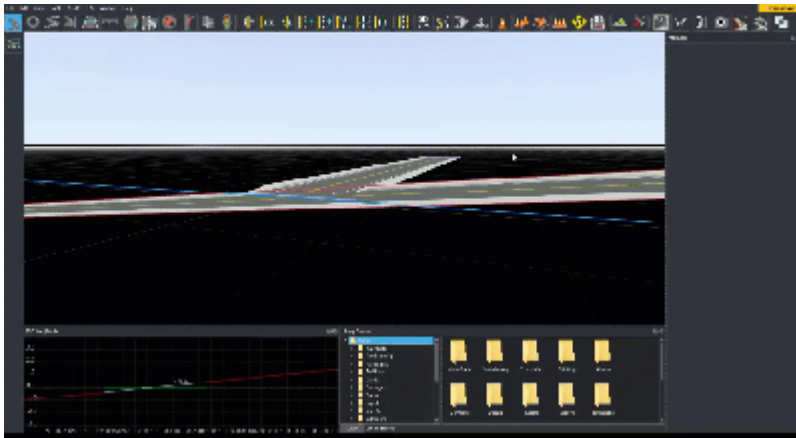
Use Slip Connections

RoadRunner offers a way to enforce height constraints between roads that have a dependent relationship, such as a freeway and a freeway offramp. By creating slip roads, the end height and slope of the slip road is constrained to that of the main road. To build a slip road, use the **Slip Road Tool** to pull a slip road off of any other road in your scene.
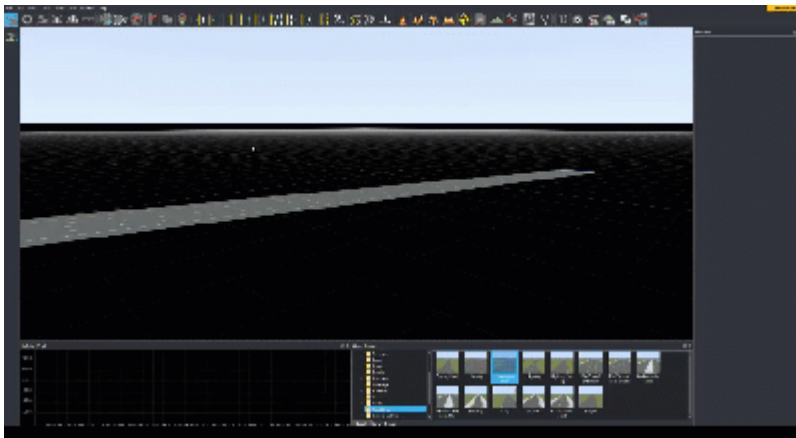
How Surfaces Work in RoadRunner

Using the **Surface Tool**, you can model surfaces around roads, such as walkways, driveways, parking lots, and natural terrain. The terrain surface model interacts differently with various aspects of a scene.

Terrain Surface Model

Terrain surfaces are region graphs bounded by curves. For more details about region graphs, see "Region Graph Editing" on page 2-78.

RoadRunner creates some curves automatically, such as the curves on the boundaries of roads. You can create other curves manually by using the **Surface Tool**.

Here is an example of a single terrain surface bounded by manually created surface curves:
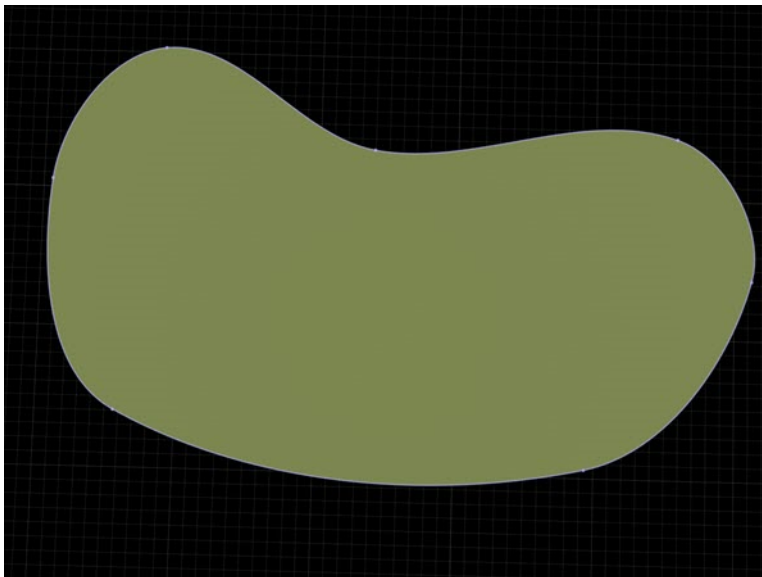
The points on the graph edge curve are curve end nodes, which can be shared by multiple curves. In most regards, these curves use the same UI concepts outlined in the "Curve Editing" on page 2-66 and "Polygon Editing" on page 2-68 topics.

In particular, each curve has a tangent direction that can be modified to change the shape of the curve, as shown in this image:
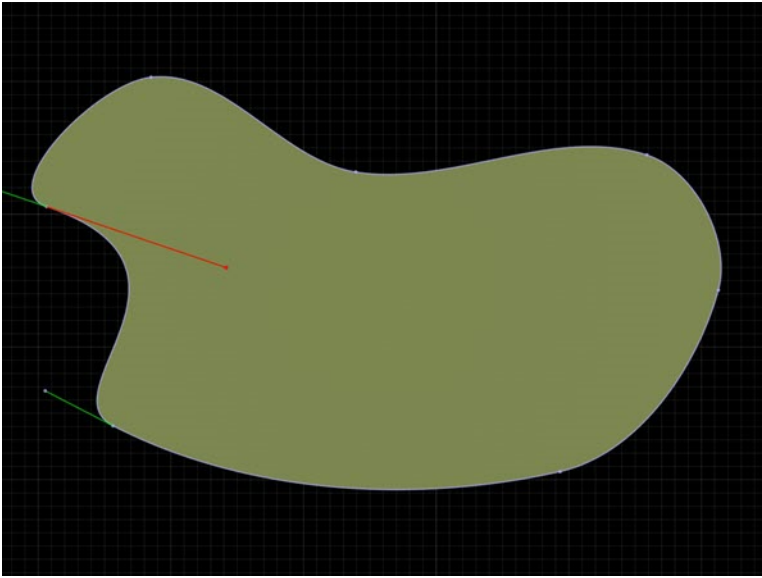
Each surface curve can have one surface connected to each side. The nodes can be shared by any number of surface curves. In this manner, the surface curves form a contiguous (nonoverlapping) patchwork of surfaces called a surface graph.

For example, you can split an initial surface into two surfaces by digitizing new surface curves in the interior, taking care to share end nodes on the perimeter of the surface:
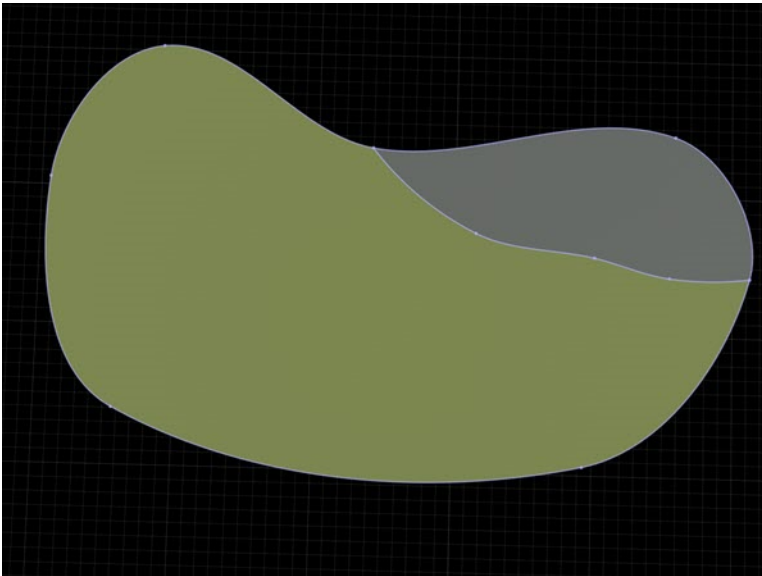
Surfaces also support enclosed surfaces, that is, surfaces within surfaces. Any time a loop of surface curves lies entirely within the interior of another surface, it creates a new surface in the interior.

The following image shows two nested levels of enclosed surfaces:
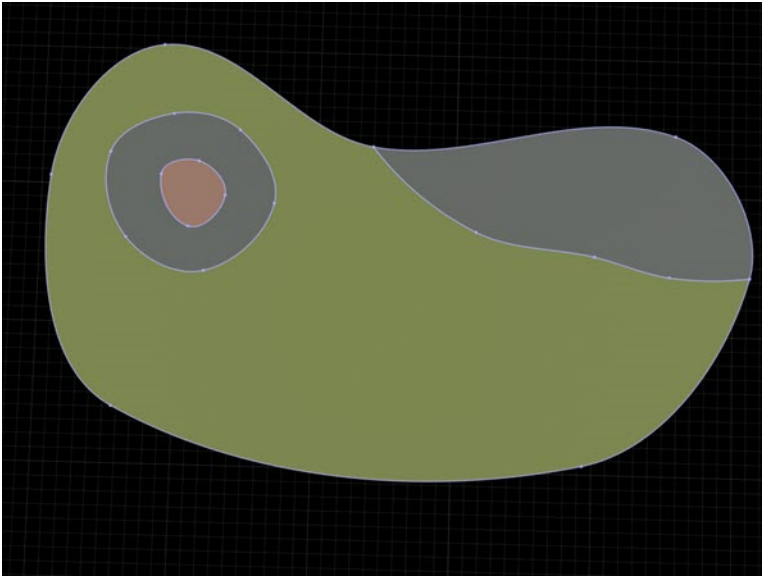
Avoid overlapping surfaces. Surfaces that overlap in the xy dimension cause visual artifacts, or "z-fighting" artifacts.

These artifacts can be seen in the following example, where a new loop of surface curves crosses the existing surface curves:
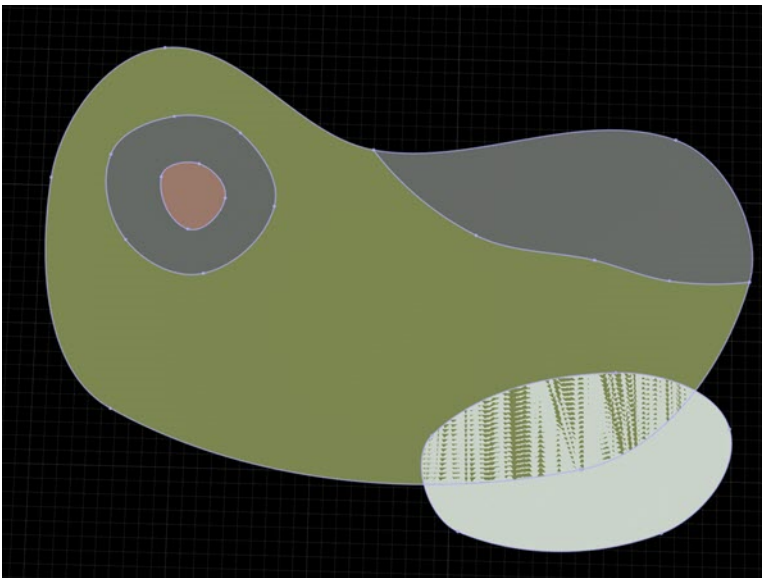
To correct this issue, you must split the original surface curves to introduce nodes. Multiple surface curves then share these nodes

This image includes corrected nodes.
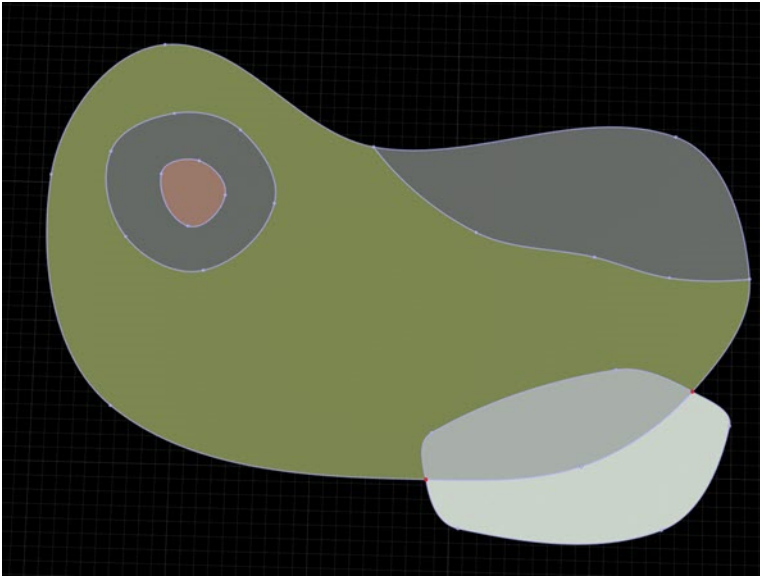
Surfaces and Roads

Roads automatically participate in the surface graph.

Roads that lie entirely within a terrain surface behave much like enclosed surfaces. Terrain surface curves are automatically created around the perimeter of the road network, forming an enclosed road surface.

In the following image, a simple intersection that was created using the **Road Plan Tool** has been digitized in the interior of the surface:
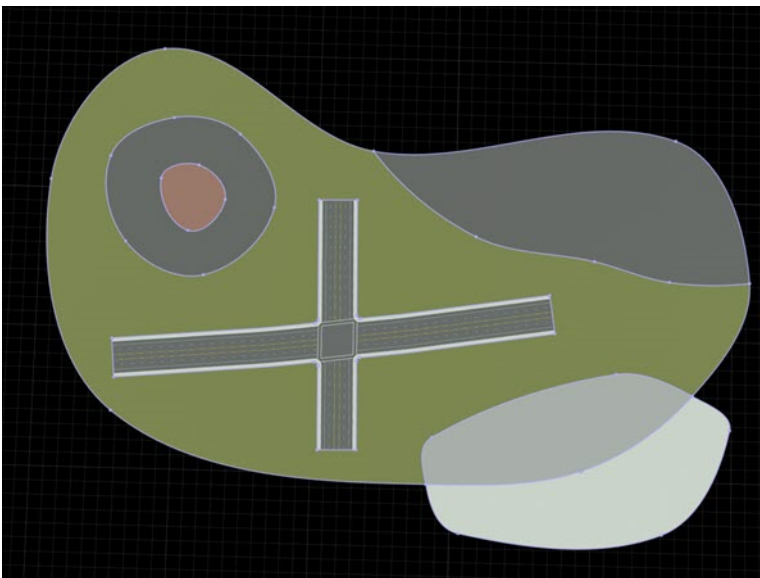
As with overlapping surfaces, roads that overlap a surface curve can cause visual artifacts.

For example, dragging the end of a road such that it crosses a surface curve causes artifacts:
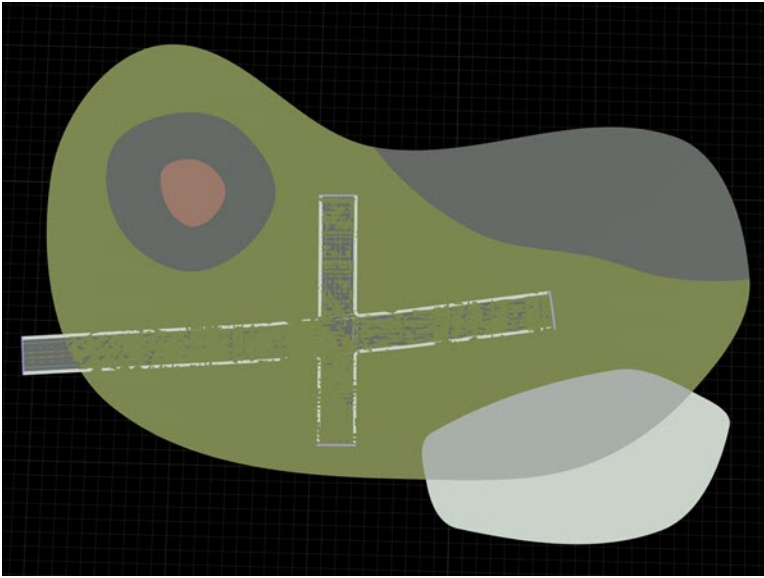
One way to correct this issue is to adjust the containing surface such that the roads are fully enclosed:
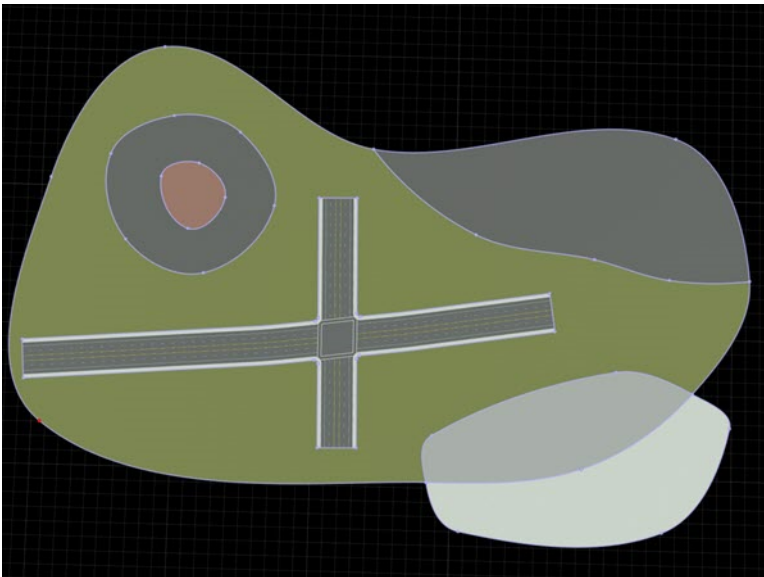
You can also connect surfaces directly to roads. This connection enables surfaces to automatically move when the roads are moved, helping to avoid overlaps.

In the following image, the three nodes attached to the roads are highlighted in red. The two nodes at the end of the road are created automatically and cannot be removed or deleted in the **Surface Tool**. Nodes can also be added parametrically along the side of a road. These points can be inserted anywhere along a road, dragged along the road, or deleted.
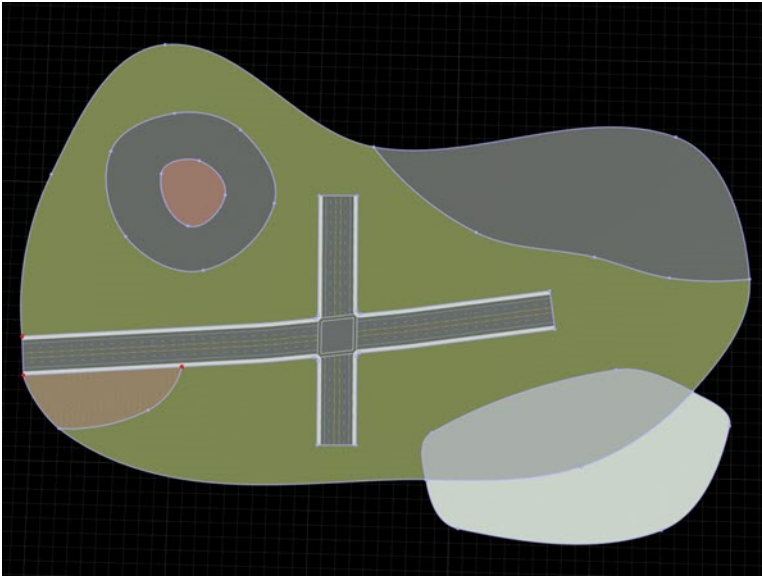
When surfaces are attached to these road surface nodes, moving the road automatically adjusts the surfaces.

In the following image, the end of the road is moved clockwise:
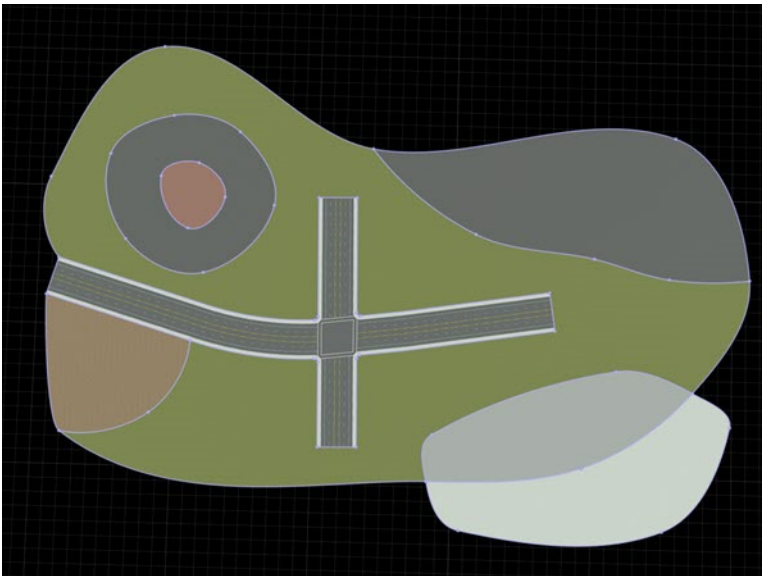
Bridges

Only nonbridge portions of road surfaces participate in the surface graph. For more details, see the **Road Construction Tool**. The surface graph ignores road construction spans that are marked as bridges.

For example, the selected construction span in the following image is marked as a bridge:
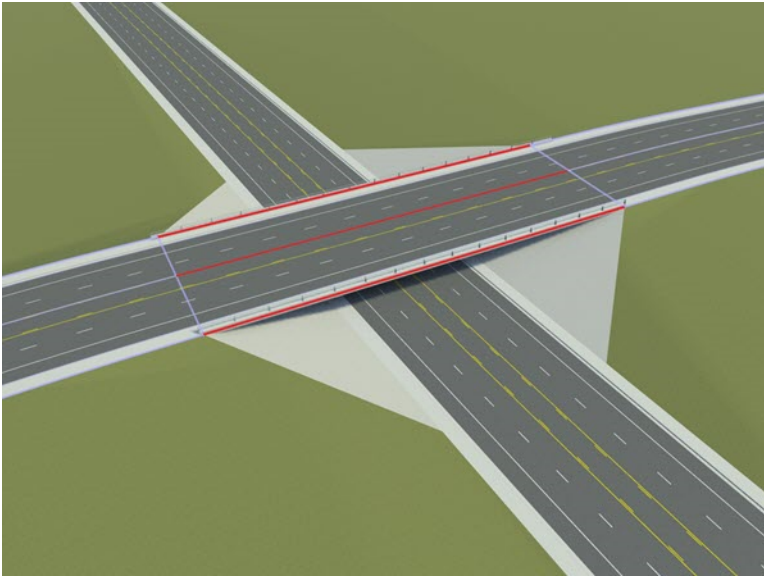
Note A bridge span that has similar elevation to the surfaces underneath it can produce visual artifacts. To avoid artifacts, use the **Road Height Tool** to change the elevation of the bridge span so that it is above the ground surface beneath it.

Extruded Surfaces

Surfaces have an optional height attribute. If the height is nonzero, the surface is vertically extruded upwards. By increasing the height, you can create simple mock buildings, as shown here:
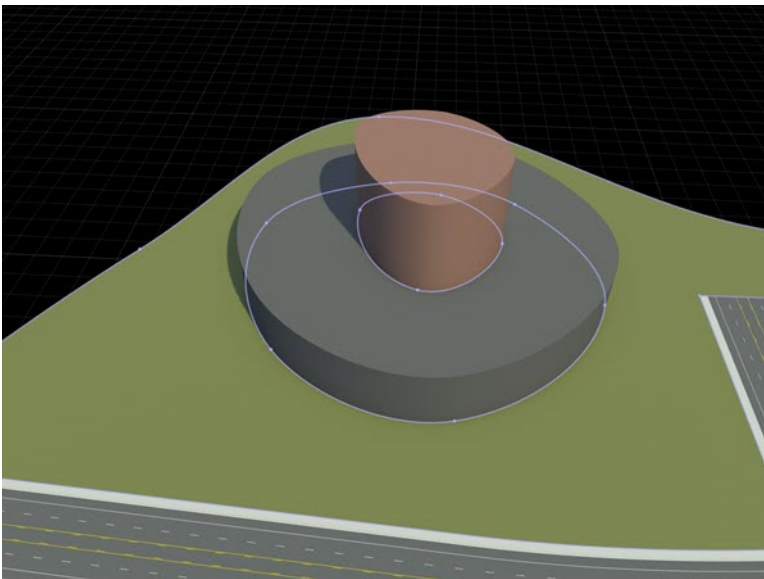
Surfaces and Elevation

By default, the heights within a surface are automatically interpolated from the heights of the surrounding surface curves.

For example, this image shows a surface containing curves that have a nonzero height, represented as a positive z-coordinate:
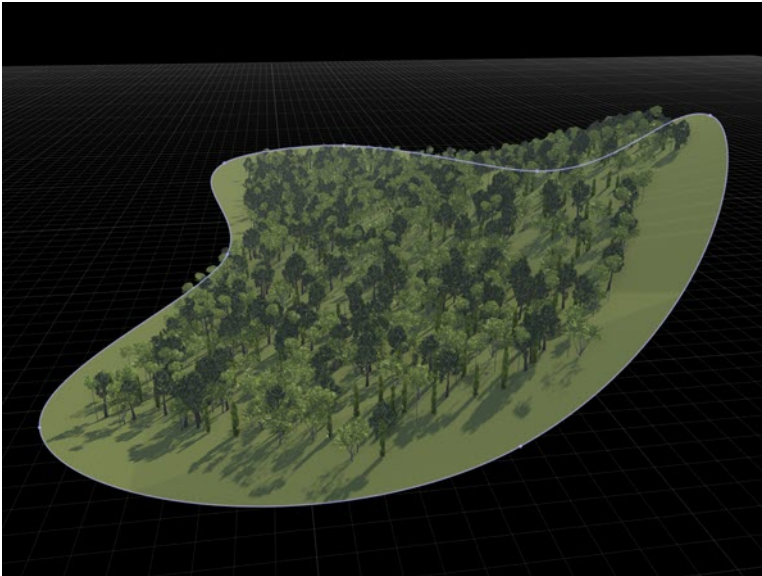
In scenes with elevation maps, that is, **Elevation Map Assets**, each surface can optionally use the elevation maps to define the interior elevations of the surface. For more details, see Control Whether a Surface Uses Elevation Samples.

See Also
Surface Tool | **Prop Point Tool** | **Prop Polygon Tool** | **Lane Marking Tool**

Related Examples
- "Region Graph Editing" on page 2-78

Create Parking Garage

This example shows how to create a parking garage structure in RoadRunner. The typical North American parking garage consists of multiple repeated levels of parking spaces with a connecting ramp, often with parking spaces, that links the levels together. To create a parking garage in RoadRunner, follow these steps:

- Create a template of one level of the parking garage.
- Modify an instance of the template for the ground level.
- Copy, paste, translate, and connect instances of the parking garage level.
- Modify the top-level routes.
- Add any additional elements to the structure or surroundings.

The next sections describe the listed steps in detail.

Create a Parking Level Scene Template

Develop a single layer of the desired parking garage layout. This example considers an exterior lane with a split-lane ramp connecting levels, and parking spaces on both sides of the ramp. Add the parking spaces using the **Lane Add Tool**, **Parking Tool**, and **Lane Marking Tool**. Create the ends of the garage using the **Custom Junction Tool**, with some adjustments to the radii. In this single layer, the split lane ramp is at the same level as the exterior lane, which enables the layers to connect when stacked.

Add additional props to the scene from the assets library, including:

- Posts — The posts, when stacked, become the structure of the garage.
- Extrusions — The extrusions form the interior and exterior guard rails of the garage.
- Stencils — The stencils, such as SLOW and YIELD, indicate expected the traffic flow in the garage.

This figure shows a sample of the single layer of the parking garage.
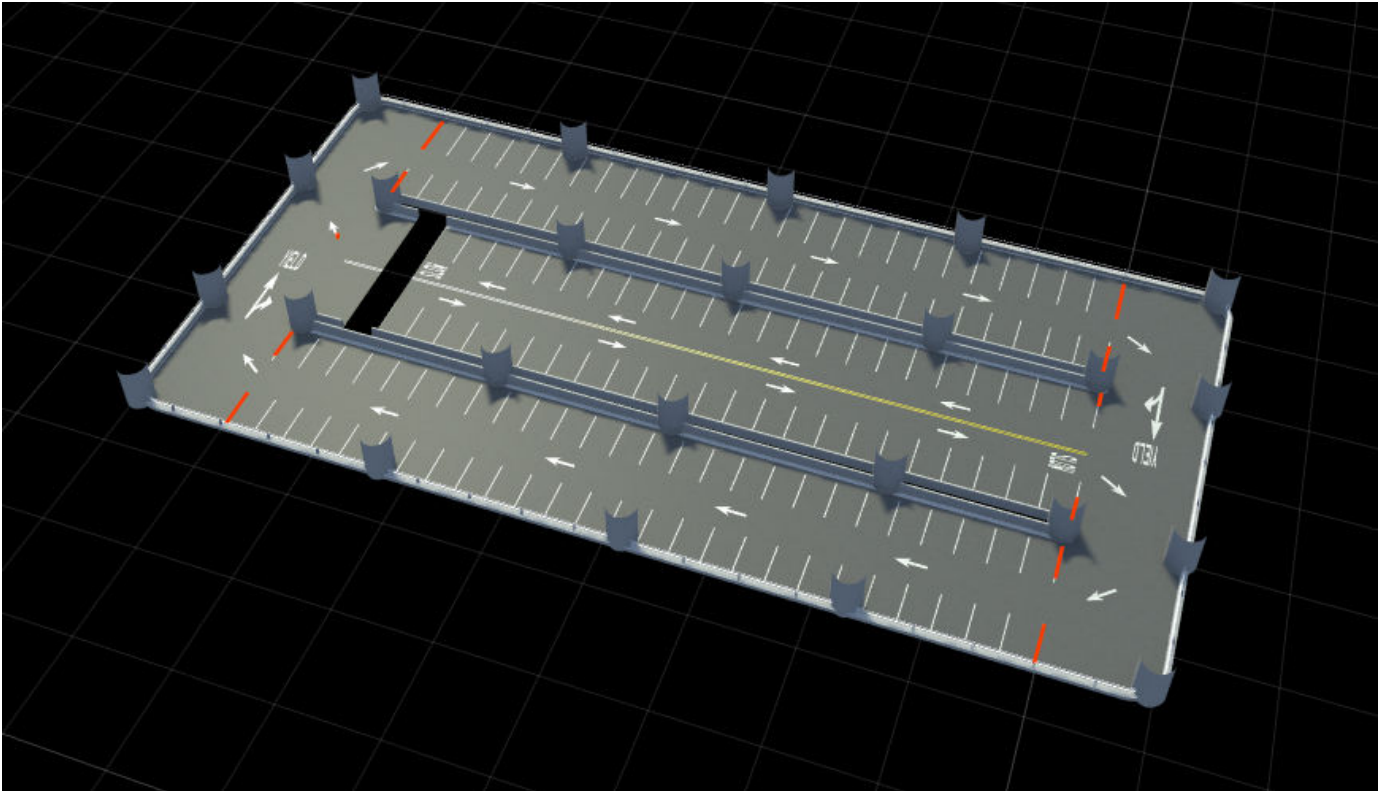
After completing a single layer of the parking garage, save the layer as a scene template. In the menu, select **Assets**, then select **New > Folder** and name the folder `Templates`. Select **New > Scene Template** to save the current layer. Choose a name, such as `ParkingGarageLevel`, for the new asset. For more information on creating scene template assets, see "Create, Import, and Modify Scene Assets" on page 2-58.
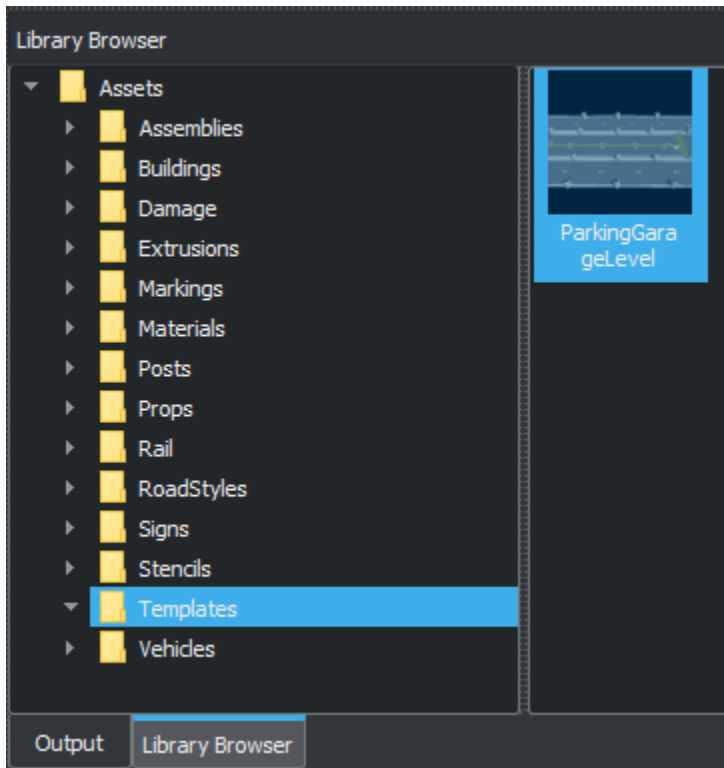
Create Ground Level

Create the ground level of the parking garage by modifying the parking garage level in the current scene. The sample ground level scene includes these modifications:

- Surrounding grass, added using the **Surface Tool**.
- An outside street, with connection to the garage.
- Sidewalks surrounding the perimeter of the garage.
- Extra-wide accessible designated parking spaces near the garage entrance.
- Removal of some of the guardrail extrusions.

This figure shows the ground level of the parking garage.
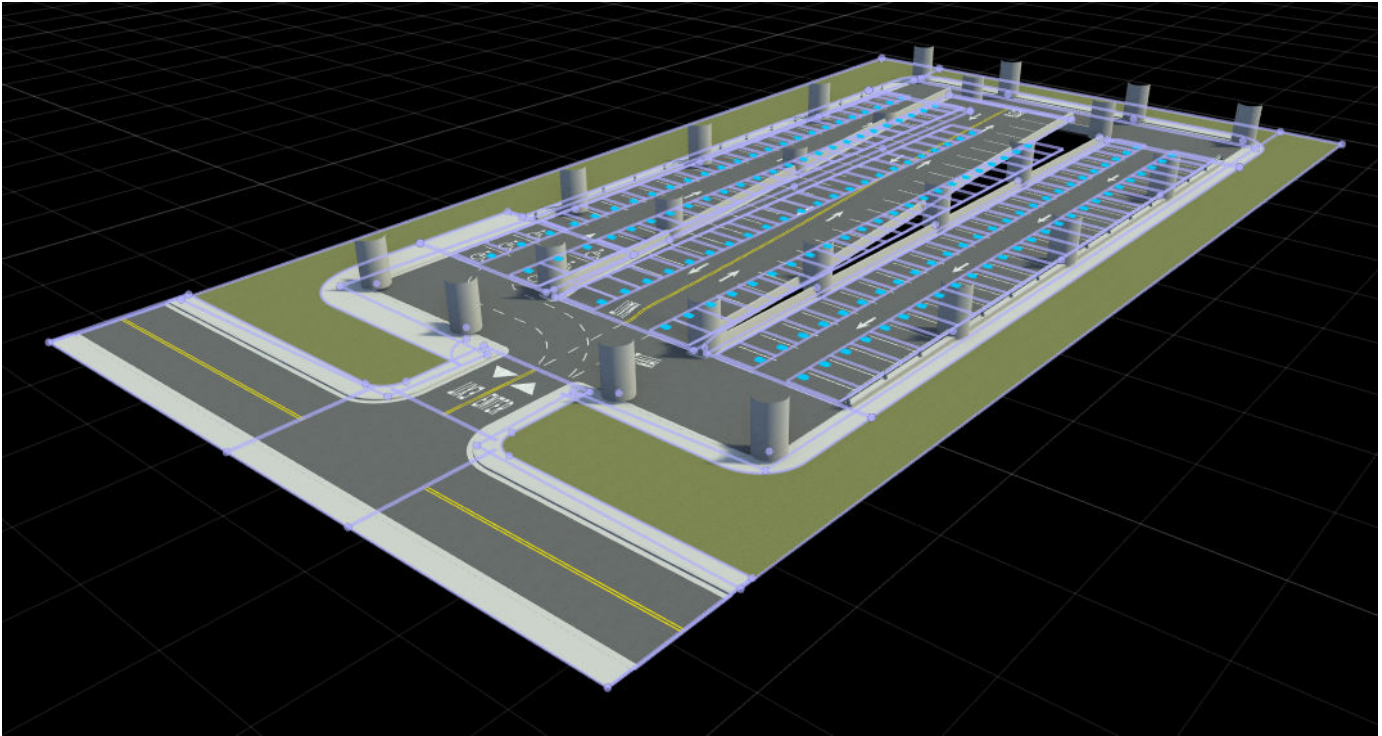
Add Levels to Parking Garage

From the RoadRunner Asset Library, drag and position the `ParkingGarageLevel` asset created in the previous section. Align the new level horizontally with the ground level using the `Posts` props as guides. Drag the green axis arrow to raise and lower the level until the level is at the desired height. With the level still selected, copy (**Ctrl+C**) the level.

Note Once you click the axis arrow, do not release the button until the level is in the final, desired position.

To add additional levels, paste (**Ctrl+V**) a copy of the previous level and repeat the previous positioning steps until you have added desired number of levels. This figure shows a new level being added to the garage.
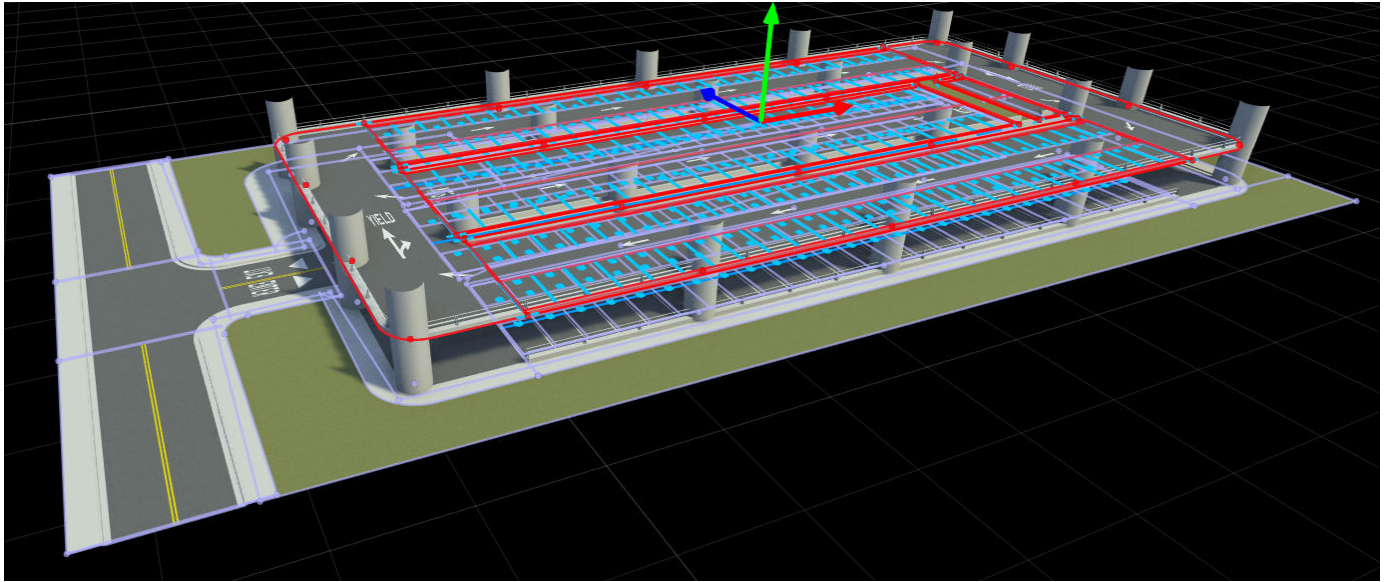
On each level, select the ramp. Using either the **Road Height Tool** or **2D Editor**, adjust the incline of the ramp to connect it to the next level. You can use the **Road Plan Tool** to connect the ramp to the next level. These figures show the ramp and **2D Editor**
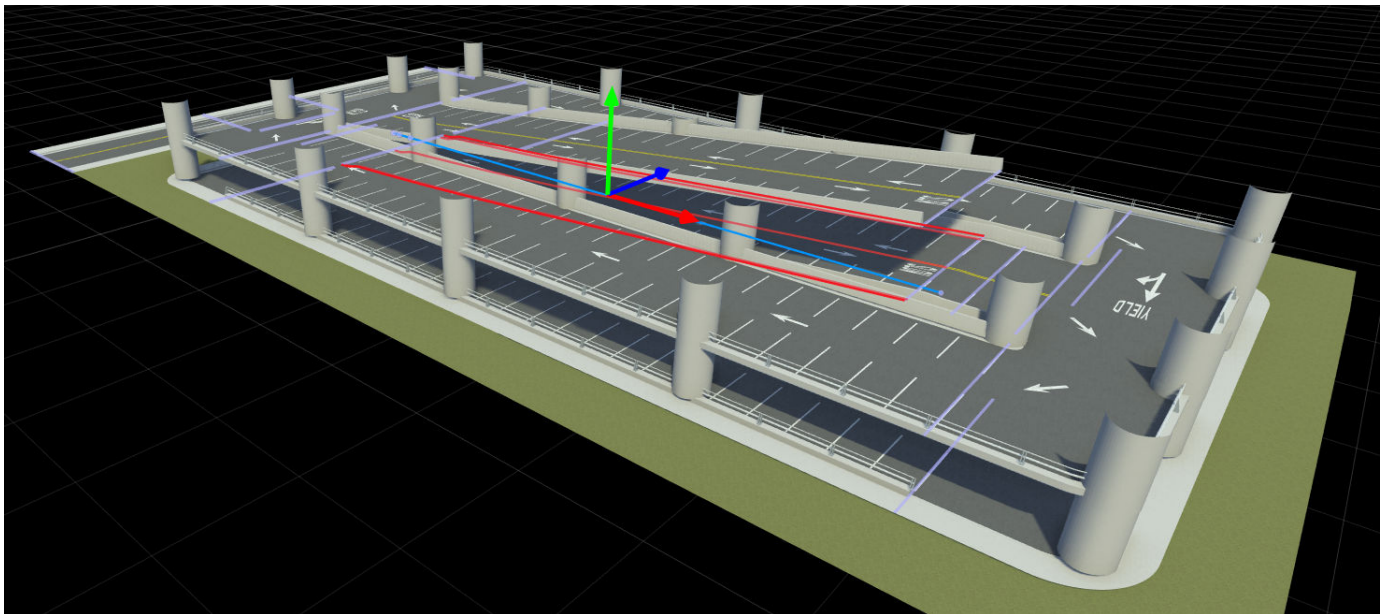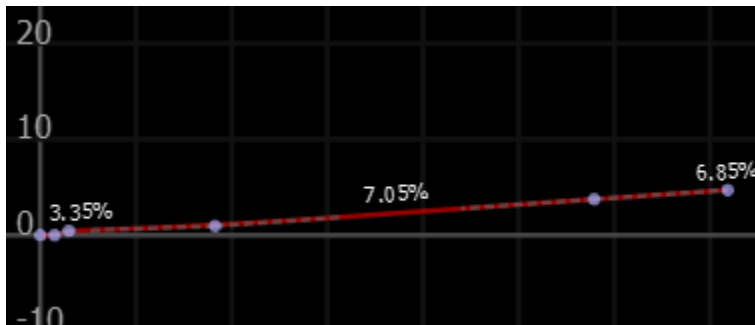
On the top level of the garage, select and delete the ramp, as it does not connect to any level. Using the **Custom Junction Tool**, modify the top level junction radii to fit over the `Posts`.
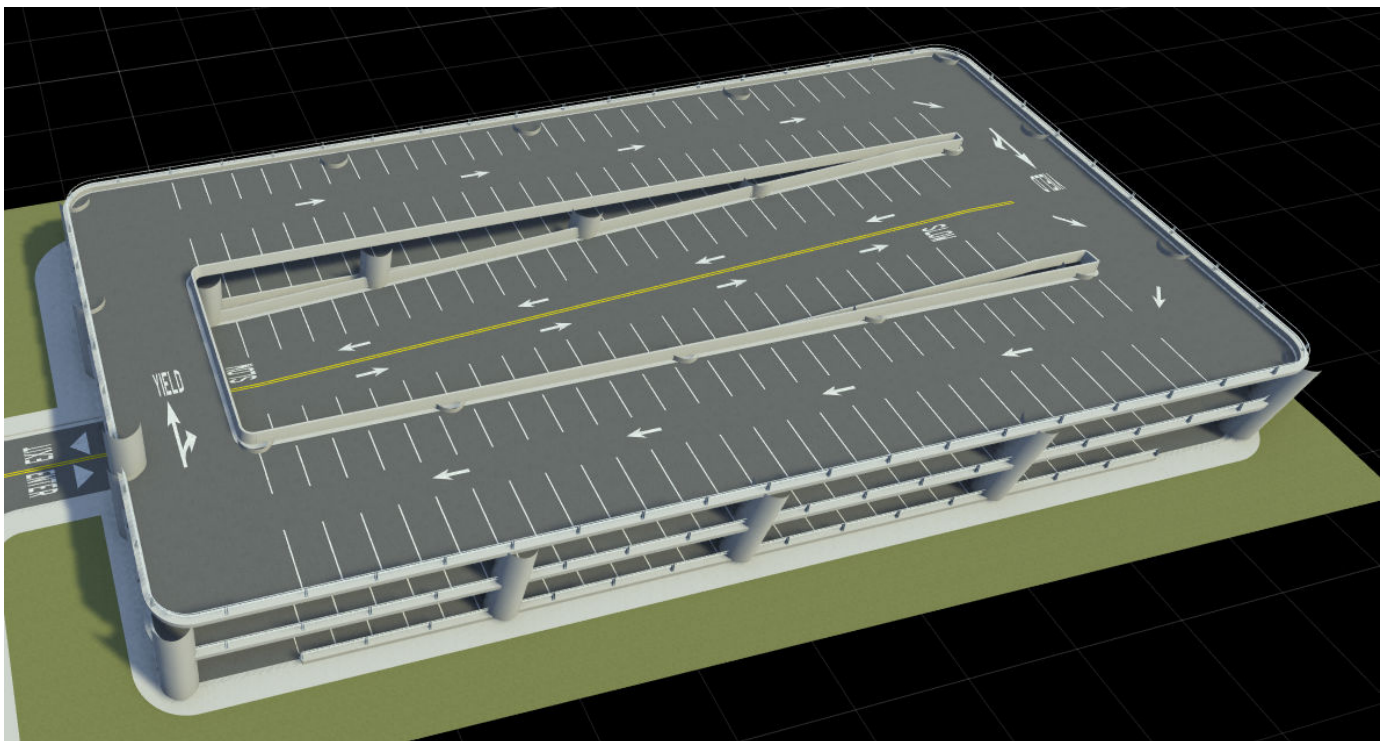
Complete Garage Structure

Complete the garage structure by either adjusting specific elements in each level, such as `Posts`, or adding additional props to the scene from the RoadRunner Asset Library. In the sample parking garage, `Signs` and `Trees` have been added to complete the scene. This image shows the final parking garage scene.
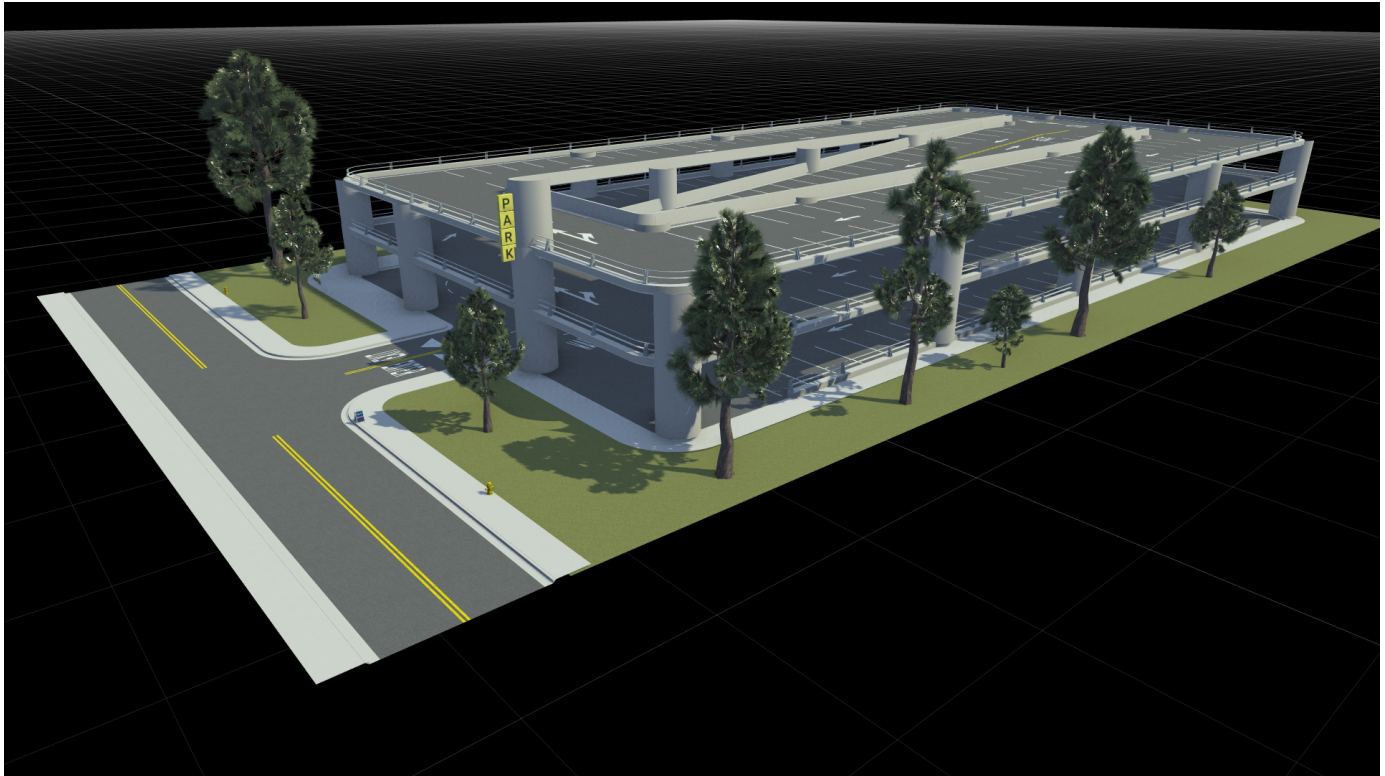
See Also
Custom Junction Tool | Road Height Tool | Road Plan Tool | Surface Tool | "Create, Import, and Modify Scene Assets" on page 2-58

Export Scenes

Export to AutoCAD

You can export RoadRunner scenes to the AutoCAD DXF file format.

AutoCAD Export

From the **File** menu, select **Export**, then **AutoCAD (.dxf)** to open the Export AutoCAD dialog box. Then, specify a path to which to export the file, and click **Export**. Before exporting, you can optionally set these parameters.

Split by Segmentation

Split meshes by their segmentation type. For more details, see Segmentation on page 5-49.

Power of Two Texture Dimensions

Resize the dimensions of exported textures by rounding them up to the next highest power of two.

Embed Textures

Embed the exported textures inside the exported file.

Export to Tiles

Split the meshes per tile. This parameter also groups props by the tile that they are in.

- By default, only one file is exported. Tiles are stored in separate nodes.
- The **Tile Size** parameter specifies the (x,y) dimensions of the exported tiles. Units are in pixels.
- The **Tile Center** parameter specifies the (x,y) location of the tile centers, which is (0,0) by default.
- If you enable the **Export Individual Tiles** parameter, then RoadRunner exports each tile as a separate file. The files follow this naming convention: *ExportedFileName*_Tile_0_0.*ext*, *ExportedFileName*_Tile_1_0.*ext*, and so on.

Export to FBX

RoadRunner can export scenes to the FBX file format. Although this option is compatible with Unity and Unreal®, for those applications, using the specific export option along with MathWorks plugins is recommended.

FBX Export

From the **File** menu, select **Export**, then **Filmbox (.fbx)** to open the Export Filmbox dialog box. Then, specify a path to which to export the file, and click **Export**. Before exporting, you can optionally set these parameters.

Split by Segmentation

Split meshes by their segmentation type. For more details, see Segmentation on page 5-49.

Power of Two Texture Dimensions

Resize the dimensions of exported textures by rounding them up to the next highest power of two.

Embed Textures

Embed the exported textures inside the exported file.

Export to Tiles

Split the meshes per tile. This parameter also groups props by the tile that they are in.

- By default, only one file is exported. Tiles are stored in separate nodes.
- The **Tile Size** parameter specifies the (*x*,*y*) dimensions of the exported tiles. Units are in pixels.
- The **Tile Center** parameter specifies the (*x*,*y*) location of the tile centers, which is (0,0) by default.
- If you enable the **Export Individual Tiles** parameter, then RoadRunner exports each tile as a separate file. The files follow this naming convention: *ExportedFileName*_Tile_0_0.*ext*, *ExportedFileName*_Tile_1_0.*ext*, and so on.

Advanced Details

Node Naming

- All nodes have "Node" appended to the end of their name (for example, *Roads*Node).
- Props have their GUID prepended to the front of their name (for example, *{40ce66ca-c817-425b-8802-17cdbd76371f}*Signal_Post_30ftNode).

  - Props generated from a curve, polygon, or span share GUIDs with the associated curve, polygon, or span.
- During export, all node names in the scene graph are made to be unique.

  - _# is appended for duplicate node names (for example, light_green_*1*Node).
  - Because the scene graph is not a tree, duplicate names are still possible when converting to FBX if there are multiple instances of the same node (such as when reusing props).

- When the mesh is split by segmentation, the roads and terrain have extra child nodes for each segmentation type it has, with the segmentation type appended to the name (for example, Road_*Sidewalk*Node).

- When the mesh is split by transparency sorting layer (for the Unreal on page 5-85 export option), the roads and terrain have extra child nodes for each sorting layer, with the layer number appended to the name (for example, Roads_*Layer2*Node).

  - This can also be combined with the segmentation type (for example, Road_*Sidewalk_Layer0*Node).

- For traffic signals (see **Signal Tool**), the name of the variant is added to the FBX name (for example, *{4674ef2e-deea-403c-9b52-487e0ba9f13d}*Signal_3Light_Bare01_*RedYellowGreen Left*Node).

Material Details

- Materials are converted into FbxSurfacePhong materials.

- When the mesh is split by segmentation, the segmentation type is appended to the material name (for example, Concrete1_*Curb*).

- When materials need a transparency sorting order defined (typically for overlapping transparent markings), a duplicate of the material is created.

- Materials with duplicate names add _# to distinguish between them (for example, Leaves_*1*).

  - This can be combined with the segmentation type (for example, OilPath01_*Road_1*).

Light Source Parameters

When you import FBX file assets containing light sources, the parameters that control lighting effects, such as color and brightness, are carried through to exported FBX files.

FBX Scene Settings

RoadRunner exports scenes with the Maya Z-up axis system. Units are in meters.

Combo Exports

Other export options combine the FBX export with other files. Depending on the target application, the RoadRunner software applies extra changes.

- Unity on page 5-59: Mesh is identical to the normal FBX export option.

- Unreal on page 5-85: Mesh is split by transparency sorting order.

- CARLA on page 5-94: Mesh is split by segmentation and transparency sorting order.

See Also

Related Examples

- "Customize Levels of Detail in Exported Scenes" on page 5-111

Export to glTF

RoadRunner can export scenes to the GL Transmission Format (glTF) and GL Transmission Format Binary File (glb).

glTF Export

From the **File** menu, select **Export**, then **glTF (.gltf, .glb)** to open the Export glTF dialog box. Then, specify a path to which to export the file, and click **Export**. Before exporting, you can optionally set these parameters.

Split by Segmentation

Split meshes by their segmentation type. For more details, see Segmentation on page 5-49.

Power of Two Texture Dimensions

Resize the dimensions of exported textures by rounding them up to the next highest power of two.

Export to Tiles

Split the meshes per tile. This parameter also groups props by the tile that they are in.

- By default, only one file is exported. Tiles are stored in separate nodes.
- The **Tile Size** parameter specifies the (x,y) dimensions of the exported tiles. Units are in pixels.
- The **Tile Center** parameter specifies the (x,y) location of the tile centers, which is (0,0) by default.
- If you enable the **Export Individual Tiles** parameter, then RoadRunner exports each tile as a separate file. The files follow this naming convention: *ExportedFileName*_Tile_0_0.*ext*, *ExportedFileName*_Tile_1_0.*ext*, and so on.

Limitations

RoadRunner follows the glTF 2.0 specification as much as possible, but there are some limitations.

- Texture sampler information is not exported. This limitation might result in texture clamping issues for objects like signs.
- RoadRunner uses a specular setup for materials. Because glTF uses metallic-roughness by default, RoadRunner attaches whatever is in the "Specular Map" slot to the "metallicRoughnessTexture" in the exported material and sets the "metallicFactor" to 0.

Export to OpenFlight

RoadRunner can export scenes to the OpenFlight (`.flt`) file format.

OpenFlight Export

From the **File** menu, select **Export**, then **OpenFlight (.flt)** to open the Export OpenFlight dialog box. Then, specify a path to which to export the file, and click **Export**. Before exporting, you can optionally set these parameters.

Split by Segmentation

Split meshes by their segmentation type. For more details, see Segmentation on page 5-49.

Power of Two Texture Dimensions

Resize the dimensions of exported textures by rounding them up to the next highest power of two.

Export to Tiles

Split the meshes per tile. This parameter also groups props by the tile that they are in.

- By default, only one file is exported. Tiles are stored in separate nodes.
- The **Tile Size** parameter specifies the (x,y) dimensions of the exported tiles. Units are in pixels.
- The **Tile Center** parameter specifies the (x,y) location of the tile centers, which is (0,0) by default.
- If you enable the **Export Individual Tiles** parameter, then RoadRunner exports each tile as a separate file. The files follow this naming convention: *ExportedFileName*`_Tile_0_0.`*ext*, *ExportedFileName*`_Tile_1_0.`*ext*, and so on.

Limitations

- Texture wrapping settings are not exported. This limitation might result in texture clamping issues for objects like signs.
- The OpenFlight file is exported through the OpenSceneGraph plugin, which exports OpenFlight version 16.1.
- RoadRunner does not export the normal or specular maps for materials.

Export to OpenSceneGraph

RoadRunner can export scenes to various OpenSceneGraph file formats, including `.osg`, `.osgb`, and `.ive`.

OpenSceneGraph Export

From the **File** menu, select **Export**, then **OpenSceneGraph (.osg, .osgb, .ive)** to open the Export OpenSceneGraph dialog box. Then, specify a path to which to export the file, and click **Export**. Before exporting, you can optionally set these parameters.

Split by Segmentation

Split meshes by their segmentation type. For more details, see Segmentation on page 5-49.

Power of Two Texture Dimensions

Resize the dimensions of exported textures by rounding them up to the next highest power of two.

Embed Textures

Embed the exported textures inside the exported file.

Export to Tiles

Split the meshes per tile. This parameter also groups props by the tile that they are in.

- By default, only one file is exported. Tiles are stored in separate nodes.
- The **Tile Size** parameter specifies the (*x*,*y*) dimensions of the exported tiles. Units are in pixels.
- The **Tile Center** parameter specifies the (*x*,*y*) location of the tile centers, which is (0,0) by default.
- If you enable the **Export Individual Tiles** parameter, then RoadRunner exports each tile as a separate file. The files follow this naming convention: *ExportedFileName*`_Tile_0_0.`*ext*, *ExportedFileName*`_Tile_1_0.`*ext*, and so on.

Limitations

- Texture wrapping settings are not exported. This limitation might result in texture clamping issues for objects like signs.
- RoadRunner does not export the normal or specular maps for materials.

Export to Wavefront OBJ

RoadRunner can export scenes to the Wavefront OBJ (`.obj`) file format.

Wavefront Export

From the **File** menu, select **Export**, then **Wavefront (.obj)** to open the Export Wavefront dialog box. Then, specify a path to which to export the file, and click **Export**.

Advanced Details

This code shows an example of a material in an exported Material Library (`.mtl`) file:

```
newmtl Grass1                       # Material Name
illum 2                             # Color with specular highlights
Ka 1.000000 1.000000 1.000000       # Ambient color matches the diffuse
Kd 1.000000 1.000000 1.000000       # Diffuse color
Ks 0.039216 0.039216 0.039216       # Specular color
Ns 800.200012                       # Specular exponent, approximated from the Roughness value
Tr 0.000000                         # Transparency
map_Kd Grass1_Diff.png              # Diffuse Map
map_bump Grass1_Norm.png            # Normal Map
map_Ks Grass1_Spec.png              # Specular Map
```

Export to GeoJSON

GeoJSON Overview

RoadRunner can export scenes to a GeoJSON file format. The GeoJSON file format is a JavaScript Object Notation(JSON) based open standard format for representing geospatial data. This format defines objects to represent geographical features. A GeoJSON object comprises one or more geometries, a feature or a collection of features. A geometry in GeoJSON format represents a basic shape such as points, surfaces, and curves. All GeoJSON geometries consist of at least a `type` property and a `coordinates` property. The geometry types supported by the GeoJSON format are `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, and `MultiPolygon`. `Point` and `LineString` are the simplest geometry `type` which are represented by a single coordinate and a set of coordinates respectively. `Polygon` geometries are comparatively complex and may contain inner and outer polygons. They are represented by a multi-dimensional array of coordinates.

In real world, other properties may also be required to represent geographical data adequately, apart from their geometry. To represent additional properties and their geometries collectively, a `Feature` is used. A `FeatureCollection` consists of multiple features. This snippet shows a simple GeoJSON `FeatureCollection` object that specifies the geometry and the properties for a feature:

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
        [[30.0, 10.0], [40.0, 40.0], [20.0, 40.0], [10.0, 20.0], [30.0, 10.0]]
    ]
    },
      "properties": {
        "name": "desert"
      }
    }
  ]
}
```

In RoadRunner, the GeoJSON format is meant to complement the ASAM OpenDRIVE file format and fill in some of its missing geospatial data, but this format can also be used independently too.

GeoJSON Export

From the **File** menu, select **Export** , then select **GeoJSON (.geojson)** to open the Export GeoJSON dialog box. Then, specify a path to which you want to export the file to, and click **Export**.

Export Options

Before clicking **Export**, you can optionally select the **Reduce file size** parameter in the Export GeoJSON dialog box.

Leaving the **Reduce file size** option unchecked formats the contents of the exported GeoJSON file in a human readable form. The snippet below shows a sample data in GeoJSON format when the **Reduce file size** option is not selected.

```
 {
  "type": "Feature",
```

```
  "geometry": {
    "type": "Point",
    "coordinates": [57.6, 98.2]
  },
  "properties": {
    "name": "London"
  }
}
```

Selecting the **Reduce file size** option formats the contents of the exported GeoJSON file in a compact format, which might be difficult for the user to read. It also reduces the size of the exported file. The snippet below shows the previous sample data in GeoJSON format when the **Reduce file size** option is selected.

```
{"type":"Feature","geometry":{"type":"Point","coordinates":[57.6,98.2]},"properties":{"name":"London"}}
```

Sample Exported GeoJSON File

The MathWorks version of the GeoJSON format is a collection of lanes, lane boundaries, junctions, gates, crosswalks, and signals.

This code shows a sample exported GeoJSON file.

```
'#' are double values compliant with JSON
{
    "features": [{
        "geometry": {
            "coordinates": [[#,#,#], ...],
            "type": "LineString"
        },
        "properties": {
            "Id": 1,
            "LaneType": "Curb",
            "LeftBoundary": {
                "Dir": "Forward",
                "Id": 0
            },
            "Predecessors": [{
                "Dir": "Forward",
                "Id": 4
            } ],
            "RightBoundary": {
                "Dir": "Forward",
                "Id": 5
            },
            "Successors": [{
                "Dir": "Backward",
                "Id": 6
            }],
            "TravelDir": "Undirected",
            "Type": "Lane"
        },
        "type": "Feature"
    },
    {
        "geometry": {
            "coordinates": [[#,#,#], ...],
            "type": "LineString"
        },
        "properties": {
            "Id": 0,
            "LeftLane": {
                "Dir": "Forward",
                "Id": 1
            },
            "RightLane": {
```

```
                    "Dir": "Forward",
                    "Id": 2
                },
                "Type": "LaneBoundary"
            },
            "type": "Feature"
        },
        {

            "geometry": {
                "coordiantes": [[[[#,#,#], …], …], …],
                "type": "MultiPolygon"
            },
            "properties: {
                "Id": 12,
                "Type": "Junction",
                "Gates": [{"Id": 775}, …],
                "Lanes": [{"Id": 52}, …],
                "Phases": [
                    "Phases": [{
                        "Intervals": [{
                            "BulbStates": [{"Id":0, "On": false, "SignalId": 767}, …],
                            "GateStates": [{"Id": 775, "State": "StopYield"}, …]
                        }]
                    }]
                ]
            },
            "type": "Feature"
        },
        {

            "geometry": {
                "coordinates": [[#,#,#], …],
                "type": "LineString"
            },
            "properties": {
                "Id": 775,
                "Lane": {"Id": 233"},
                "Signals": [{"Id":771}, …],
                "Type": "Gate"
            },
            "type": "Feature"
        },
        {

            "geometry": {
                "coordinates": [[[#,#,#], …], …],
                "type": "Polygon"
            },
            "properties": {
                "Id": 763,
                "Bulbs": [
                    {
                        "Point": [#,#,#],
                        "Name": "LeftTurnRed",
                        "NodeName": "light_red"
                    },
                    …
                ],
                "Name": "PathToFile.fbx",
                "SignalType": "Signal",
                "Type": "Signal"
            },
            "type": "Feature"
        }]
}
```

Traffic Signal Phases in GeoJSON

Most of the information in the GeoJSON files describe the geometry of a scene. The `Phases` sections of these files describe the traffic light phases of signals at each junction.

Consider a junction with three signal phases, with each one containing green-yellow-red intervals of varying durations. This figure shows such a sample phase as it appears in the **2D Editor**.
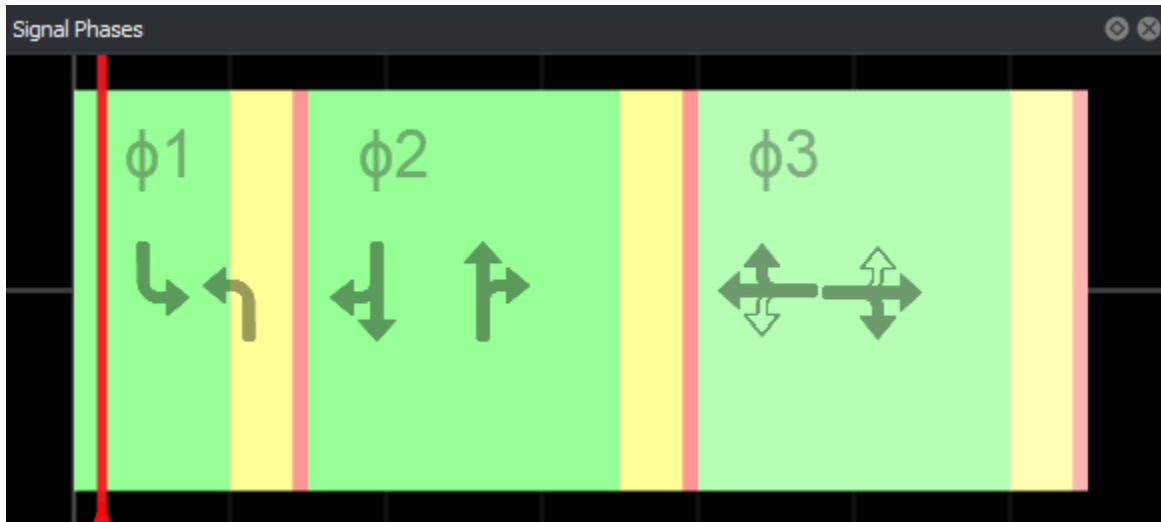
This GeoJSON snippet corresponds to the first traffic phase. Some of the data for this phase has been omitted for clarity.

```
"Phases": [
  {
    "Intervals": [
      {
        "BulbStates": [
          {
            "Id": "{8bedd7ab-7e17-4177-b095-ddca457e6985}",
            "On": false,
            "SignalId": "{cc65a9c4-f47c-465b-8a8f-ae7c7e2aca50}"
          },
          .
          .
          .
        ],
        "GateStates": [
          {
            "Id": "{3233fba0-a5b6-4f09-a442-2e0ddc07c4c0}",
            "State": "Go"
          },
          .
          .
          .
        ],
        "Time": 10,
        "Type": "Green"
      },
      {
        "BulbStates": [
          {
            "Id": "{8bedd7ab-7e17-4177-b095-ddca457e6985}",
            "On": false,
            "SignalId": "{cc65a9c4-f47c-465b-8a8f-ae7c7e2aca50}"
```

```
        },
          .
          .
          .
          .
      ],
      "GateStates": [
        {
          "Id": "{3233fba0-a5b6-4f09-a442-2e0ddc07c4c0}",
          "State": "Go"
        },
          .
          .
          .
          .
      ],
      "Time": 4,
      "Type": "Yellow"
    },
    {
      "BulbStates": [
        {
          "Id": "{8bedd7ab-7e17-4177-b095-ddca457e6985}",
          "On": true,
          "SignalId": "{cc65a9c4-f47c-465b-8a8f-ae7c7e2aca50}"
        },
          .
          .
          .
          .
      ],
      "GateStates": [
        {
          "Id": "{3233fba0-a5b6-4f09-a442-2e0ddc07c4c0}",
          "State": "Stop"
        },
          .
          .
          .
          .
      ],
      "Time": 1,
      "Type": "Red"
    }
  ]
},
{
  "Intervals": [
      .
      .
      .
      .
]
```

The `Time` and `Type` values specify that the first phase has a green interval that lasts 10 seconds, a yellow interval that lasts 4 seconds, and a red interval that lasts 1 second.

Each `BulbStates` section lists the unique ID of a traffic light bulb in the scene. In full GeoJSON files, you can find the bulb specification by searching for this ID. In the first (green) interval, the first bulb in `BulbStates` has an ID of `8bedd7ab-7e17-4177-b095-ddca457e6985` with its `"On"` state set to `false`. In the full GeoJSON file (not shown here), this bulb is specified as being light red. Therefore,

it is expected that the bulb is off at this phase. In the GeoJSON file snippet, you can see that the bulb with this ID is also off in the second (yellow) interval but is then on in the third (red) interval.

Each `GateStates` section lists the unique ID of a maneuver gate in the junction and what its state is during each interval. As with the bulbs in the `BulbStates` sections, the gates in the `GateStates` sections are repeated in each interval, with only the states differing at each interval.

Though not shown in this GeoJSON snippet, the full GeoJSON file includes two additional `Intervals` sections for the two remaining intervals in the phase.

See Also

More About
- "Export to ASAM OpenDRIVE" on page 5-24
- "Importing ASAM OpenDRIVE Files" on page 3-2

Export to USD

RoadRunner can export to the Universal Scene Description (USD) file format.

USD Export

From the **File** menu, select **Export**, then **USD (.usd, .usdc, .usda)** to open the Export USD dialog box.

Specify a path to which to export the file, and click **Export**. Setting the file extension to `.usd` or `.usdc` creates a USD binary file. Setting the extension to `.usda` creates a USD ASCII file.

Before exporting, you can optionally set these parameters.

Split by Segmentation

Split meshes by their segmentation type. For more details, see Segmentation on page 5-49.

Power of Two Texture Dimensions

Resize the dimensions of exported textures by rounding them up to the next highest power of two.

Export to Tiles

Split the meshes per tile. This parameter also groups props by the tile that they are in.

- By default, only one file is exported. Tiles are stored in separate nodes.
- The **Tile Size** parameter specifies the (*x*,*y*) dimensions of the exported tiles. Units are in pixels.
- The **Tile Center** parameter specifies the (*x*,*y*) location of the tile centers, which is (0,0) by default.
- If you enable the **Export Individual Tiles** parameter, then RoadRunner exports each tile as a separate file. The files follow this naming convention: *ExportedFileName*`_Tile_0_0.`*ext*, *ExportedFileName*`_Tile_1_0.`*ext*, and so on.

Limitations

There are several limitations with the `UsdPreviewSurface` schema, which limits the ability to map RoadRunner materials to USD materials. Here is a list of known issues:

- `UsdPreviewSurface` cannot support diffuse color scaling and vertex coloring at the same time. RoadRunner sets the texture reader's scale to the diffuse color of the material. However, vertex colors and a `primvar` reader for them are still included if you want to use them.
- Because `UsdPreviewSurface` does not support specifying a rendering order, overlapping transparent surfaces are not supported.
- Because the double-sided attribute is stored per mesh instead of per material, it is not exported. Therefore, some props (mainly trees) might not render properly.

Convert Asset Data Between RoadRunner and ASAM OpenDRIVE

RoadRunner enables you to import scenes from and export scenes to the ASAM OpenDRIVE (`.xodr`) file format.

RoadRunner assets are represented in ASAM OpenDRIVE using parent elements, such as `Objects`, `Signals`, and `Markings`, and child elements, such as `Type`, `SubType`, `FilePath`, in an XML configuration file. During the export process, RoadRunner uses this configuration file to map the existing assets in the scene to the appropriate parent element and child element for ASAM OpenDRIVE representation. During the import process, RoadRunner resolves the mapping between ASAM OpenDRIVE and the existing RoadRunner assets in the current project. RoadRunner then uses this mapping to place the assets in the scene.

To specify assets when importing, and to specify `Type` or `Subtype` when exporting scenes to the ASAM OpenDRIVE file format, you can modify the asset mapping file manually or interactively. information on how to customize assets interactively, see "Configure Asset Mapping File Interactively" on page 5-20.

Open Asset Configuration File

The asset configuration file is an XML file named `OpenDriveAssetData.xml`. Each RoadRunner project has its own `OpenDriveAssetData.xml` configuration file, which is shared by all the scenes in the project. This file is located in the `Project` folder of the RoadRunner project.

Open the `OpenDriveAssetData.xml` file from this location, where *ProjectFolder* is the path to the folder of the RoadRunner project. For more details on the project folder layout, see "RoadRunner Project and Scene System" on page 2-2.

ProjectFolder/Project/OpenDriveAssetData.xml

Explore File Structure

The `OpenDriveAssetData.xml` file has a top-level element, `OpenDriveAssetData`, containing `Objects`, `Signals`, and `Markings` elements that specify props, signals, and lane markings, respectively.

This XML code shows a template of the `OpenDriveAssetData.xml` file, and lists the significance of the element tags. The `Type` and `FilePath` are required fields for `Objects`, `Signals`, and `Markings` tags, whereas others are optional fields that may only be required during import. For more details about these elements, see "ASAM OpenDRIVE Representations" on page 5-26.

```
<?xml version="1.0"?>
<OpenDriveAssetData>
    <Objects>
        ...
        <Object>
            <Type> OpenDRIVE "type" </Type> (Required for Export-Optional for Import)
            <Id> OpenDRIVE object "id" </Id> (Optional-only used for Import)
            <Name> OpenDRIVE object "name" </Name> (Optional-only used for Import)
            <Radius> OpenDRIVE object "radius" </Radius> (Optional-only used for Import)
            <Height> OpenDRIVE object "height" </Height > (Optional-only used for Import)
            <FilePath> Relative file path to RoadRunner asset </FilePath> (Required)
        </Object>
        ...
```

```
        </Objects>
        <Markings>
            ...
            <RoadMark>
                <Type> OpenDRIVE "type" </Type> (Required for Export-Optional for Import)
                <Color> OpenDRIVE "color" </Color> (Optional)
                <FilePath> Relative file path to RoadRunner asset </FilePath> (Required)
            </RoadMark>
            ...
        </Markings>
        <Signals>
            ...
            <Signal>
                <Type> OpenDRIVE "type" </Type> (Required for Export-Optional for Import)
                <SubType> OpenDRIVE "subtype" </SubType> (Optional)
                <Id> OpenDRIVE signal "id" </Id> (Optional-only used for Import)
                <Name> OpenDRIVE signal "name" </Name> (Optional-only used for Import)
                <Country> OpenDRIVE signal "country" </Country> (Optional-only used for Import)
                <Value> OpenDRIVE signal "value" </Value> (Optional-only used for Import)
                <FilePath> Relative file path to RoadRunner asset </FilePath> (Required)
                <Variant> Variant of RoadRunner signal/sign asset (integer, where 0 is the first variant, 1 is the second, et
            </Signal>
        ...
        </Signals>
</OpenDriveAssetData>
```

Configure Assets for Export

Use this process to configure assets for export:

1   Open the `OpenDriveAssetData.xml` file in a text editor. When you create a new project, the associated `OpenDriveAssetData.xml` file contains template code that you can modify.

2   Add corresponding `Object`, `Signal`, or `Marking` entries in the configuration file for unmapped props, signs, signals, or markings, and for additional assets that you have added to your scene in the RoadRunner canvas.

For example, after adding a `Drum01` (prop), `ContinentalCrosswalk` (marking), and `Sign_CrossRoadAhead` (signal) to a scene, the `OpenDriveAssetData.xml` configuration file has this structure:

```
<?xml version="1.0"?>
<OpenDriveAssetData>
    <Objects>
        ...
        <Object>
            <Type>obstacle</Type>
            <FilePath>Props/TrafficControl/Drum01.fbx_rrx</FilePath>
        </Object>
        ...
    </Objects>
    <Markings>
        ...
        <RoadMark>
            <Type>broken</Type>
            <FilePath>Markings/DashedSingleWhite.rrlms</FilePath>
        </RoadMark>
        ...
    </Markings>
    <Signals>
        ...
        <Signal>
            <Type>Sign_CrossRoadAhead</Type>
            <FilePath>Signs/Sign_CrossRoadAhead.svg_rrx</FilePath>
        </Signal>
    ...
    </Signals>
</OpenDriveAssetData>
```

Use `obstacle` as the `Type` for the `Drum01` asset. To determine the standard `Type` values for other assets, refer to ASAM OpenDRIVE `1.4 (or 1.5) Object Type` specifications.

To determine the `FilePath` value for `Drum01`, navigate to the **Library Browser** in RoadRunner. Under the `Props` folder, select the `TrafficControl` folder and click `Drum01`. In the **Attributes** pane, the text `Drum01.fbx_rrx` next to the **Prop Model** label is the file name for the prop `Drum01`.The relative file path is constructed by navigating through the folders `Props` and `TrafficControl` to locate the file `Drum01.fbx_rrx`. Hence, the value for the `FilePath` tag for `Drum01` is `/Props/TrafficControl/Drum01.fbx_rrx`.
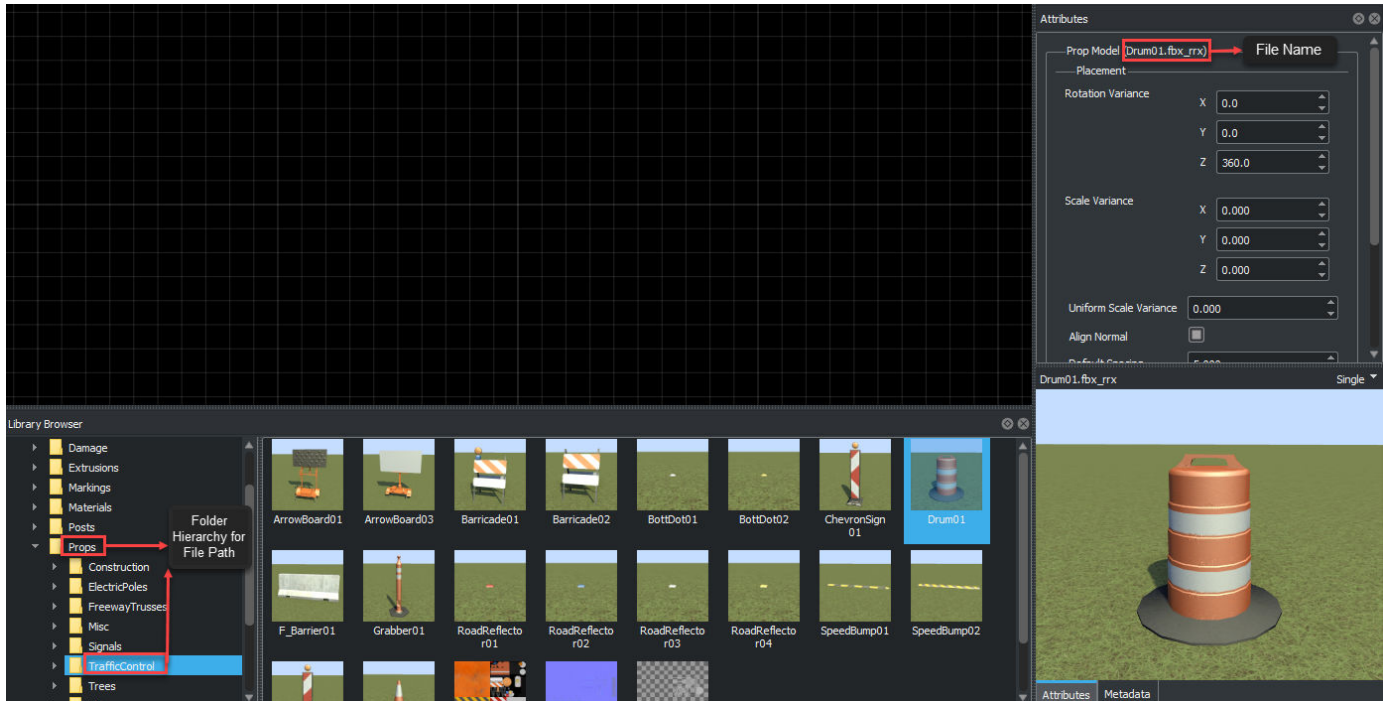
3   Save your project and export it to an ASAM OpenDRIVE file. You do not need to restart RoadRunner after creating or modifying the `OpenDriveAssetData.xml` file.

To verify that the object `Drum01` has been exported correctly in the ASAM OpenDRIVE format, select the **OpenDRIVE Export Preview Tool**
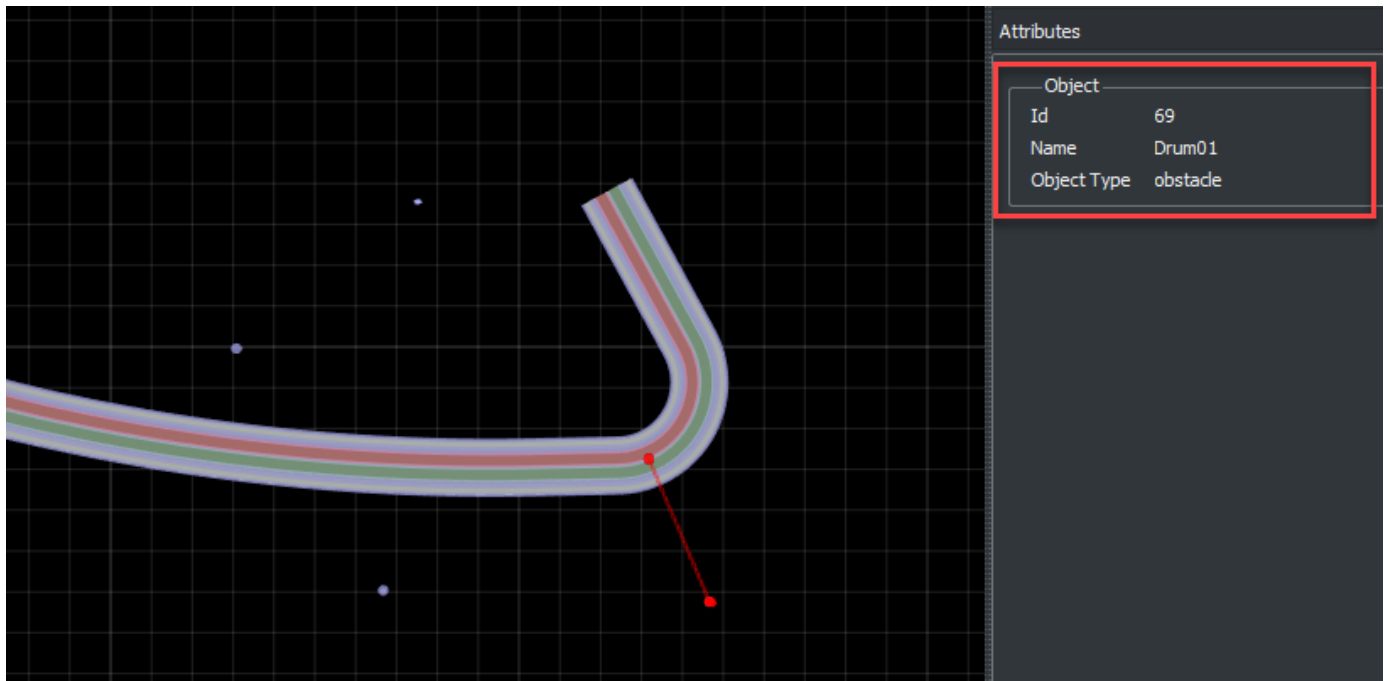
The **Object Type** is `obstacle`, which implies that the RoadRunner asset has been mapped correctly to the ASAM OpenDRIVE representation.
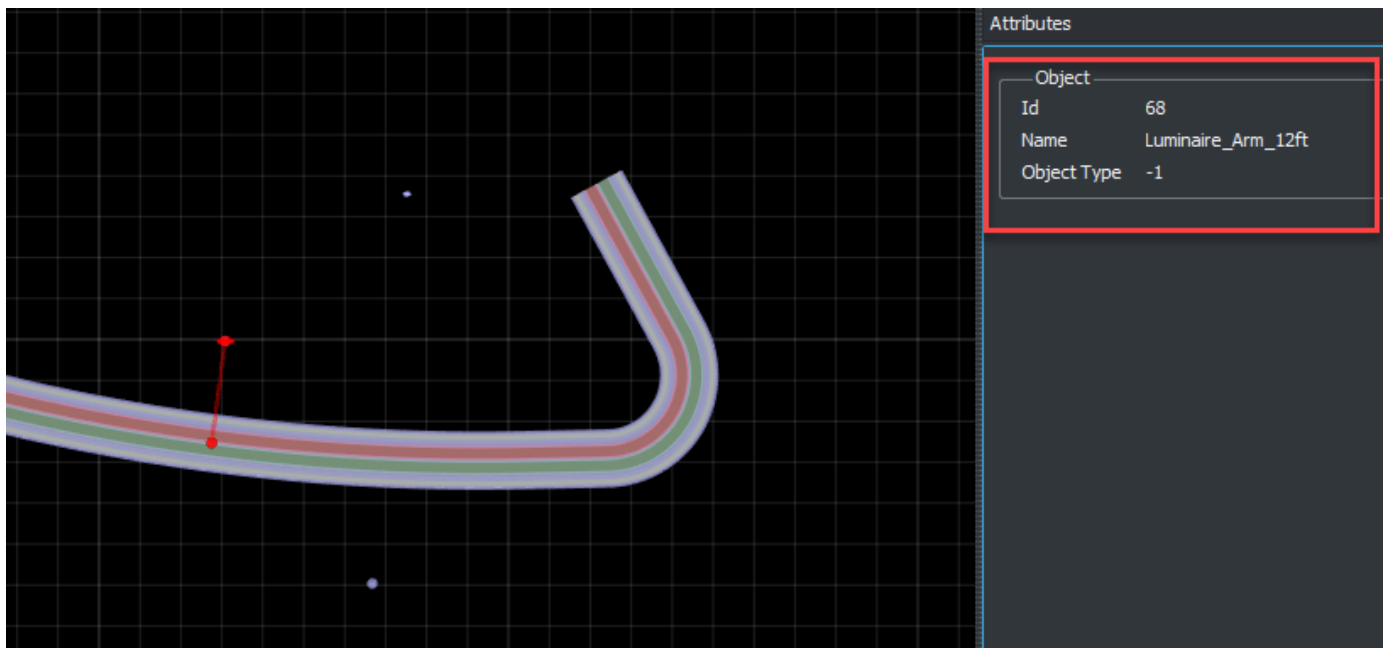
If a prop or a signal is missing in the `OpenDriveAssetData.xml` file, its **Object Type** is `-1` in the OpenDRIVE Export Preview Tool. For example, if you add the prop `Luminaire_Arm_12ft` to the scene editing canvas, but not to the configuration file, the **Object Type** displays as `-1` in the OpenDRIVE Export Preview Tool because it is not configured in the corresponding `OpenDriveAssetData.xml` file.

For information on how to configure `<Objects>`, `<Markings>`, and `<Signals>` interactively using the Asset Mapping dialog box, see "Configure Asset Mapping File Interactively" on page 5-20.

Configure Imported Assets

The ASAM OpenDRIVE import option in RoadRunner uses an `OpenDriveAssetData.xml` configuration file to convert ASAM OpenDRIVE data to the internal road format by mapping ASAM OpenDRIVE representations to RoadRunner assets. For more details about importing an ASAM OpenDRIVE file into RoadRunner, see "Importing ASAM OpenDRIVE Files" on page 3-2.

An `object`, `roadMark`, or `signal` defined in the imported ASAM OpenDRIVE file may not have a `type` value specified or may have an undefined `type` value of `-1`. In this case, you can use other attributes such as `name`, `height`, and `radius` to correlate ASAM OpenDRIVE representations to RoadRunner assets.

For example, this imported ASAM OpenDRIVE snippet shows that `type=-1` for `object id=21`.

```
<?xml version="1.0" encoding="UTF-8"?>
<OpenDRIVE>
    ...
        <objects>
            <object id="21" type="-1" name="post" height="3.65" radius="0.05"/>
        </objects>
    ...
</OpenDRIVE>
```

Use the `name` attribute in the `OpenDriveAssetData.xml` configuration file to identify the `object`.

```
<Object>
        <Name>post</Name>
        <Height>3.65</Height>
        <FilePath>Props/Signals/Signal_Post_12ft.fbx</FilePath>
</Object>
```

Note The `OpenDriveAssetData.xml` file is case-sensitive. The mappings are expected to be upper-case, for example, `<Type>pole</Type>`.

Note that this is different than the lower-case attributes in ASAM OpenDRIVE files, for example `type="pole"`.

For information on how to configure `<Objects>`, `<Markings>`, and `<Signals>` interactively using the Asset Mapping dialog box, see "Configure Asset Mapping File Interactively" on page 5-20.

Configure Asset Mapping File Interactively

In the **Assets** menu, select Asset Mapping to open the Asset Mapping dialog box. set **Select Map type:** to ASAM OpenDRIVE. This enables the **Objects**, **Markings**, and **Signals** tabs, which you can use to interactively map assets by updating the `OpenDriveAssetData.xml` configuration file. For more information on the configuration file, see "Explore File Structure" on page 5-16.
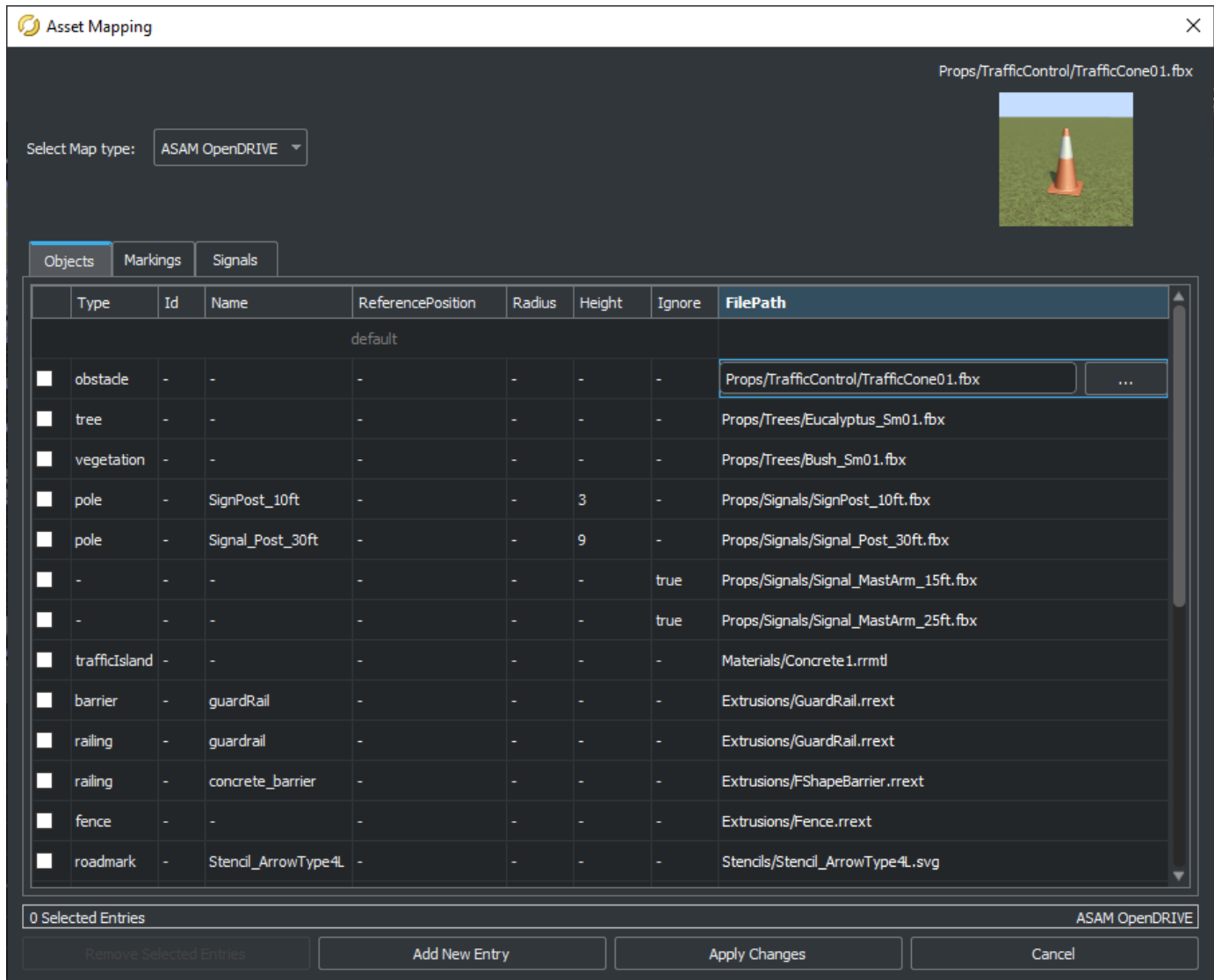
In each tab, you can click an entry to select it or double-click it to edit it.

Double-click an entry in the **FilePath** column to enable a **...** button. Click the button to browse to and specify assets from the `Assets folder`.

To configure objects interactively, select the **Objects** tab and edit these table entries.

- **Type** — Object type, based on the valid types specified by the ASAM OpenDRIVE Map service.
- **Id** — Object ID, based on the valid IDs specified by the ASAM OpenDRIVE Map service.
- **Name** — Object name, based on the valid names specified by the ASAM OpenDRIVE Map service.
- **ReferencePosition** — Object reference position, specified by the ASAM OpenDRIVE Map service.
- **Radius** — Object radius, specified by the ASAM OpenDRIVE Map service.
- **Height** — Object height, specified by the ASAM OpenDRIVE Map service.
- **Ignore** — If `true`, RoadRunner ignores the object while exporting the scene. Otherwise, RoadRunner includes the object while exporting.

- **FilePath** — Path to the asset file used to render the object. This path is relative to the `Assets` folder of the RoadRunner project.

To configure road markings interactively, select the **Markings** tab and edit these table entries.

- **Type** — Road marking type, based on the valid types specified by the ASAM OpenDRIVE Map service.
- **Color** — Road marking color, based on the valid colors specified by the ASAM OpenDRIVE Map service.
- **FilePath** — Path to the asset file used to render the road marking. This path is relative to the `Assets` folder of the RoadRunner project.

To configure signals interactively, select the **Signals** tab and edit these table entries.

- **Type** — Signal type, based on the valid types specified by the ASAM OpenDRIVE Map service.
- **SubType** — Signal subtype, based on the valid subtypes specified by the ASAM OpenDRIVE Map service.
- **Id** — Signal ID, based on the valid IDs specified by the ASAM OpenDRIVE Map service.
- **Name** — Signal name, based on the valid names specified by the ASAM OpenDRIVE Map service.
- **ReferencePosition** — Signal reference position, specified by the ASAM OpenDRIVE Map service.
- **Country** — Country name, based on the valid names specified by the ASAM OpenDRIVE Map service.
- **Variant** — Signal variant, based on the valid variants specified by the ASAM OpenDRIVE Map service.
- **Value** — Signal value, specified by the ASAM OpenDRIVE Map service.
- **FilePath** — Path to the asset file used to render the signal. This path is relative to the `Assets` folder of the RoadRunner project.

Note For each tab, you can map a default asset. Use the `default` entry in the table to specify the path to the asset.

When you select an entry in the table, the Asset Mapping dialog box displays a preview of that asset. If the file path for the asset is invalid, the preview does not display and the path entry renders in red.

Note Previews are not displayed for invalid file paths and these paths are represented red.

You can select multiple entries in the table by clicking the check box next to each entry you want to select. Selecting entries enables the **Remove Selected Entries** button. Click this button to remove the selected entries. You can also add new entries for mapping assets by clicking **Add New Entry**.

Save the customized asset mappings to the `OpenDriveAssetData.xml` configuration file by clicking **Apply Changes**.

You can discard the changes by clicking **Cancel**. Clicking **Cancel** returns a prompt confirming whether you want to apply the changes, discard the changes, or cancel the operation and return to asset mapping.

See Also
OpenDRIVE Viewer Tool | OpenDRIVE Export Preview Tool

More About
- "Importing ASAM OpenDRIVE Files" on page 3-2
- "Export to ASAM OpenDRIVE" on page 5-24

External Websites
- ASAM OpenDRIVE

Export to ASAM OpenDRIVE

RoadRunner can export scenes to the ASAM OpenDRIVE (.xodr) file formats.

ASAM OpenDRIVE Overview

RoadRunner can export scenes to OpenDRIVE 1.4, OpenDRIVE 1.5, and ASAM OpenDRIVE 1.6 formats. The ASAM OpenDRIVE export option exports an ASAM OpenDRIVE (.xodr) file containing the layout of the scene and an associated MathWorks GeoJSON file.

Export to ASAM OpenDRIVE

From the menu, select **File > Export > ASAM OpenDRIVE (.xodr)**.

To preview the ASAM OpenDRIVE export and explore exported data interactively, use the **OpenDRIVE Export Preview Tool**.

Export Options

The Export ASAM OpenDRIVE dialog box has several options to conform to various simulator needs.

Options	Description
Version	Specifies the ASAM OpenDRIVE file versions that you can select for export. Select the value as: • OpenDRIVE 1.4 to export the scene to OpenDRIVE 1.4 file version. • OpenDRIVE 1.5 to export the scene to OpenDRIVE 1.5 file version. • ASAM OpenDRIVE 1.6 to export the scene to ASAM OpenDRIVE 1.6 file version.
Database Version	A user-defined identifier for the exported scene. Useful for versioning exports of the same scene.
Database Name	A user-defined name for the exported scene.
Driving Side	A hint to the exporter for the driving side of the scene. Travel direction is explicitly defined in RoadRunner using the **Lane Travel Direction**. The travel direction in ASAM OpenDRIVE is implicit based on the country and side of the road.
Enforce connected road continuity	Select this attribute to enforce continuity between the reference lines of roads connected end-to-end. This option is enabled by default.
Export markings as <line>	Select this attribute to export additional lane marking data (spacing, dash length, and individual paint strip widths).

Options	Description
Export signals	Select this attribute to export all signals and signs mapped to junctions as `<signal>` entries. This selection applies only to signals and signs that have been associated with junctions (using the **Signal Tool**). Refer to the Traffic Signals and Signs on page 5-35 section.
Export objects	Select this attribute to export all props as `<object>` entries. Refer to "Convert Asset Data Between RoadRunner and ASAM OpenDRIVE" on page 5-16 .
Export hOffset relative to orientation	Select this attribute to export the `<hOffset>` (heading offset) values of `<signal>` entries as being relative to `<orientation>`, which is the direction of travel of the road that the signal applies to. By default, the heading offset is relative to the heading of the road, regardless of its direction of travel
Export conflict points	Select this attribute to export an `<object>` entry for every point in a junction where two roads intersect.
Export scene origin reference	Select this attribute to export a point at 0,0 in the scene. This point enables a connection between FLT or IVE files and the exported ASAM OpenDRIVE file. This point is contained in a `<road>` entry with no lanes that is positioned at the far left edge of the scene, nested within an `<object>` entry.
Clamp distances (prevents very short roads)	Select this attribute to clamp distances in the RoadRunner scene to be a multiple of 1 cm to prevent very short roads. **Note** This clamping is performed on the scene itself, so it can cause very small changes to the roads in the scene.
Export OpenCRG and Synthetic OpenCRG Options	Select this option to export road surface data assigned to different road segments to the ASAM OpenCRG file format. For more information, see "Export to ASAM OpenCRG" on page 5-48.

Options	Description
Road Data Format	Specifies the road data format for the output ASAM OpenCRG file. Select from these options: • LRFI (default) — Long, real, formatted, interchangeable data format • LDFI — Long, double, formatted, interchangeable data format **Dependencies** To enable this attribute, select the **Export OpenCRG and Synthetic OpenCRG Options** attribute.

ASAM OpenDRIVE Representations

This section describes how various types of RoadRunner objects are represented in ASAM OpenDRIVE. The **Attributes** and the **Metadata** panes contain the description for the elements in a RoadRunner scene. You can use the **Metadata** pane to add custom attributes or set an attribute value when exporting the scene to ASAM OpenDRIVE format. This section explains how the scene elements and their attributes are represented in ASAM OpenDRIVE. For information about how the metadata is exported to ASAM OpenDRIVE, see "Add Metadata to RoadRunner Scene Elements" on page 5-42 and Set OpenDRIVE Attributes Using Metadata on page 5-45.

Roads, Lanes, and Junctions

Roads, lanes, and junctions are exported to ASAM OpenDRIVE using the standard `<road>`, `<lane>`, and `<junction>` entries. Roundabouts are also composed of roads. Hence, roundabouts are exported as the roads.

For each road in a scene, RoadRunner creates one or more `<road>` entries. Whenever a road ends or a junction begins or ends, RoadRunner creates a unique `<road>` entry. ASAM OpenDRIVE `<road>` entries cannot extend through a junction, so the geometry is cut and exported as separate roads.

Note The `<shape>`, `<crossfall>`, `<surface>`, and `<railroad>` entries are not used.

ASAM OpenDRIVE 1.6 requires any two roads connected end-to-end in a scene meet exactly, indicated by an unbroken reference line between them. **Enforce connected road continuity** option, selected by default in the Export ASAM OpenDRIVE dialog box during export, ensures that connected roads in your scene fulfill this ASAM OpenDRIVE 1.6 connection requirement.

This figure shows an offset between the red reference lines of two roads that are connected end-to-end without the **Enforce connected road continuity** option enabled. The discontinuity between the reference lines does not load the scene properly after export.
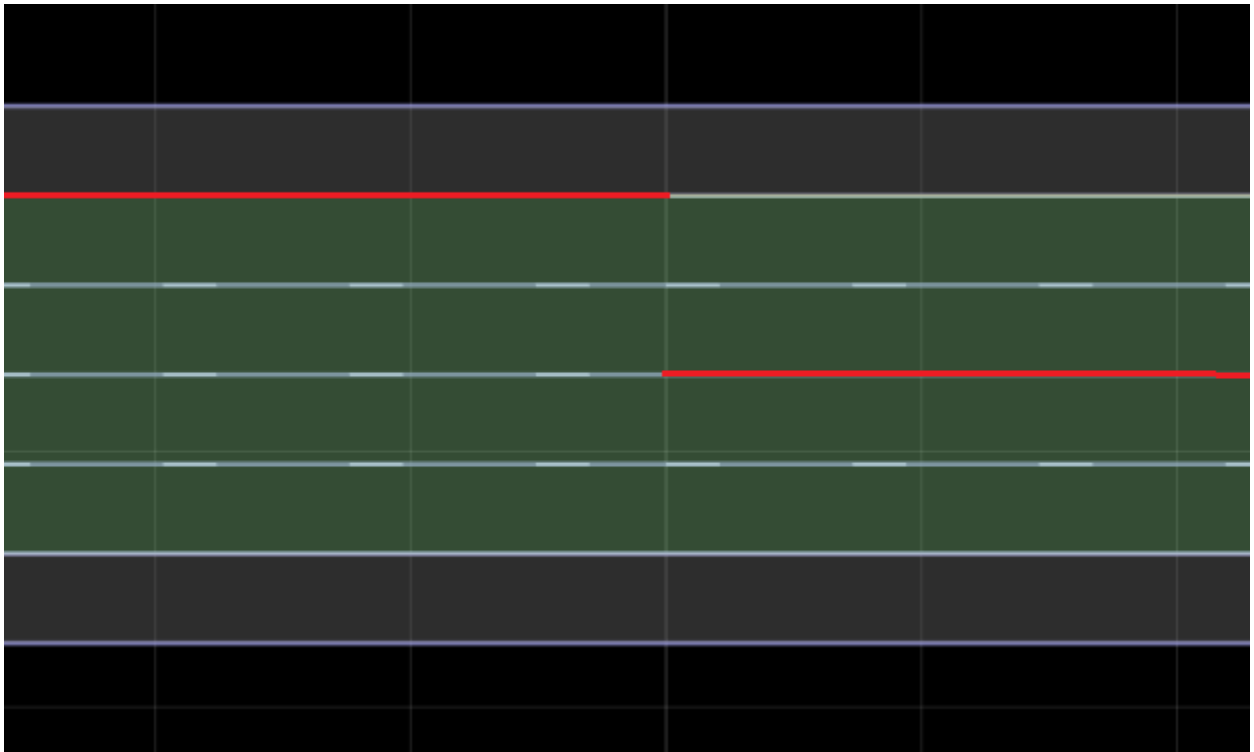
This figure shows the same two roads that are connected end-to-end with the **Enforce connected road continuity** option enabled. The **Enforce connected road continuity** option produces continuity between the red reference lines of the two roads, aligning them with one another.
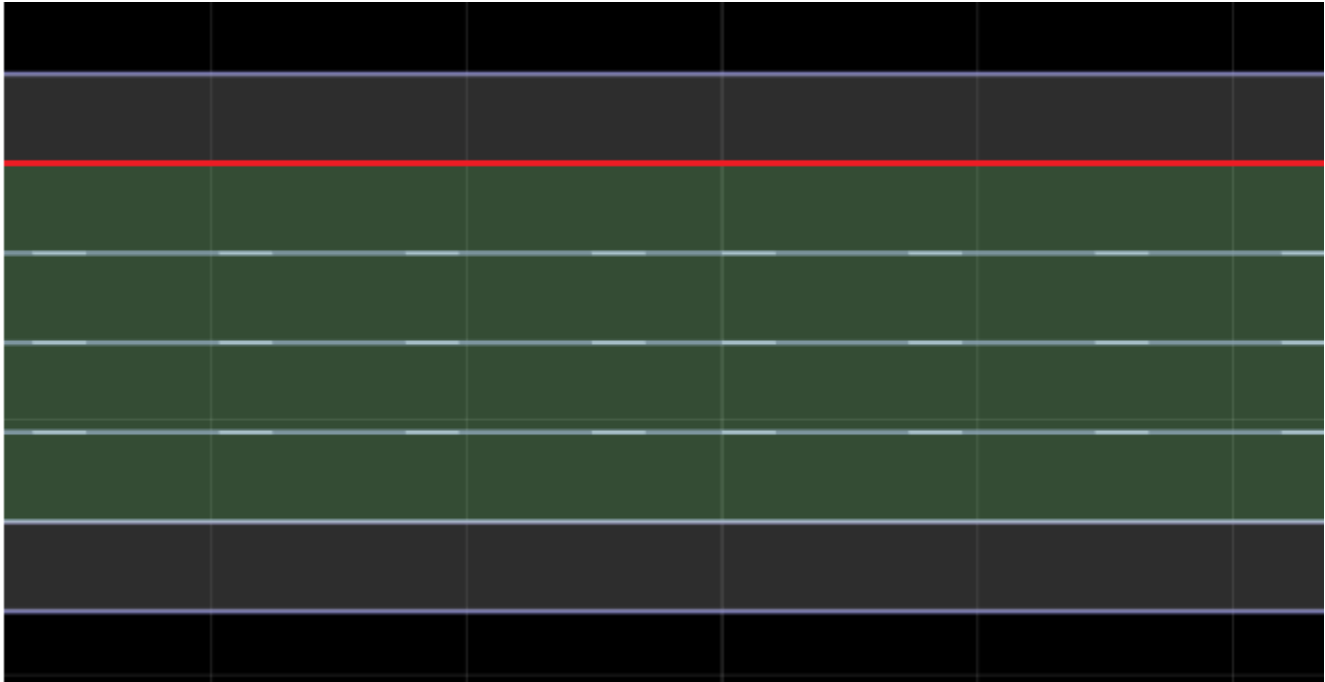
For each lane in a scene, RoadRunner creates one or more `<lane>` entries. The resulting `<lane>` entry is placed on one side or the other of the center lane, depending on its travel direction or the travel direction of neighboring lanes and the selected **Driving Side** during export. Whenever a lane starts or ends, RoadRunner creates a new `<laneSection>` entry.

Note The `level` flag in `<lane>` entries is not used. The `<height>`, `<material>`, `<visibility>`, and `<access>` entries are also not used.

For each junction in a scene, RoadRunner creates a `<junction>` entry. RoadRunner exports some junctions as one `<junction>` entry due to overlapping maneuver roads or corners. A connecting `<road>` entry is exported for each maneuver road in each junction. Where possible, the exporter prefers the geometry and lane markings of nonmaneuver roads that extend through the junction. The resulting geometry of each connecting road might be the combination of multiple maneuver and nonmaneuver roads.

Note The `<priority>` entry is not used.

Props
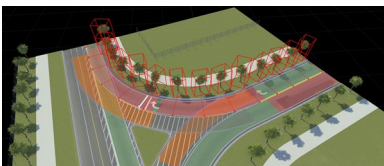
Note Prop polygons are not exported, but if you run the **bake** operation to convert them to points, you can export them in point format. See **Prop Polygon Tool**.

With the exception of traffic signals and signs (see below), point props are exported as ASAM OpenDRIVE `<object>` instances. The exported prop includes sufficient information to identify the prop type and the oriented bounding box (OBB) of the prop model.

In ASAM OpenDRIVE, objects are stored on roads. The position and orientation of a given object depends on the geometry of the road it is assigned to. RoadRunner props are freely positioned in the world, so the export process must choose a road for each prop to export. In most cases, RoadRunner selects the road closest to the prop.

Note In some cases, it is impossible to represent a prop position in ASAM OpenDRIVE. In this image, the bush on the right is past the end of the road and there is no other road in the scene. In this case, the prop is not exported and a warning is displayed during export.
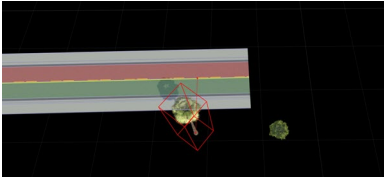
In some cases, it is unclear where to best place an imported `<object>` or which dimensions of a prop should be exported (for example, a tree trunk's width or height may be desirable over the full size of the tree including the leaves). Similarly, in some cases, it is unclear where to place an imported guardrail, wall, fence `<outline>` or `<repeat>` (for example, does the `<repeat>` define the top or the bottom of the fence). To better define the import and export position, a `<ReferencePosition>` entry can be added to the `OpenDriveAssetData.xml` file per asset. For more details about the `OpenDriveAssetData.xml` file, see "Convert Asset Data Between RoadRunner and ASAM OpenDRIVE" on page 5-16.

The `<ReferencePosition>` entry has two possible values:

- `FitToBoundingBox`
- `UseOrigin`

The `FitToBoundingBox` is selected by default. This figure shows the `<ReferencePosition>` tag set to `FitToBoundingBox` during export for props. When `<ReferencePosition>` is set to `FitToBoundingBox` for props during export, the `<object>` is created with the position of the `<object>` at the bottom center of the resulting bounding box.
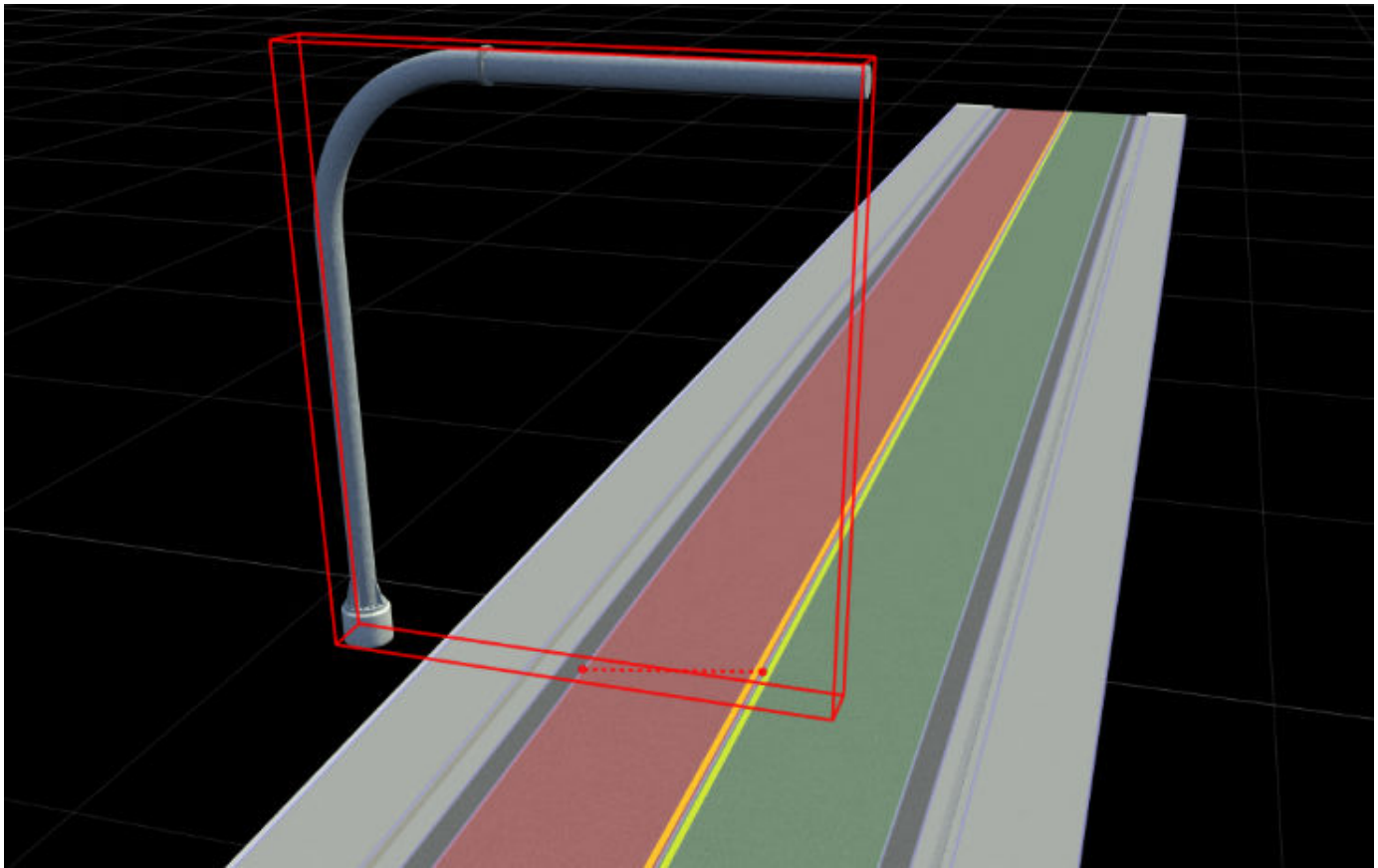
During import, when the `<ReferencePosition>` is set to `FitToBoundingBox`, the prop is placed centered in the bounding box. When the `<ReferencePosition>` is set to `FitToBoundingBox` for extrusions during export, the `<object>` is created with `<outline>` that follows the bottom center of the extrusion shape. During import, the prop curve is created such that the bottom center of the extrusion shape follows the `<object><outline>` or `<repeat>`. This figure shows a guardrail extrusion imported with the `FitToBoundingBox` option set for `<ReferencePosition>`.
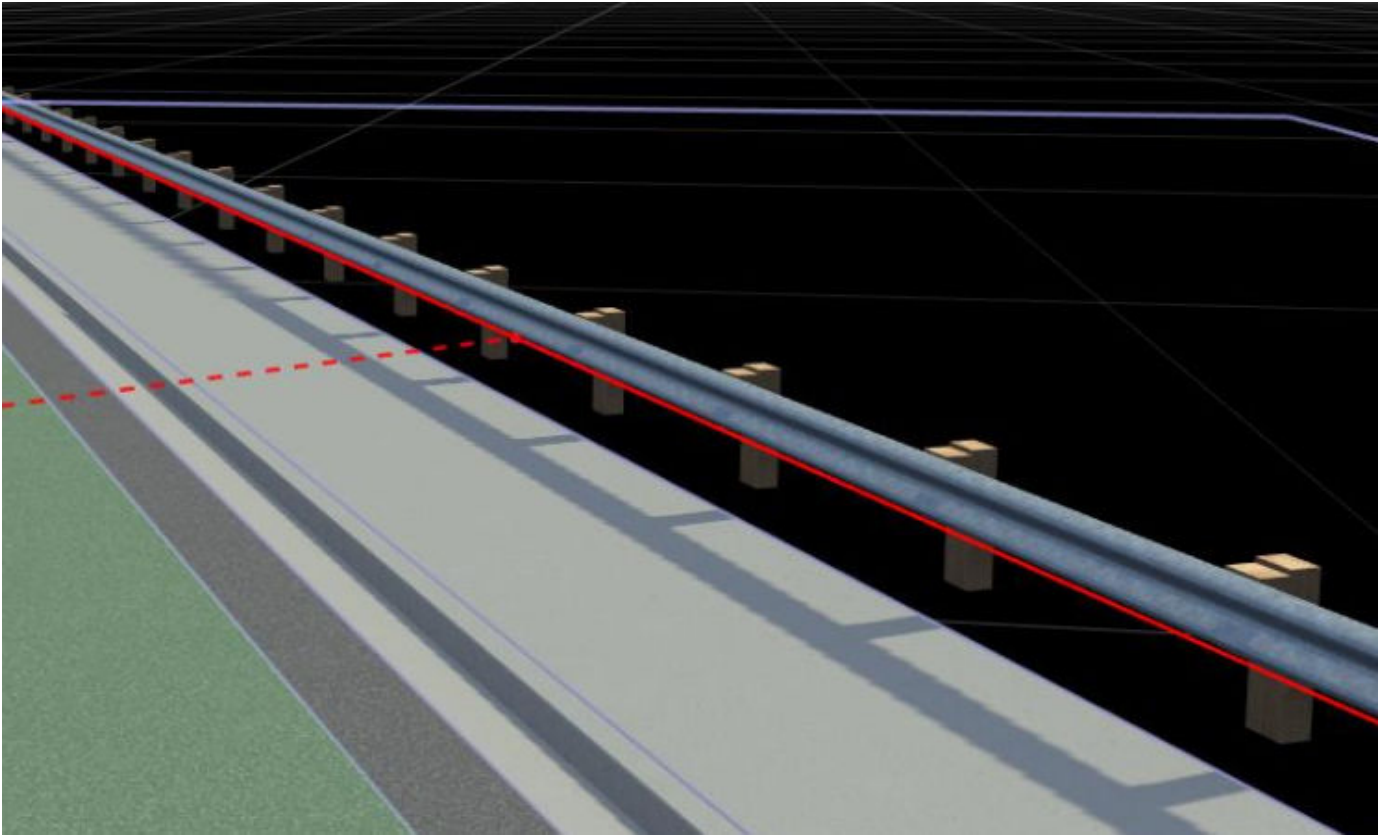
When the `<ReferencePosition>` is set to `UseOrigin` for props during export, the `<object>` is created with the position of the `<object>` positioned at the bottom of the prop. The width and length values are not exported. Only the height value is exported. The radius value is optionally exported if specified in the `OpenDriveAssetData.xml` file for that prop. This image shows a prop exported using the `UseOrigin` value.
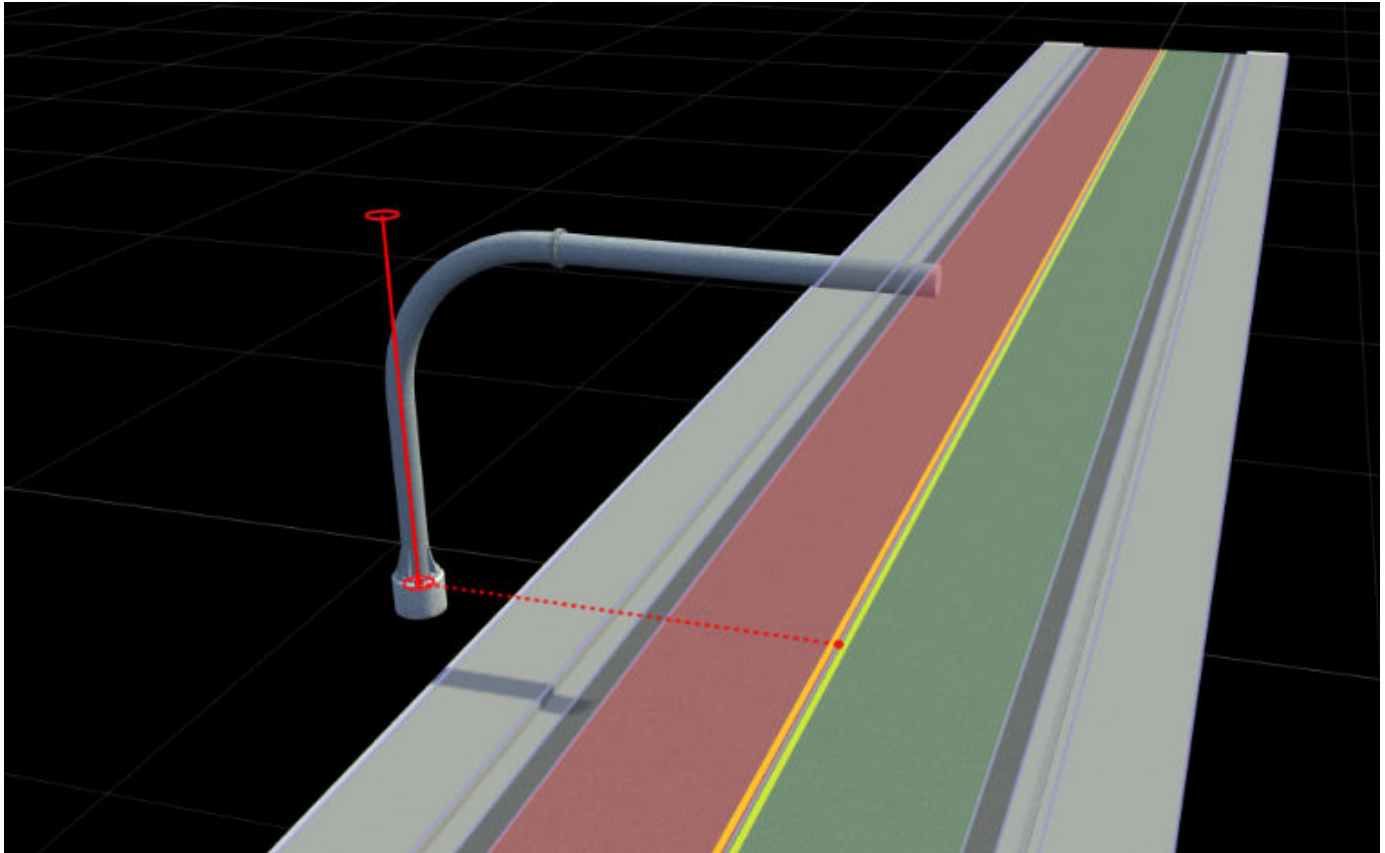
When the <ReferencePosition> is set to `UseOrigin` for extrusions during export, <object> is created with <outline> following the prop curve path. During import, the prop curve is created following the <object> <outline> or <repeat>. This image shows a guardrail extrusion imported using the `UseOrigin` value.
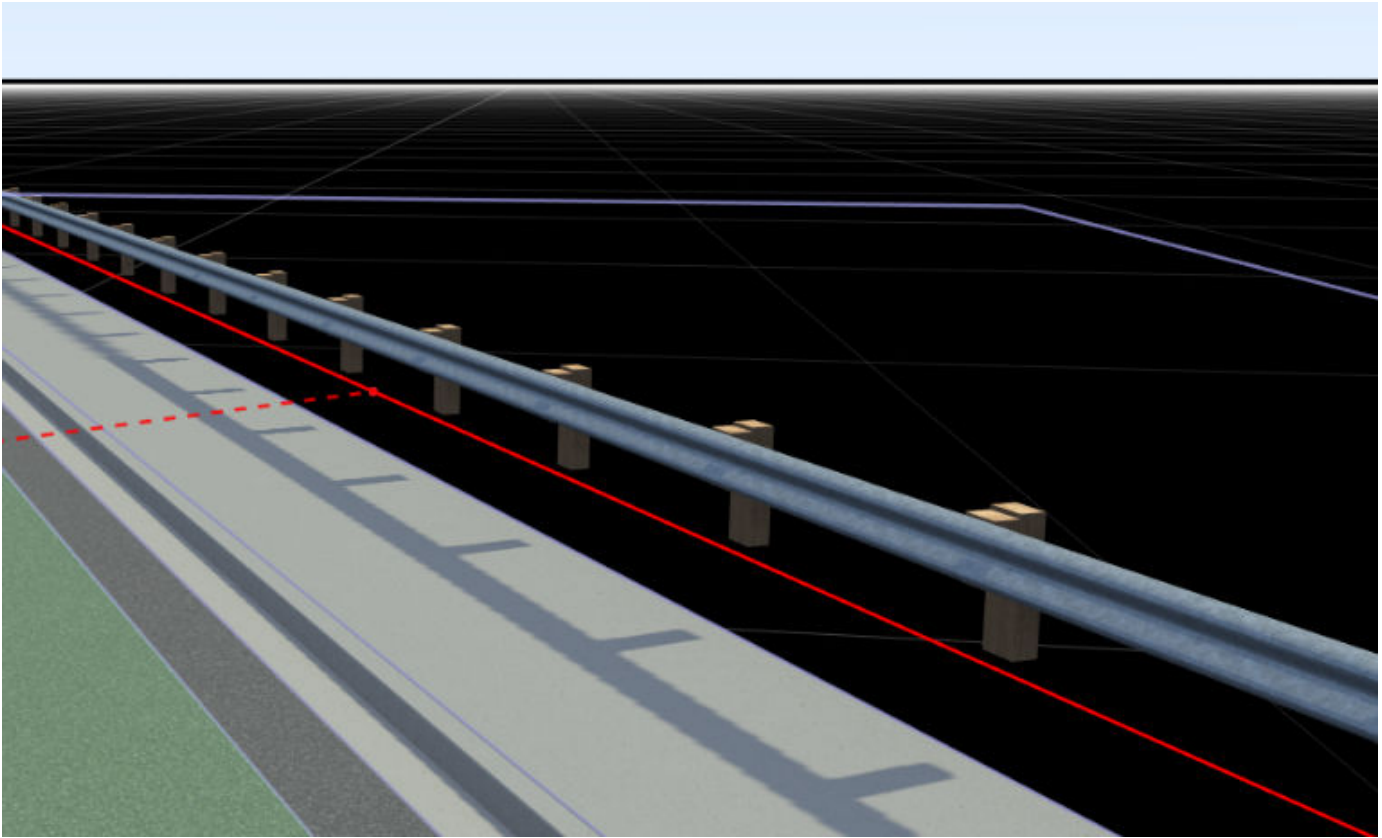
Prop Attributes

Exported props include the following attributes:

ASAM OpenDRIVE Attribute	Description
name	Name of the prop asset (for example, "Signal_Post_30ft")
s/t	Inertial position of the prop point
hdg/roll/pitch	Inertial rotations of the prop point
zOffset	Relative height of the prop point
height/width/length	Dimensions of the prop model's bounding box
type	Object type, as defined by the configuration XML file for the point's asset (refer to "Convert Asset Data Between RoadRunner and ASAM OpenDRIVE" on page 5-16)

Crosswalks and Marking Polygons
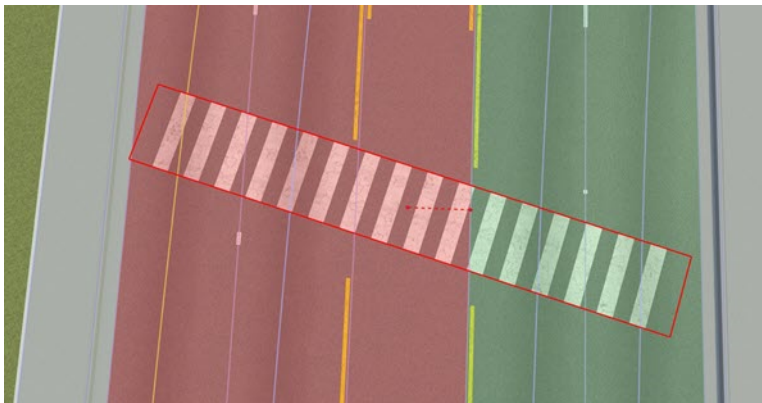
Crosswalks and marking polygons are exported as ASAM OpenDRIVE `<outline>` objects, similar to the crosswalk example in section 7.4 of the OpenDRIVE 1.5M specification.

Unlike that example, RoadRunner exports the polygon vertices as `<cornerLocal>` objects (rather than `<cornerRoad>` objects), which means that the vertices are defined relative to the pivot point specified in the attributes of the `<object>` parent.

This example code shows the representation of the crosswalk polygon in the previous image.

```
<object id="162" name="ContinentalCrosswalk" s="5.9095723267801631e+1" t="1.7834630409170869e+0" zOffset="1.907348632812!
    <outline>
        <cornerLocal u="-8.0157129245630365e+0" v="1.4628157932607735e+0" z="0.0000000000000000e+0"/>
        <cornerLocal u="8.0819274304890261e+0" v="1.0363072624453764e+0" z="-1.9073486328125000e-6"/>
        <cornerLocal u="8.0157129245556504e+0" v="-1.4628157932298933e+0" z="-1.9073486328125000e-6"/>
        <cornerLocal u="-8.0819274304964104e+0" v="-1.0363072624144962e+0" z="0.0000000000000000e+0"/>
        <cornerLocal u="-8.0157129245630365e+0" v="1.4628157932607735e+0" z="0.0000000000000000e+0"/>
    </outline>
</object>
```

Crosswalks and Marking Polygon Attributes

Exported crosswalks and marking polygons include the following attributes.

ASAM OpenDRIVE Attribute	Description
name	Name of the marking (for example, `"ContinentalCrosswalk"`)
s/t	Inertial position of the pivot point
hdg/roll/pitch	Inertial rotations of the pivot point
zOffset	Relative height of the pivot point
height/width/length	Dimensions of an oriented bounding box fit to the polygon's vertices. The `'width'` is treated as the dimension along the road, and the `'length'` is treated as the dimension across the road.
type	Object type, as defined by the configuration XML file for the marking's asset (refer to "Convert Asset Data Between RoadRunner and ASAM OpenDRIVE" on page 5-16)

Traffic Signals and Signs
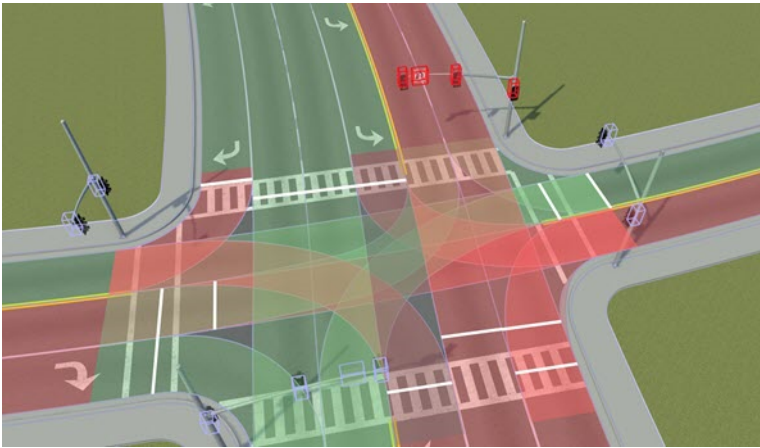
RoadRunner exports traffic signals and signs as ASAM OpenDRIVE `<signal>` objects.

For optimal behavior, traffic signals and signs for controlled intersections should be mapped to junction gates by using the **Signal Tool**. Traffic signals are exported only if they are mapped to junction gates. Signs are exported regardless if they are mapped to junction gates and are automatically mapped to the nearest road if not explicitly mapped.

If you need to add a traffic signal outside of a controlled intersection (for example, for a freeway onramp or pedestrian crossing), you can use the **Custom Junction Tool** to create a junction along a single road.

Note Traffic signals and signs within **Prop Assembly Assets** are not exported. To export signals or signs in an assembly, you must first expand the instance of the assembly.

Signals and Signal References

When a signal or sign is mapped to a junction gate, it appears in the ASAM OpenDRIVE export as a `<signal>` instance and one or more `<signalReference>` instances, where:

- `<signal>` defines the physical location of the signal. Use `<signal>` to derive the 3D location of the signal, regardless of which roads or lanes are controlled by the signal. In most cases, the signal is mapped to the closest road (similar to the approach used for "Props" on page 5-28). This mapping might have no logical association to the signal (for example, the signal could be a nearby side street).

- `<signalReference>` associates the signal to the roads and lanes that are controlled by the signal. Signal references indicate the semantic relationship between the signal and the road network (as opposed to `<signal>`, which is used purely for geometric positioning). Signal references are present for each maneuver road gate associated with the signal (through the **Signal Tool**).

Signal Attributes

Exported signals and signs include the following attributes:

ASAM OpenDRIVE Attribute	Description

`name`
 	
`s/t`
 	
`hOffset/roll/pitch`
 	
`zOffset`
 	
`height/width/length`
	Refer to Prop Attributes on page 5-33.
	
	By default, `"hOffset"` is treated the same as the `<object>` `"hdg"` attribute. However, if you select **Export hOffset relative to orientation** when exporting, then `"hOffset"` is offset from the `<orientation>` (direction of travel) of the road that the signal applies to.

`type/subtype`	Signal type and subtype, as defined by the configuration XML file for the signal's asset (refer to "Convert Asset Data Between RoadRunner and ASAM OpenDRIVE" on page 5-16)
`country`	`"OpenDRIVE"` is always used.
`dynamic`	Specify `"yes"` for dynamic junction signalization and `"no"` for static signalization (for example, `"All Go"` or `"All Stop"`)
`value`	Unused (set to `"-1"` in all cases).
`text`	Unused (set to empty string in all cases).

Parking Spaces
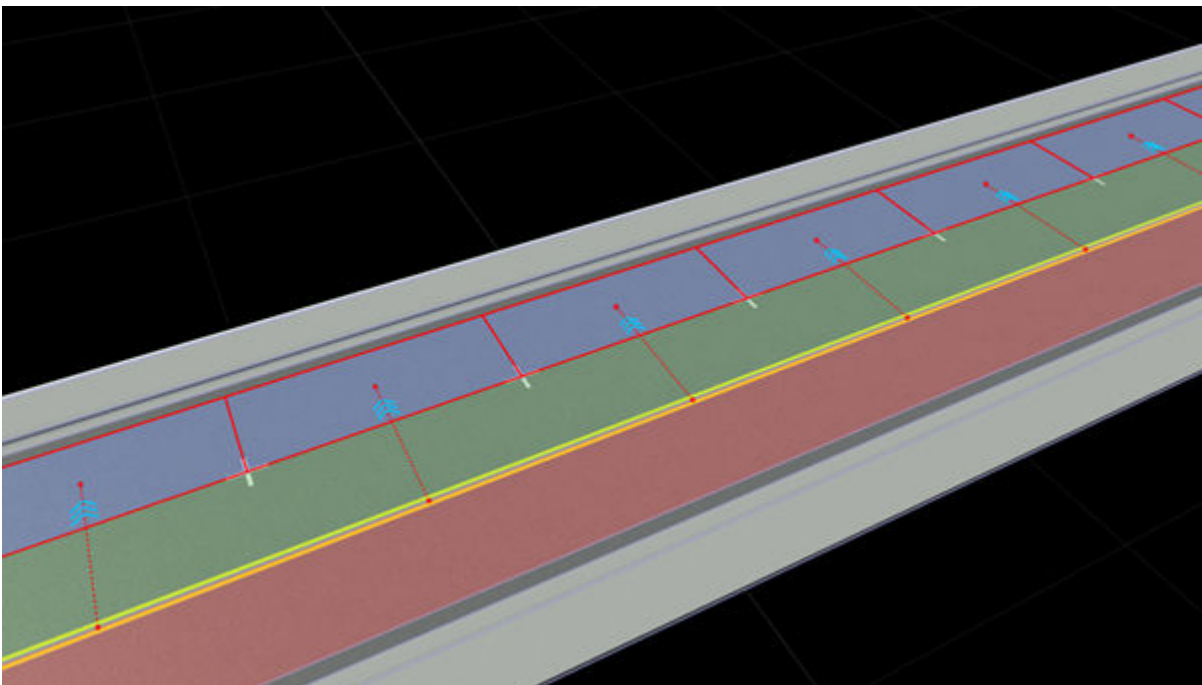
RoadRunner exports parking spaces as an `<object>` with type `"parking"` and an additional `<parkingSpace>` entry under the `<object>` following section 5.3.8.1.5 in the OpenDRIVE 1.4H specification. Markings on a parking space are exported as `<marking>` under the `<parkingSpace>` entry following section 5.3.8.1.6 in the OpenDRIVE 1.4H specification.

Parking Attributes

Exported parking spaces include the following attributes.

ASAM OpenDRIVE Attribute	Description
name s/t hdg/roll/pitch zOffset height/width/length	Refer to Prop Attributes on page 5-33.
type	Always set to `"parking"`.
side (attribute for `<marking>` entries under `<parkingSpace>`)	Side of the marking (`"left"`, `"right"`, `"front"`, or `"rear"`), where the rear is the entry point of the parking space.
type/width/color (attributes for `<marking>` entries under `<parkingSpace>`)	Same properties as `<roadMark>` entries for `<lane>`.

Limitations

General

- RoadRunner does not export lane height to ASAM OpenDRIVE.
- To export prop polygons from RoadRunner to ASAM OpenDRIVE, you must run the `'bake'` operation to convert them to points. For more details, see **Prop Polygon Tool**.
- Prop curves and props spans export to ASAM OpenDRIVE as road objects. In ASAM OpenDRIVE, the exported road objects are described by `<objects>` element. An `<outline>` element within the `<object>` element describes the shape of an exported object.
- RoadRunner does not export `<repeat>` entries to describe repeated objects. Instead, it exports an `<object>` element for every individual object, or an `<outline>` element for extrusions.
- If a lane has multiple lane predecessors and successors, RoadRunner exports only one predecessor and successor to ASAM OpenDRIVE.

Specific to OpenDRIVE 1.5 / ASAM OpenDRIVE 1.6

- If there is a lateral offset between the reference lines of two serially connected roads in a RoadRunner scene, RoadRunner does not automatically correct the lateral offset when exporting the scene to the ASAM OpenDRIVE 1.6 format.
- RoadRunner does not export a `<positionRoad>` element for a signal. Instead, it uses inertial coordinates and exports the `<positionInertial>` element to reference the deviation between the physical position and logical position of a signal.

See Also
OpenDRIVE Viewer Tool | **OpenDRIVE Export Preview Tool**

More About

- "Importing ASAM OpenDRIVE Files" on page 3-2
- "Add Metadata to RoadRunner Scene Elements" on page 5-42

External Websites

- ASAM OpenDRIVE

Left-Hand Drive Export to ASAM OpenDRIVE

Recommended Approach

If the scene being built in RoadRunner is meant to have left-hand driving, the `Driving Side` should be set to `Left`. Otherwise, it should be set to `Right`.

To set the `Driving Side` export option, open the **Export ASAM OpenDRIVE** dialog box by selecting **File > Export > ASAM OpenDRIVE (.xodr)**. For more details about export options, see "Export to ASAM OpenDRIVE" on page 5-24.

ASAM OpenDRIVE Details

The supported ASAM OpenDRIVE formats OpenDRIVE 1.4, OpenDRIVE 1.5, and ASAM OpenDRIVE 1.6 does not have a notion of lane travel direction. Instead, it is expected that all drivable lanes on one side of the road go one way and the drivable lanes on the other side of the road go the opposite way.

The supported ASAM OpenDRIVE formats does not have a notion of "driving side" (for example, left-hand driving in the UK or Japan). Instead, it is expected that the travel direction be one of these options:

- Assumed right-side (common)
- Assumed based on `<header>` country code (uncommon)
- Determined using the `<incoming>` lanes in `<junction>` entries (uncommon, difficult, and sometimes impossible if no junctions are present)
- Determined by the initial orientation of placed vehicles in a scenario (most common)

RoadRunner Export
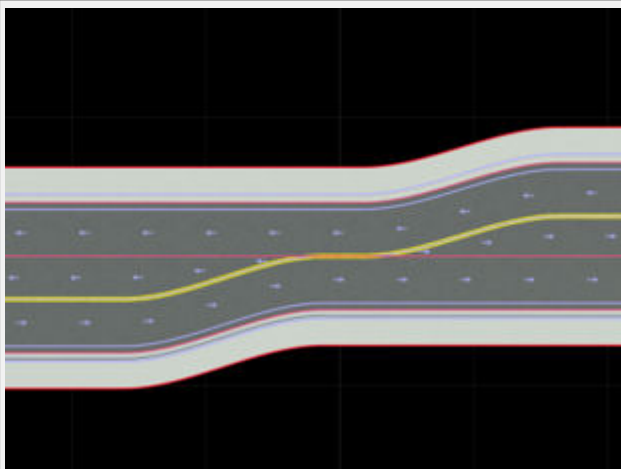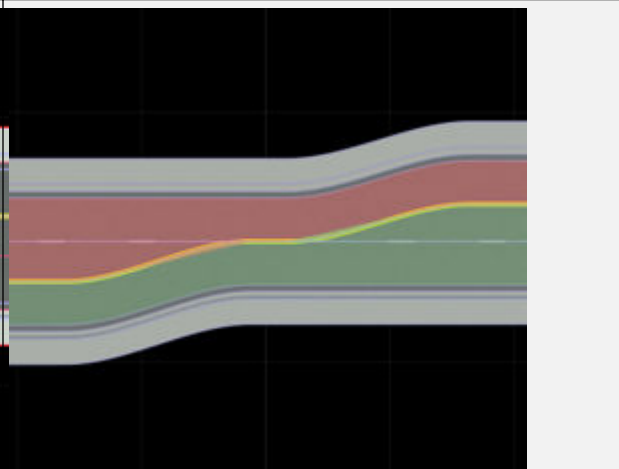
RoadRunner does the following on export for travel direction:

1 Ensures that lanes are placed on one side or the other of the ASAM OpenDRIVE road based on travel direction
2 Writes out the travel direction of the lane in `<userData>` for each lane.
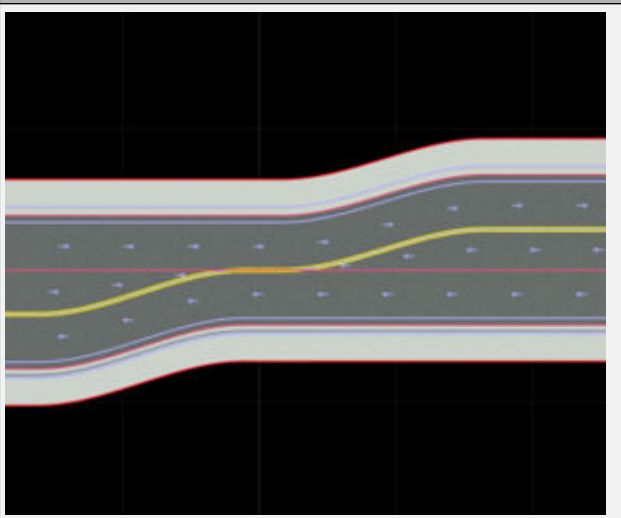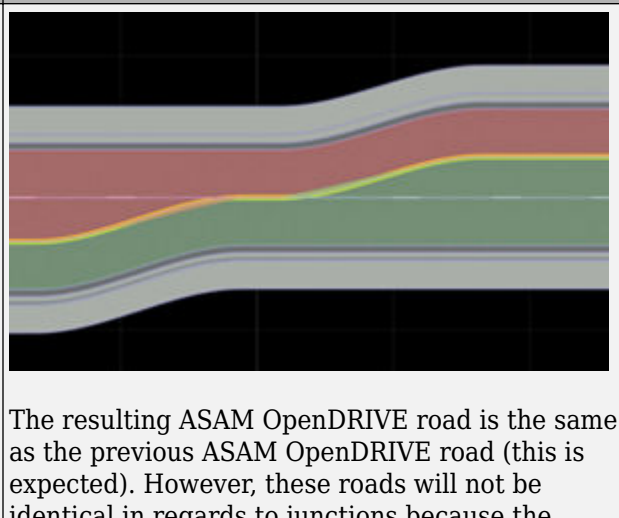
During export, lanes traveling in one direction are placed on one side of the ASAM OpenDRIVE road (regardless of the lane's original side of the road in RoadRunner), and the lanes traveling in the opposite direction are placed on the other side of the ASAM OpenDRIVE road. The `Driving Side = Left` option provides a hint to the exporter that lanes marked as `Forward` travel direction should (in general) be placed on the `Left` side of the ASAM OpenDRIVE road.

Examples
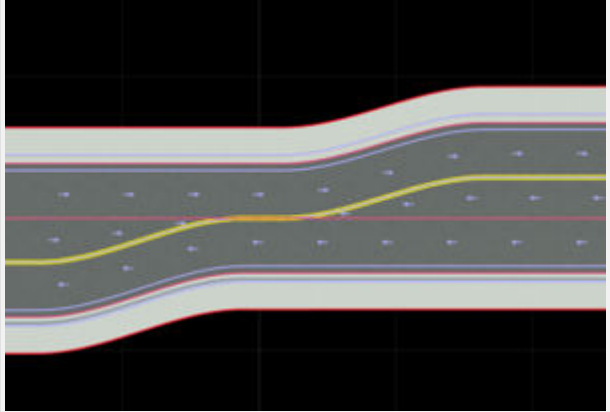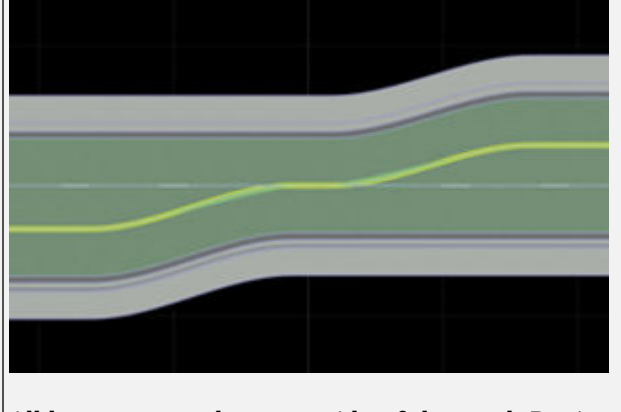
Right-Hand Driving with "Driving Side = Right"

RoadRunner Scene	Exported ASAM OpenDRIVE Scene
The forming lanes are created on either side of the center lane. Travel direction is irrespective of the side of the road.	The lanes are placed on each side (colored red and green for left and right, respectively) to adhere to the ASAM OpenDRIVE travel direction restrictions.

Left-Hand Driving with "Driving Side = Left"

RoadRunner Scene	Exported ASAM OpenDRIVE Scene
The forming lanes are created on either side of the center lane. Travel direction is irrespective of the side of the road.	The resulting ASAM OpenDRIVE road is the same as the previous ASAM OpenDRIVE road (this is expected). However, these roads will not be identical in regards to junctions because the `<incoming>` lanes will be different for the right vs. left cases.

Left-Hand Driving with "Driving Side = Right" and Right-Hand Driving with "Driving Side = Left" are Not Recommended

RoadRunner Scene	Exported ASAM OpenDRIVE Scene
  The forming lanes are created on either side of the center lane. Travel direction is irrespective of the side of the road.	 All lanes are on the same side of the road. During export, warnings like the following are printed in the **Output** pane: WARNING: Detected non-critical validation issues. Road 1 has more than two driving directions or reversed driving directions. Some OpenDRIVE importers may not interpret this data correctly.

Add Metadata to RoadRunner Scene Elements

RoadRunner enables you to add metadata to describe the elements in a scene. The metadata sets the values for one or more ASAM OpenDRIVE attributes when you export the scene to the ASAM OpenDRIVE file format. For information about how to export a RoadRunner scene to the ASAM OpenDRIVE file format, see "Export to ASAM OpenDRIVE" on page 5-24.

Add Metadata

Use the **Metadata** pane to add metadata for a road, lane, junction, signal, or object element in a RoadRunner scene.

To add metadata for an element in the RoadRunner scene, you must first select a tool and then select an element in the scene. For example, to add metadata for a road element, you must first select the **Road Plan Tool** from the RoadRunner toolbar and select a road from the scene. This enables the Add Metadata dialog box. Specify the name and the data type for the metadata by using the **Name** and **Type** parameters, respectively.
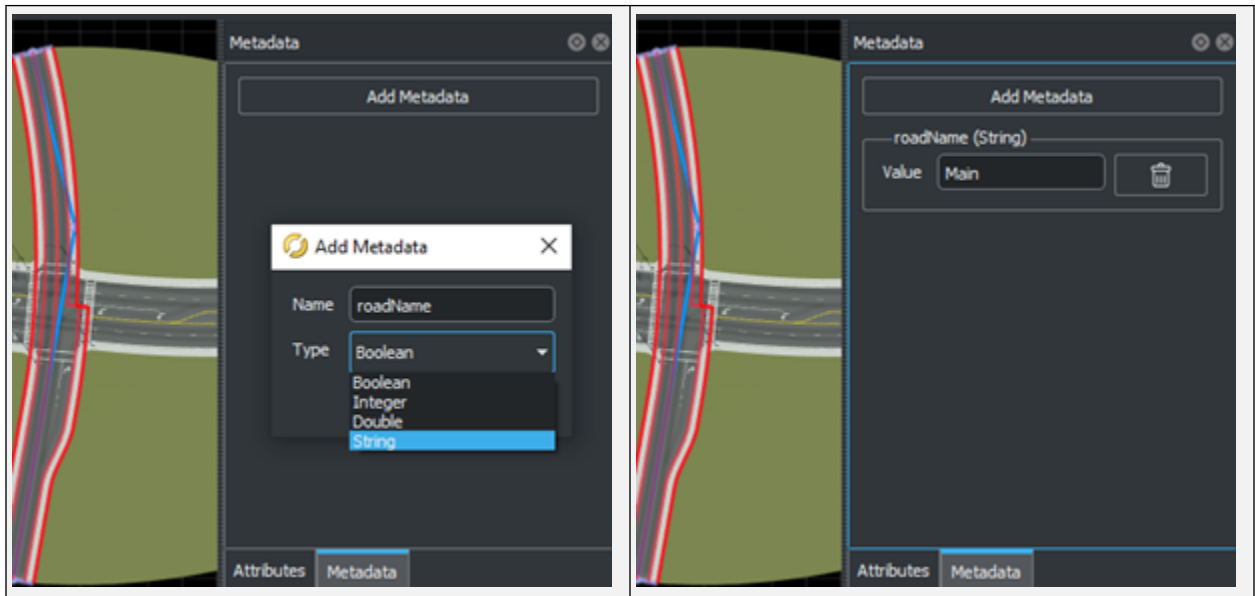
Select the data type of the metadata value from these options: `Boolean`, `Integer`, `Double`, and `String`. You can then enter the value for the new metadata by using the **Value** parameter in the **Metadata** pane.

Set Attributes

You can add metadata to define both custom attributes and attributes described in ASAM OpenDRIVE standard. The **Name** and **Value** parameters set the attribute name and the value to be exported to the ASAM OpenDRIVE file, respectively. For an example of how to set attributes by defining the metadata, see Set OpenDRIVE Attributes Using Metadata on page 5-45.

Set Custom Attributes

Custom attributes are attributes that are not described in the ASAM OpenDRIVE standard. The custom attributes defined for an element in the scene are exported to the ASAM OpenDRIVE file as ancillary data and are represented by `<userData>` element.

Set Attributes Described in ASAM OpenDRIVE Standard

You can use the metadata to set the values for the `name`, `type`, `subtype`, `country`, `value`, `unit`, and `text` attributes of `<road>`, `<junction>`, `<object>`, and `<signal>` elements. To set these attributes, you must add `OpenDRIVE_` as a prefix to the attribute names when specifying them to the **Name** parameter.

Name Specified to Name Parameter	Corresponding ASAM OpenDRIVE Attribute
`OpenDRIVE_name`	Sets the `name` attribute for the selected element.
`OpenDRIVE_type`	Sets the `type` attribute for the selected element.
`OpenDRIVE_subtype`	Sets the `subtype` attribute for the selected element.
`OpenDRIVE_country`	Sets the `country` attribute for the selected element.

OpenDRIVE_value	Sets the value attribute for the selected element.
OpenDRIVE_unit	Sets the unit attribute for the selected element.
OpenDRIVE_text	Sets the text attribute for the selected element.

Note

- Metadata specified for elements other than `<road>`, `<junction>`, `<object>`, and `<signal>` is exported to the ASAM OpenDRIVE file as custom attributes. These attributes are represented by the `<userData>` element.

- Metadata for ASAM OpenDRIVE attributes other than `name`, `type`, and `subtype` is exported to the ASAM OpenDRIVE file as custom attributes and is represented by the `<userData>` element.

- To export the metadata for prop polygons, you must first convert them to points by using the **bake** operation. Then, add metadata and export to ASAM OpenDRIVE.

See Also
OpenDRIVE Viewer Tool | OpenDRIVE Export Preview Tool

Related Examples
- Set OpenDRIVE Attributes Using Metadata on page 5-45

More About
- "Importing ASAM OpenDRIVE Files" on page 3-2

External Websites
- ASAM OpenDRIVE

Set ASAM OpenDRIVE Attributes Using Metadata

This example shows how to add metadata to elements in a RoadRunner scene and then export the scene to the ASAM OpenDRIVE file format. The metadata sets the values for ASAM OpenDRIVE attributes in the exported file. For more information about metadata values for scene elements, see "Add Metadata to RoadRunner Scene Elements" on page 5-42.

Load RoadRunner Scene

From the menu, select **File > Open Scene** to load an existing RoadRunner scene (`.rrscene`) into the scene editor.

Alternatively, you can also create a RoadRunner scene or import an ASAM OpenDRIVE file and convert the data into a RoadRunner scene.

Add Metadata for Road

1   On the RoadRunner toolbar, click the **Road Plan Tool** button. Select a road in the scene to which you want to add metadata.

2   Click **Add Metadata** in the **Metadata** panel and specify the **Name** parameter as `OpenDRIVE_name`. Set the **Type** parameter to `String`. Click **OK** and specify the **Value** parameter as `Highway`.
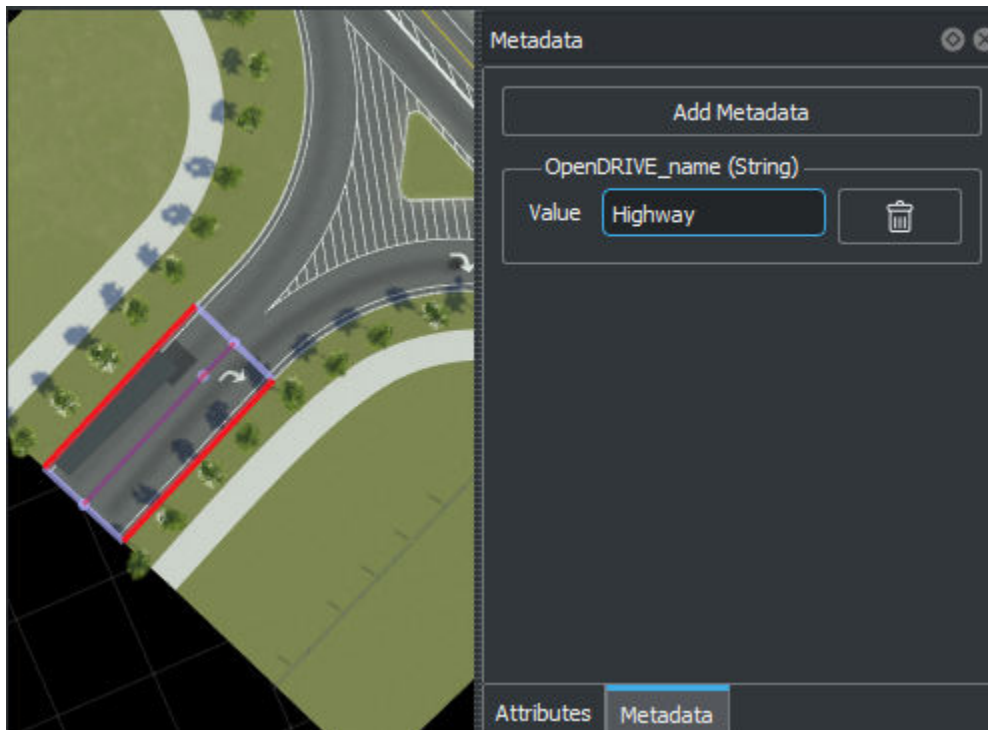
Add Metadata for Junction

1   On the RoadRunner toolbar, click the **Custom Junction Tool** button. Select a junction in the scene to which you want to add metadata.

2 Click **Add Metadata** in the **Metadata** panel and specify the **Name** parameter as `JunctionType`. Set the **Type** parameter to `String`. Click **OK** and specify the **Value** parameter as `Automatic`.
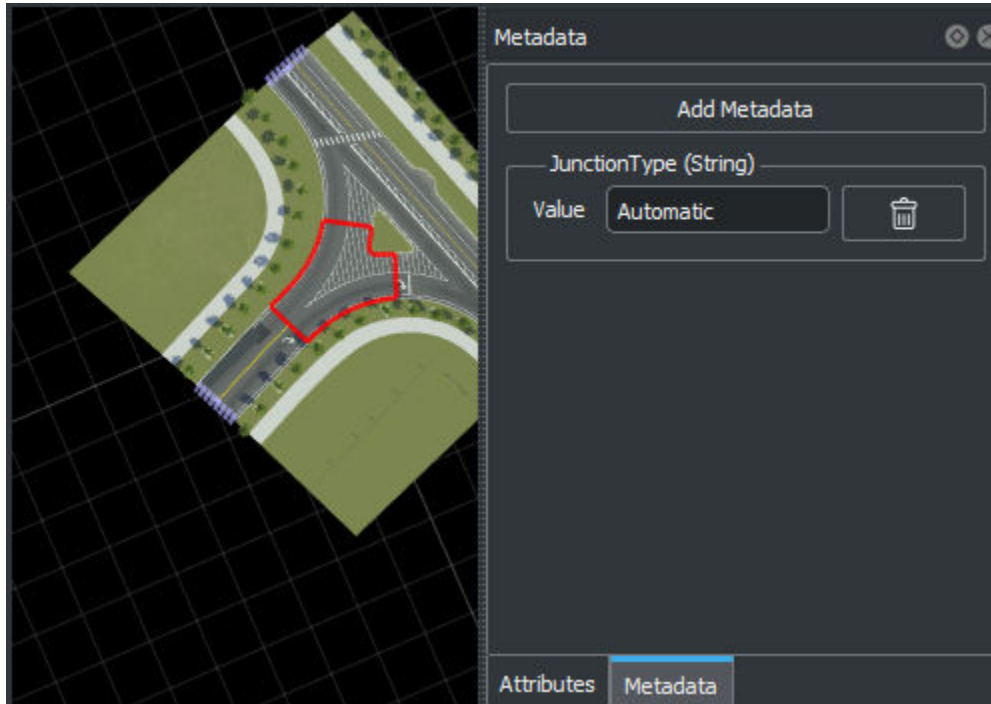
Export to ASAM OpenDRIVE

From the menu, select **File** > **Export** > **ASAM OpenDRIVE (.xodr)**. Export the scene to the OpenDRIVE 1.5 format.

Inspect ASAM OpenDRIVE Attributes

Inspect the attributes in the exported `.xodr` file.

You can verify that the `OpenDRIVE_name` metadata with the value `Highway` added to a road has set the `name` attribute for the road in the exported file to "`Highway`". The attribute value is defined within the `<road>` element.

```
<road name="Highway" length="1.9080212697060187e+1" id="4" junction="-1">
        <link>
            <successor elementType="junction" elementId="28"/>
        </link>
```

You can also verify that the `JunctionType` metadata with the value `Automatic` has been exported as user data. The attribute `JunctionType` is defined within a `<userData>` element.

```
<junction id="28" name="junction28">
        <connection id="0" incomingRoad="5" connectingRoad="29" contactPoint="start">
            <laneLink from="-1" to="-1"/>
        </connection>
        <connection id="1" incomingRoad="3" connectingRoad="30" contactPoint="start">
```

```
                    <laneLink from="-2" to="-1"/>
                    <laneLink from="-4" to="-3"/>
                    <laneLink from="-5" to="-4"/>
                    <laneLink from="-6" to="-5"/>
                </connection>
                <connection id="2" incomingRoad="4" connectingRoad="30" contactPoint="end">
                    <laneLink from="-2" to="-1"/>
                    <laneLink from="2" to="-4"/>
                    <laneLink from="3" to="-5"/>
                </connection>
                <connection id="3" incomingRoad="4" connectingRoad="33" contactPoint="start">
                    <laneLink from="-1" to="-1"/>
                    <laneLink from="-2" to="-2"/>
                    <laneLink from="-3" to="-3"/>
                </connection>
            <userData code="JunctionType" value="Automatic"/>
            <userData>
                <vectorJunction junctionId="{275e57c8-914f-466d-9c62-37ae8f570897}"/>
            </userData>
        </junction>
```

See Also
Custom Junction Tool|Road Plan Tool

More About
- "Create Simple RoadRunner Scene" on page 1-19
- "Importing ASAM OpenDRIVE Files" on page 3-2
- "Export to ASAM OpenDRIVE" on page 5-24
- "Add Metadata to RoadRunner Scene Elements" on page 5-42

External Websites
- ASAM OpenDRIVE

Export to ASAM OpenCRG

ASAM OpenCRG is an open standard that enables you to specify road surface data using the curved regular grid (CRG) format. Using RoadRunner, you can export road surface data when you export a scene to an ASAM OpenDRIVE file. RoadRunner exports the surface data of different road segments to different ASAM OpenCRG files, and links these files to an ASAM OpenDRIVE file.

The export file format conforms to the ASAM OpenCRG V1.2.0.

Export to ASAM OpenCRG

Follow these steps to export CRG data from a RoadRunner scene to an ASAM OpenCRG file.

1 From the menu, select **File** > **Export** > **ASAM OpenDRIVE (.xodr)**.

2 In the Export ASAM OpenDRIVE dialog box, select **Export OpenCRG and Synthetic OpenCRG Options**.

3 Select the desired **Road Data Format** from these options:

  - LRFI (default) — Long, real, formatted, interchangeable data format
  - LDFI — Long, double, formatted, interchangeable data format

4 Select your other desired options in the dialog box and click **Export**. For more details, see "Export to ASAM OpenDRIVE" on page 5-24.

RoadRunner saves all the ASAM OpenCRG files linked to a 3D scene to the folder specified in **File path** for the ASAM OpenDRIVE file.

See Also
Road CRG Tool

More About
  - Synthetic OpenCRG Assets
  - "Importing ASAM OpenCRG Files" on page 3-10
  - "Importing ASAM OpenDRIVE Files" on page 3-2
  - "Export to ASAM OpenDRIVE" on page 5-24

External Websites
  - ASAM OpenCRG

Segmentation

Segmentation Overview

RoadRunner has the ability to export scene geometry by category for easy generation of segmentation training data.

Segmentation categories are identified in the export scene by using suffixes applied to each material name. For example, a material named `Concrete` that is applied to a curb is named `Concrete_Curb` on export. The exported material inherits the built-in segmentation category suffix `Curb`, which uniquely distinguishes it from the material `Concrete_Sidewalk` applied to neighboring sidewalk geometry.

Toggle Segmentation Display

1　To enter segmentation display, select **View** > **Sensor** > **Segmented**.

2　To exit segmentation display, select **View** > **Sensor** > **Camera**.

Categories

RoadRunner uses a default set of segmentation categories when building geometry within a scene. These categories include: `Road`, `Sidewalk`, `Curb`, `Gutter`, `Marking`, `Ground`, `Building`, `Vehicle`, `Bike`, `Pedestrian`, `Sign`, `Signal`, `Foliage`, `Prop`, and `FireHydrantProp`. The default segmentation type is `Sign` for all the sign assets. However, a few of the sign assets have highly segmented categories which include: `StopSign`, `YieldSign`, `SpeedLimitSign`, `WeightLimitSign`, `RightArrowWarningSign`, `LeftArrowWarningSign`, `LeftAndRightArrowWarningSign`, `LeftChevronWarningSign`, `RightChevronWarningSign`, `LeftOneWaySign`, `RightOneWaySign`, `WheelChairWarningSign`, `SchoolBusOnlySign`, `RightTurnOnlyArrowSign`, `LeftTurnOnlyArrowSign`, `StraightOnlyArrowSign`, `RightTurnOnlySign`, `LeftTurnOnlySign`, `StraightOnlySign`, `NoLeftTurnSign`, `NoRightTurnSign`, `NoThruTrafficSign`, `NoUturnSymbolSign`, `NoRightTurnSymbolSign`, `NoLeftTurnSymbolSign`, `NoRightTurnOnRedSign`, `CrossWalkSign`, `CurveRightWarningSign`, `CurveLeftWarningSign`, `UpRightArrowWarningSign`, `UpLeftArrowWarningSign`, `DownRightArrowWarningSign`, `DownLeftArrowWarningSign`, `RailRoadCrossingSign`, `StreetSign`, `RoundaboutWarningSign`, `ExitSign`, `BikeLaneSign`, and `CrossWalkSignal`.

Additionally, you can extend the categories and assign them to the following project asset types to create custom categories: `Props`, `Signals`, `Lane Markings`, `Polygon Markings`, and `Stencils`.

Add a Custom Category

1　In a text editor, open the `SegmentationCategories.xml` file located in the `Project` folder of the project. If the file does not exist, create one.

2　Add a new `Category` entry. Include a name and color attribute, which are used during export and segmentation display, respectively.

3　Existing categories can also be modified or removed. Changing the name of an existing category is equivalent to removing the old category and adding a new one. Existing assets referencing this old name will default.

4　Save the file and restart RoadRunner. New categories are available only after the project is reloaded.

This code shows an example `SegmentationCategories.xml` file.

```xml
<?xml version="1.0"?>
<CustomSegmentationCategories>
    <Category name="Bush" color="#7BA269"/>
    <Category name="Tree" color="#0F5F32"/>
    <Category name="Crosswalk" color="#963"/>
    <Category name="DashedMarking" color="#369"/>
    <Category name="SolidMarking" color="#48a"/>
    <Category name="DoubleMarking" color="#69b"/>
</CustomSegmentationCategories>
```

Export Scene Geometry Grouped for Segmentation

On export, RoadRunner supports grouping materials by segmentation category or separating them into individual meshes. To toggle between these options, follow these steps.

1 Select **File > Export** and select a triangulated format, such as Filmbox or OpenFlight.

2 In the export dialog box, fill out the file location and any tiling options.

3 Optionally use the **Split by Segmentation** toggle in the **Options** group to control whether each mesh is split by category or remains grouped.

4 Click **Export**.

Assign a Category to an Asset

1 Select the asset in the **Library Browser**.

2 In the **Attributes** pane, under **Segmentation**, select the appropriate category.

3 Select **File > Save Project** in the menu bar.

Downloading Plugins

RoadRunner provides plugins for exporting scenes to Unity, Unreal, and CARLA.

The latest plugins can be downloaded here.

The plugins are delivered as a zip file containing subfolders for each plugin type. Refer to the following sections for details on installing and using a specific plugin.

Unity

See "Export to Unity" on page 5-59 for instructions on installing and using the plugin.

Unreal and CARLA

The basic RoadRunner importer and the CARLA integration plugin are included in the same plugin folder. If you are using Unreal and not CARLA, copy only the `RoadRunnerImporter`, `RoadRunnerRuntime`, `RoadRunnerMaterials` folders. If you are using CARLA, copy the `RoadRunnerImporter`, `RoadRunnerRuntime`, `RoadRunnerMaterials`, and `RoadRunnerCarlaIntegration` folders.

See "Export to Unreal Using Filmbox (.fbx) File" on page 5-85 or "Export to CARLA" on page 5-94 for instructions on installing and using the plugin.

RoadRunner Metadata Export

RoadRunner includes an extra metadata file for certain export options.

Metadata Overview

When exporting to Unity on page 5-59, Unreal on page 5-85, or CARLA on page 5-94, an additional ".rrdata.xml" file is generated during export. This file is used in combination with the RoadRunner import plugins to help cover the information not available in the FBX file. The metadata file holds information about the materials included in the scene and holds traffic signal information. For examples on how to parse this information, refer to the Unity or Unreal plugins included with the RoadRunner installation under the "Tools" folder.

File Details

The metadata file continues to update as needed. The metadata version is stored under the top-level element (for example, <RoadRunnerMetadata Version="3">).

The data is organized into three main sections: SignalConfigurations, Signalization, and MaterialList.

SignalConfigurations

This section holds information about how the signal bulbs change for each configuration of a traffic light (for example, which bulbs are on and off during a green light or red light).

Example:

```
<Signal>
    <ID>{9b15662e-0dae-40d5-ab82-55e0077bcbc2}</ID> // GUID of signal asset
    <Type>Straight Right</Type> // Supported turn types
    <Configuration> // Configuration entry
        <Name>Red</Name> // State name
        <LightState> // Light bulb mesh state
            <Name>light_red_on</Name> // Mesh node name in signal's FBX
            <State>true</State> // "true" if mesh should be visible
        </LightState>
        <LightState>
            <Name>light_red_off</Name> // Corresponding mesh node when light is off
            <State>false</State>
        </LightState>
        ...
    </Configuration>
    ...
</Signal>
```

Signalization

This section holds information about each traffic junction in the scene and how each signal changes over time.

Example:

```
<Junction>
    <ID>{5c348c08-d2d7-423e-b560-04eb52ddcd10}</ID> // GUID of junction
    <SignalPhase> // Each signal phase holds a list of intervals
        <Interval> // Each interval represents state of junction (e.g., "northbound road currently has yellow light")
            <Time>20</Time> // Duration of the interval
            <Signal>
                <ID>{d33d9030-c427-44ff-860b-486f3caf45b2}</ID> // GUID of signal prop. Can be referenced to node in expo
                <SignalAsset>{9b15662e-0dae-40d5-ab82-55e0077bcbc2}</SignalAsset> // GUID of signal asset. Can be referer
```

```
                        <ConfigurationIndex>2</ConfigurationIndex> // Index into the list of configurations held in the signal as
                    </Signal>
                    <Signal>
                        <ID>{00dd1cc3-9b68-44dd-bb20-a3e49452606f}</ID> // All signals attached to the junction are listed
                        <SignalAsset>{9b15662e-0dae-40d5-ab82-55e0077bcbc2}</SignalAsset>
                        <ConfigurationIndex>0</ConfigurationIndex>
                    </Signal>
                    ...
            </Interval>
            ...
        </SignalPhase>
        ...
</Junction>
```

MaterialList

This section contains a list of all the materials used in the scene, along with all the parameters so that
they can be reconstructed in the target software.

Example:

```
<Material>
    <Name>Asphalt1</Name> // Name of the material, matches the one stored in the FBX
    <DiffuseMap>Asphalt1_Diff.png</DiffuseMap>
    <NormalMap>Asphalt1_Norm.png</NormalMap>
    <SpecularMap>Asphalt1_Spec.png</SpecularMap>
    <AmbientColor>1.000000,1.000000,1.000000</AmbientColor> // Ambient color matches diffuse
    <DiffuseColor>1.000000,1.000000,1.000000</DiffuseColor>
    <SpecularColor>0.058824,0.058824,0.058824</SpecularColor>
    <Roughness>0.150000</Roughness>
    <SpecularFactor>1.000000</SpecularFactor>
    <TransparencyFactor>0.000000</TransparencyFactor> // Inverse of diffuse color alpha
    <Emission>0.000000</Emission>
    <TextureScaleU>0.35</TextureScaleU>
    <TextureScaleV>0.35</TextureScaleV>
    <TwoSided>false</TwoSided>
    <DrawQueue>0</DrawQueue> // Render order for overlapping transparent markings
    <ShadowCaster>true</ShadowCaster>
    <IsDecal>false</IsDecal> // Set to "true" for transparent markings
    <SegmentationType>Road</SegmentationType>
</Material>
```

Export to Apollo

Apollo Overview

RoadRunner can export road scenes to Baidu Apollo formats. You can export to Apollo 3.0 and 5.0 XML formats and Apollo 5.0 binary format.

Before you export to any Apollo format, ensure that your scene's world origin is set in the **World Settings Tool**. Once you are ready to export, navigate to **File > Export > Apollo (.bin, .xml)** to open the export options window. Make sure you choose the appropriate Apollo version before completing your export, and ensure that you specify the file extension (`.xml` or `.bin`) you intend to export to. Exports to the binary format will include a human readable `.txt` representation of the protobuf data serialized in the `.bin` counterpart.

About the Different Apollo Maps

The Apollo Dreamview front-end tool can visualize and simulate routing on RoadRunner Apollo exports. A complete map is composed of the following files for proper simulation:

- `base_map.bin` — A protobuf representation of HD Map information. The representation might be accompanied with a human-readable `.txt` version.
- `sim_map.bin` — A downscaled version of `base_map.bin` used for faster visualization at runtime. The file might be accompanied with a `.txt` version.
- `routing_map.bin` — Topological map information used for generating routes.
- `default_end_way_point.txt` — A start point for routing.

Given either a binary or XML export from RoadRunner, these files can be generated using various tools provided by the Apollo codebase.

Generating Necessary Map Files

Note If you have not yet set up your Apollo Docker® environment, follow the Apollo 3.0 guide or Apollo 5.0 guide to do so.

Binary maps can be generated from XML by using the following command in the Apollo Docker environment:

```
bazel-bin/modules/maps/tools/proto_map_generator --map_dir=INPUT_DIR
--output_dir=OUTPUT_DIR
```

`INPUT_DIR` is the name of the directory containing the XML file, and `OUTPUT_DIR` is the desired output directory. Within the input directory, ensure the XML file is named `base_map.xml` before running it. This will generate a binary file named `base_map.bin` and a text file version named `base_map.txt` in the output directory specified.

With this `.bin` file or a `.bin` file exported directly from RoadRunner, a `sim_map` can be generated with the following command:

```
bazel-bin/modules/maps/tools/sim_map_generator --map_dir=INPUT_DIR
--output_dir=OUTPUT_DIR
```

Again, ensure that the `.bin` file is named `base_map.bin` before running it.

A routing map can be generated with the following command:

```
scripts/generate_routing_topo_graph.sh --map_dir=INPUT_DIR
```

More information about the different Apollo map types can be found here.

Visualizing Maps in Apollo Dreamview

Once you have all of the components for an Apollo map, you can visualize and simulate it in the Dreamview front-end.

Create a folder for your map in `apollo/modules/maps/data`, and add all the map files to that folder. Rename the folder to what you would like to appear in the Dreamview map selection drop-down. Restart Dreamview to refresh the maps in your data folder.

Once Dreamview starts, select your newly added map and a test vehicle in the top-right corner. Ensure that the standard mode is selected.

Go to the **Tasks** tab on the left, and enable **Sim Control** to render the map.

Routing Simulations in Apollo Dreamview

To run a road simulation in Dreamview, ensure that **Routing** is enabled in the **Modules** window. In the routing window, define a route on the map by using at least two waypoints. Click **Send Route Request** to run the simulation.

Visualizing Maps in SVL Simulator

The SVL Simulator has the ability to import Apollo 5.0 binary files for editing and visualization.

Note If you have not yet set up the SVL Simulator with Unity, see the SVL Simulator documentation.

To import an Apollo map into the SVL Simulator, open the **HD Map Import** window under **Simulator > Import HD Map**. Under **Import Format**, select **Apollo 5 HD Map**, and optionally modify the **Distance** and **Delta Threshold** values. Click "**...**" to open the file browser and select the binary file export. Click **import** to add the map to the scene.

More information about importing maps into the SVL Simulator can be found in the SVL Simulator documentation.

Apollo User Asset Configuration

The Apollo exporter uses a configuration XML file to map RoadRunner props, signals, signs, and markings to the appropriate `<object>` or `<signal>` `"id"` and `"subtype"`. This process works the same way as the ASAM OpenDRIVE asset configuration.

Exporting a Custom Prop or Signal

1 Copy the `ApolloAssetData.xml` file located in the RoadRunner install location under `AssetsInstall/ResourceAssets` to the `Project` folder in your project (next to the `Project.rrproj` file).

2 Open the new `ApolloAssetData.xml` file in a text editor.

3 Add entries for new objects, markings, or signals.

4 Save your file and export an Apollo file.

You do not need to restart RoadRunner after creating or modifying the `ApolloAssetData.xml` file.

The format of an `ApolloAssetData.xml` file is the same as the format of an `OpenDRIVEAssetData.xml` file. For a detailed definition of this format, see "Export to ASAM OpenDRIVE" on page 5-24. The template file in the `AssetsInstall/ResourceAssets` folder also contains examples for a traffic light, stop sign, and yield sign.

Exporting a Traffic Signal with Multiple Variances

Here is the definition of the format to properly set a traffic signal's asset data for different variances and subsignals.

```
<Signals>
    <Signal>
        <Type>trafficLight</Type>
        <SubType>mix3Vertical</SubType>
        <SubSignals>
            <Variant>1</Variant>
            <SubSignal>
                <LightName>light_red</LightName>
                <Type>circle</Type>
            </SubSignal>
            <SubSignal>
                <LightName>light_yellow</LightName>
                <Type>circle</Type>
            </SubSignal>
            <SubSignal>
                <LightName>light_green</LightName>
                <Type>circle</Type>
            </SubSignal>
        </SubSignals>
        <SubSignals>
            <Variant>2</Variant>
            <SubSignal>
                <LightName>light_red</LightName>
                <Type>arrowLeft</Type>
            </SubSignal>
            <SubSignal>
                <LightName>light_yellow</LightName>
                <Type>arrowLeft</Type>
            </SubSignal>
            <SubSignal>
                <LightName>light_green</LightName>
                <Type>arrowLeft</Type>
            </SubSignal>
        </SubSignals>
        <SubSignals>
```

```
            <Variant>3</Variant>
            <SubSignal>
                <LightName>light_red</LightName>
                <Type>arrowRight</Type>
            </SubSignal>
            <SubSignal>
                <LightName>light_yellow</LightName>
                <Type>arrowRight</Type>
            </SubSignal>
            <SubSignal>
                <LightName>light_green</LightName>
                <Type>arrowRight</Type>
            </SubSignal>
        </SubSignals>
        <FilePath>Props/Signals/Signal_3Light_Post01.fbx</FilePath>
    </Signal>
</Signals>
```

Unsupported Features

The following data records are currently unsupported for export.

- Route View Record (`<routeView>` … `</routeView>` and its children)
- Lane Overlap Group (`<laneOverlapGroup>` … `</laneOverlapGroup>` and its children)

See Also

Export to Metamoto

RoadRunner can export road scenes for use in Metamoto simulations. To export to Metamoto, follow these steps:

1 Select **File > Export > Metamoto**.

2 Specify a file name. All other export options are set for exporting to Metamoto.

When you export, the software generates a zip file containing the files needed to use your scene in Metamoto. These zipped files have the same name you specified when exporting. For example, *MyScene*`.zip` would contain:

- *MyScene*`.fbx` (and any other necessary texture files)
- *MyScene*`.xodr`
- *MyScene*`.geojson`
- *MyScene*`.rrdata.xml`

For any limitations, refer to the documentation about exporting to FBX on page 5-3, OpenDRIVE® on page 5-24, and GeoJSON on page 5-9.

See Also

External Websites

- https://www.foretellix.com/

Export to Unity

Unity Overview

RoadRunner can export scenes to Unity format. The Unity export option exports a Filmbox (.fbx) file containing the 3D objects in a scene along with an additional XML file to hold extra data for materials and traffic signals in the scene.

On the Unity side, a set of scripts are included in the RoadRunnerUnityTool asset package to help import the FBX file using the information stored in the XML file. The script handles the following details:

- Setting up materials

  - Material data is read in from the XML file and mapped into the included custom shaders.

- Adding colliders to roads and terrain

  - Colliders are added to all imported meshes.

- Setting up the components of traffic signals

  - Signal data is read in from the XML file to create a new game object in the prefab, with the light bulb references to game objects set up during import.
  - The traffic signals will cycle through their phases during play mode.
  - The UUIDs prefixed in the game object for prop instances are needed only at import time to set up references to game objects in the traffic signal script so that they can be renamed freely.

- Unity software requirements: Unity Version 2017.3+

Installing the Import Tool

Follow the instructions in this section to install the Import Tool into your Unity project.

1 See the page "Downloading Plugins" on page 5-51 for instructions for downloading the latest version of the plugin.

2 Extract the RoadRunner Plugins zip file and locate the "RoadRunnerUnityTool.unitypackage" file in the "Unity" folder.

> **Note** For Unity versions 2017.1 through 2017.3, use `RoadRunnerUnityTool_2017.unitypackage` file in the `Unity` folder.

3 Open your project in Unity.

4 Open the RoadRunnerUnityTool asset package file to import it. Alternatively, drag the package file into the Unity Project window, or select

  Assets > **Import Package** > **Custom Package** and then select the package file.

5 Click **Import** in the Import Unity Package dialog box.
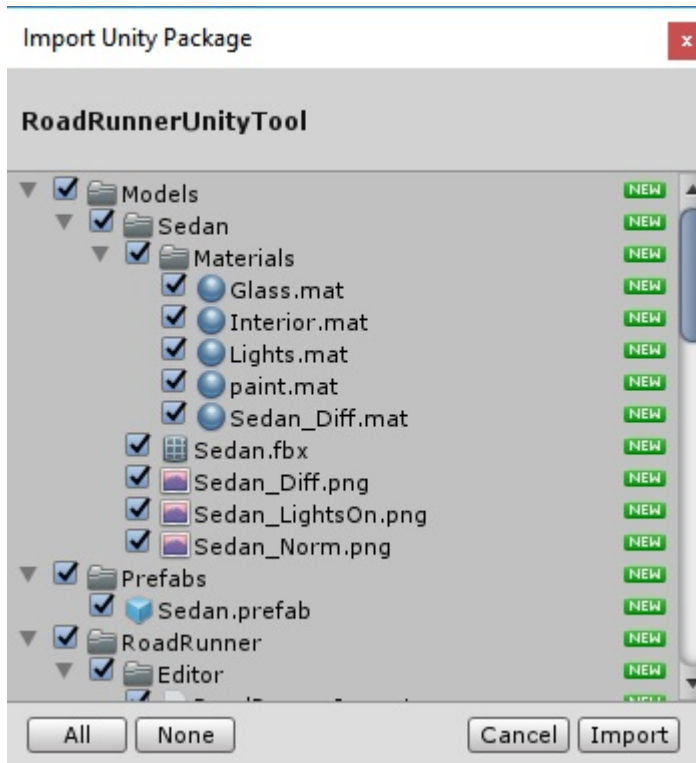
Selecting Package Files to Import

The package includes some extra files to add a drivable vehicle to your scene. If you do not need these extra files, then you can deselect them when you import the package.

The essential scripts inside the "RoadRunner" folder are needed to set up the materials and traffic signals in the scene.
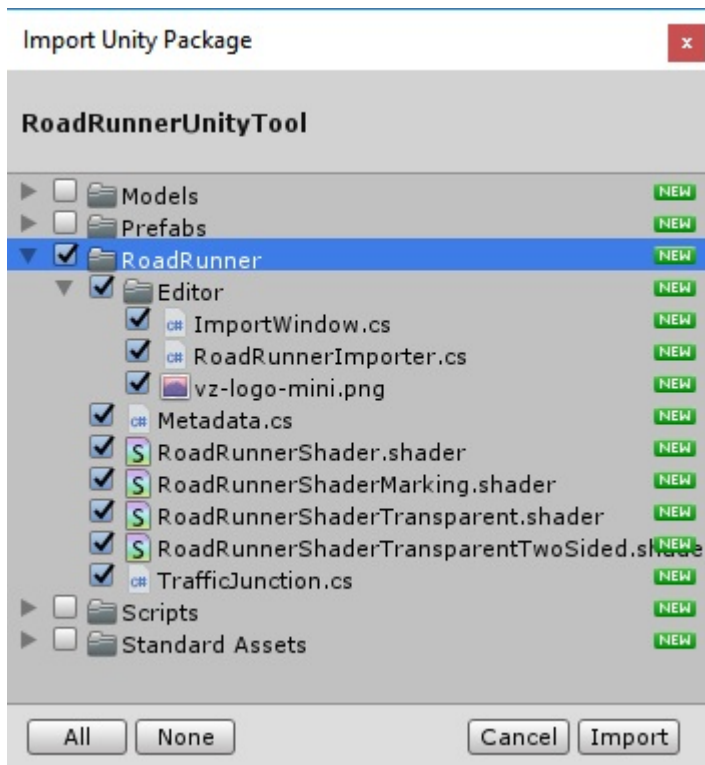
Package Contents

- Models: Mesh, materials, and textures for the drivable car prefab.
- Prefabs: The prefab for the car with scripts set up.
- RoadRunner:

  - ImportWindow.cs: Editor window to display messages for the RoadRunner importer.
  - RoadRunnerImporter.cs: Editor script for importing the FBX file with the data from the XML file.
  - Metadata.cs: Contains classes to hold the imported metadata XML file.
  - TrafficJunction.cs: Component for controlling signals from data in the XML file at import time.
  - Various shaders to match RoadRunner material settings.
- Scripts: For the Sedan prefab.
- Standard Assets: For the Sedan prefab.

Exporting from RoadRunner to Unity

Follow these steps to export a scene from RoadRunner to Unity:

1   Open your scene in RoadRunner.
2   Export the scene to Unity format using **File > Export > Unity (.fbx + .xml)** from the menu bar.
3   In the Export Unity dialog box, set your desired options, and then click **Export**.

4 Browse to open the file dialog box to set the exported file's name and path. The textures and the XML file are exported to the same folder. (Tip: Create a new folder when choosing a file location, so you can import the entire folder into Unity.)

- The mesh can be split by segmentation type. Meshes have "<segmentation type>Node" appended to their names.
- If the **Export To Tiles** option is selected, meshes are split per tile. Props are grouped by the tile they are in.
  - By default, only one file is exported. Tiles are stored in separate nodes.
  - If **Export Individual Tiles** is enabled, each tile will be stored in its own FBX file.
- When exporting with **Embed Textures** selected, you need to manually extract the textures inside Unity.
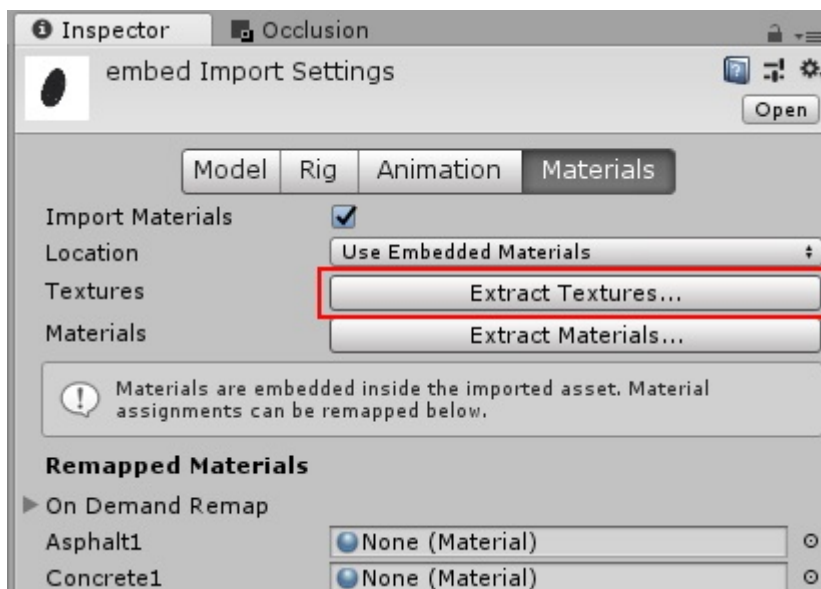
Importing into Unity

To import a scene into Unity that you previously exported from RoadRunner, drag all the exported files (or the entire folder) into the Unity project window. Alternatively, use **Assets > Import New Asset** in Unity and select all the exported files.

The output window that opens contains log messages from the import plugin.

```
Output Log

> Found metadata: Assets/_Scene/scene.rrdata.xml
> Importing material data
> Importing traffic signals
> Finished importing scene


                                                    Close
```
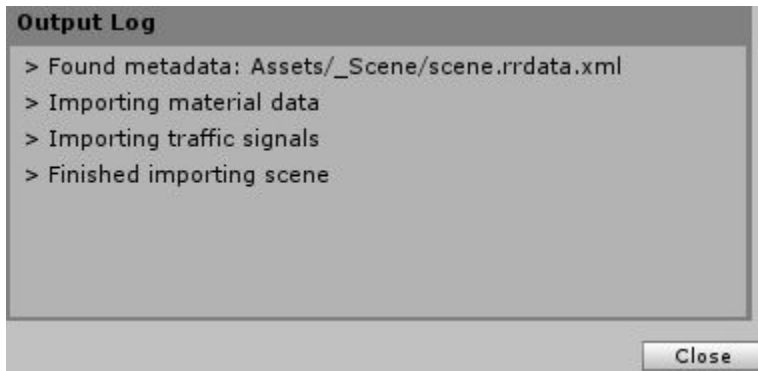
(Optional) Test Drive in Unity

You can place and drive a car model around an imported scene by following these steps:

1 Drag the Sedan prefab from the Prefabs folder into the scene. (Note: In some versions of Unity, you might need to manually tag the Main Camera as "MainCamera" for some scripts to work.)
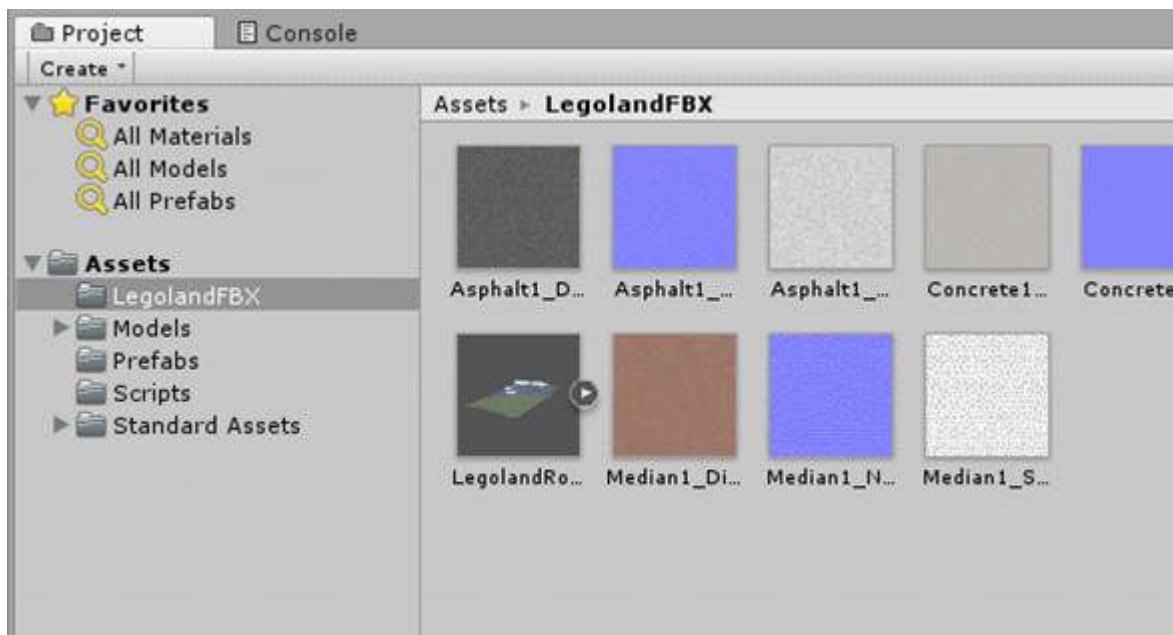
2 Click **Run**.

About Importing Traffic Signals into Unity

If traffic signals were set up in RoadRunner, then they are imported into Unity as junction controllers. These controllers are automatically created during import and attached to the prefab.

Prop instances for traffic signals are prefixed by their UUID so that the traffic signal controller has a way to identify which signals it controls. The TrafficJunction script handles the logic for switching between signal states.

FBX details

The FBX file is identical to the one exported from the Filmbox export option. The only difference is the extra rrdata.xml metadata file.

Setting Up the Sample Vehicle

The RoadRunnerUnityTool unitypackage also includes the RoadRunner Sedan 3D model. This section covers how to set it up with Unity Standard Assets.

Note The following section was tested on Unity 2019.1. Older versions might require different steps to modify prefabs.

Adding the Standard Assets

1   Download the "Standard Assets (for Unity 2017.3)" package from the Unity Asset Store.

2   Select the following folders to import:

- "Standard Assets/Cameras"
- "Standard Assets/CrossPlatformInput/Scripts"
- "Standard Assets/Editor/CrossPlatformInput"
- "Standard Assets/Vehicles/Car"
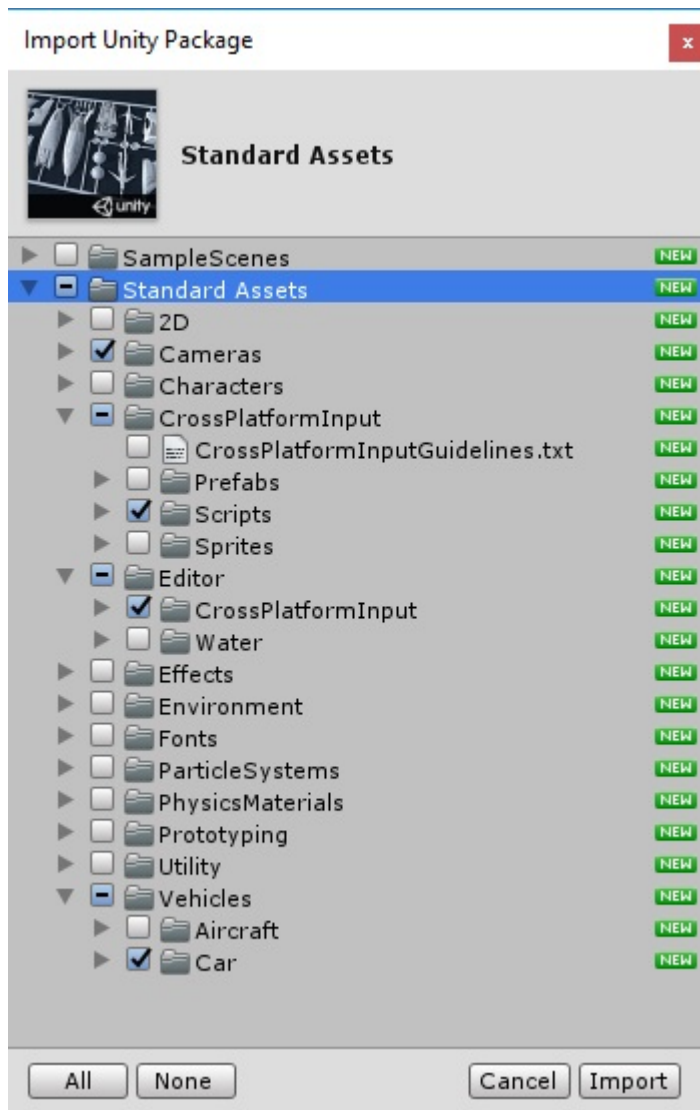
3   Import the package by clicking **Import**.

Setting Up the Sedan Prefab

1   Select the Sedan prefab (located in "Assets/Prefabs") and click **Open Prefab** in the Inspector window.
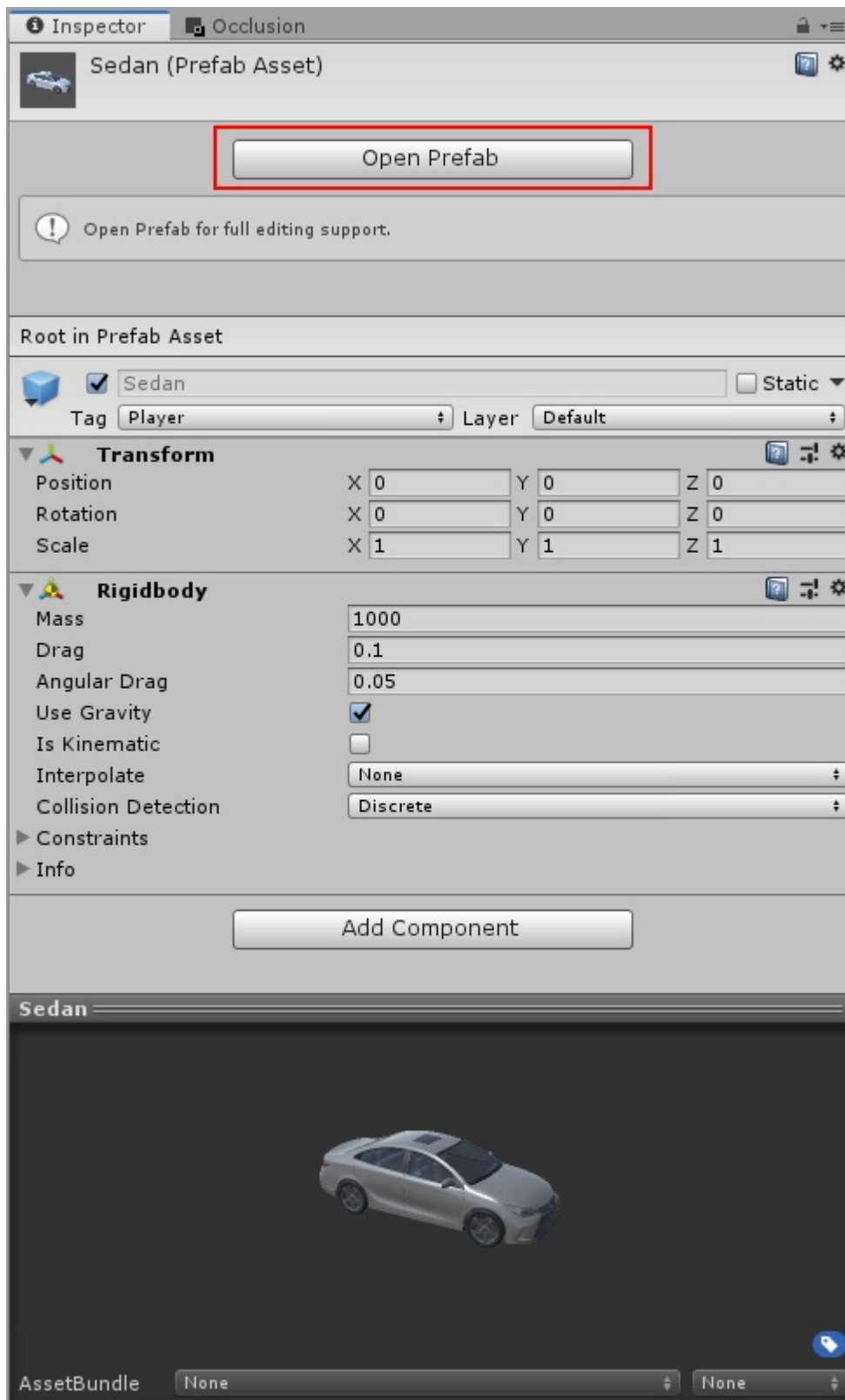
2 For each Wheel Hub Game Object under "Sedan/WheelsHubs" (for example, "WheelHubFrontRight"), complete these steps:
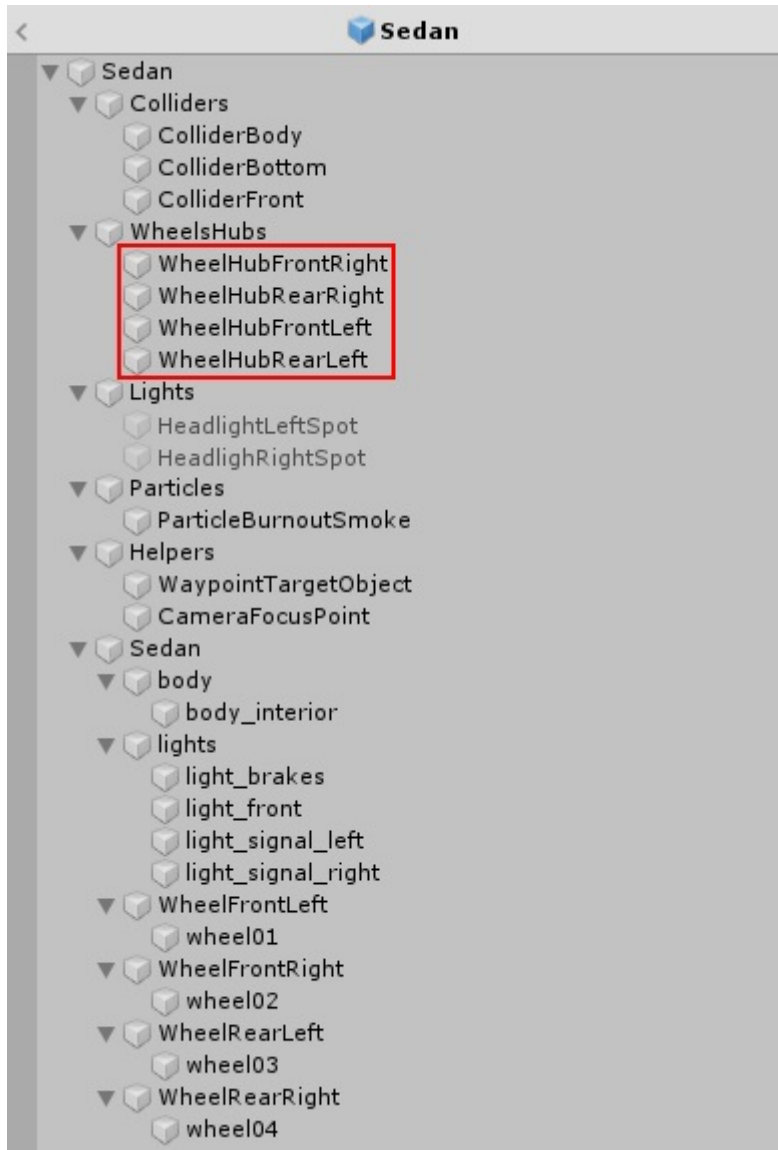
a Add the "Wheel Effects" component.

  - Set the "Skid Trail Prefab" to the "SkidTrail" Prefab (located in "Assets/Standard Assets/Vehicles/Car/Prefabs").

b Set the Audio Source's "AudioClip" to the "Skid" sound effect if it is missing.
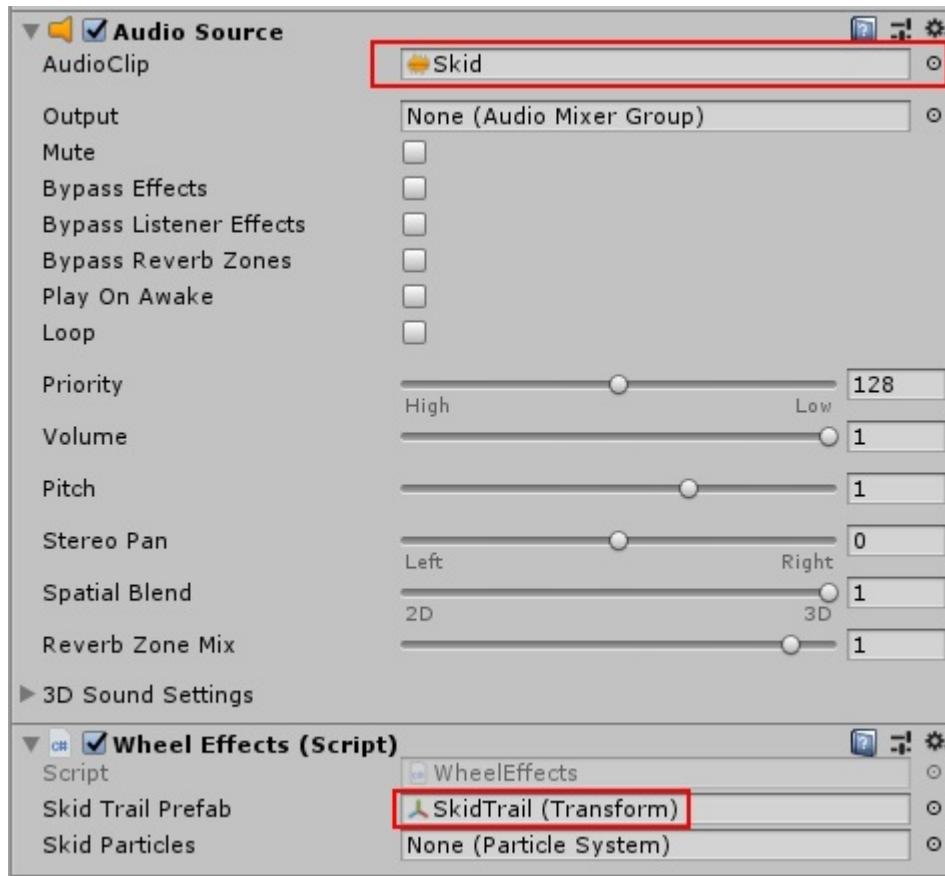
3   Select the top level "Sedan" game object in the hierarchy window.

a Add the "Car Controller" component with the following settings.
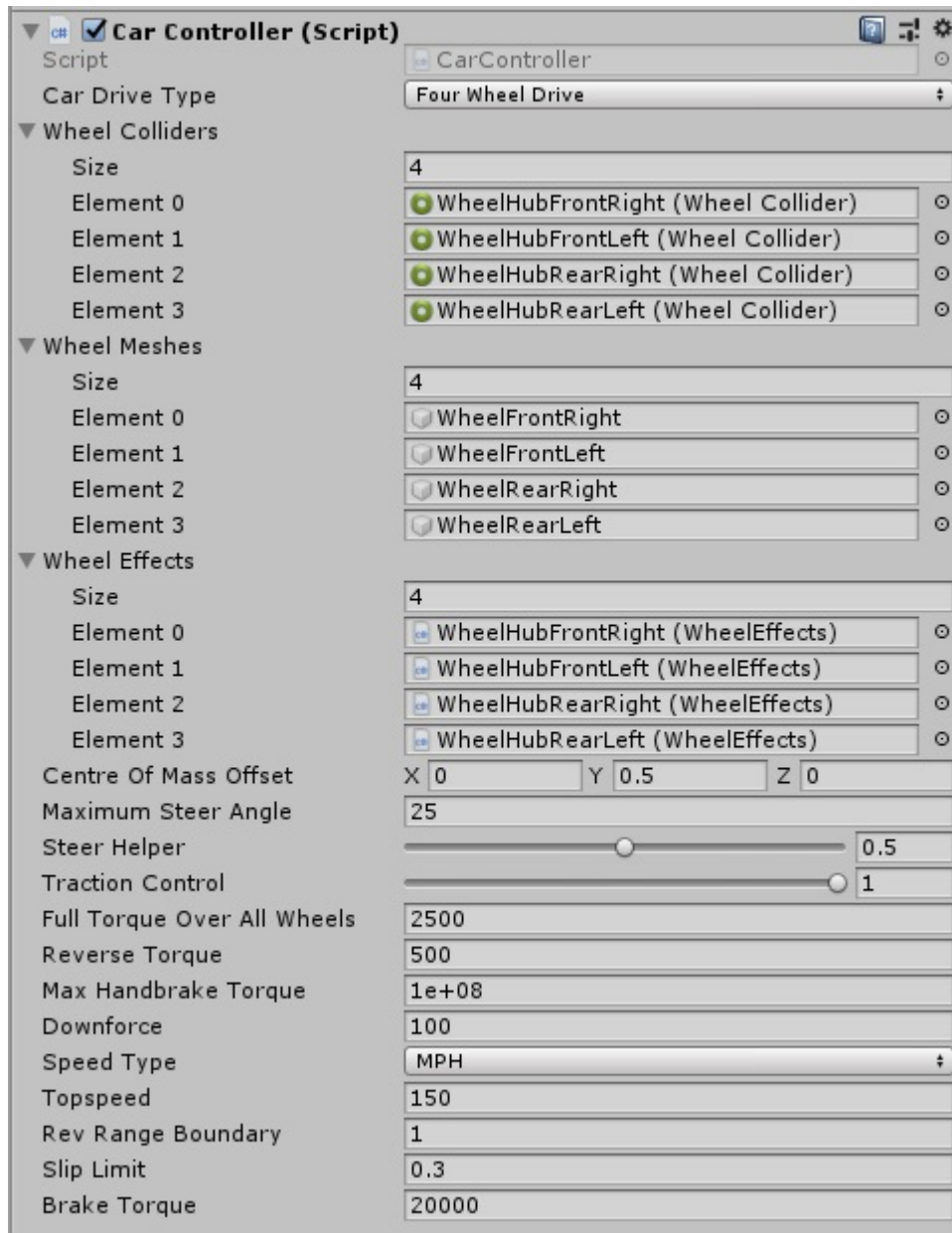
To avoid errors, verify that the order of the wheels is correct.

b   Add the "Car Audio" component with the following settings.
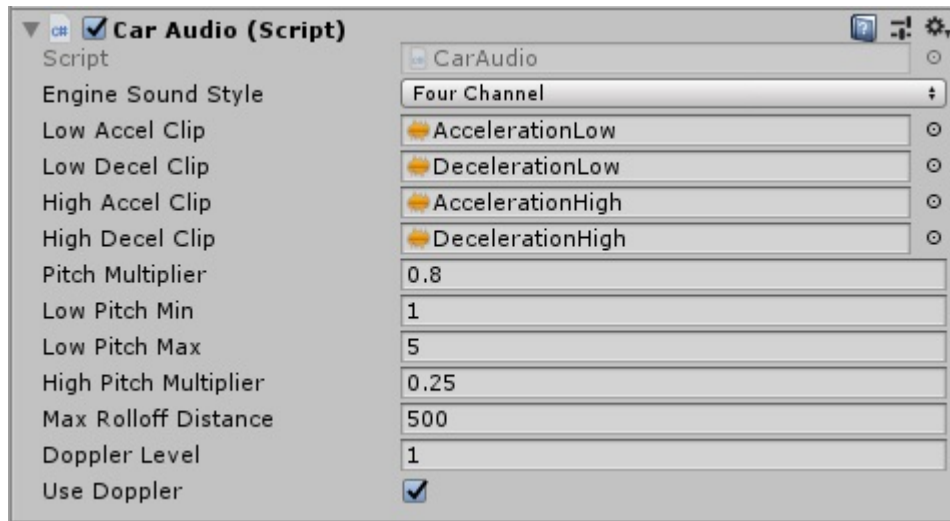
c Add the "Car User Control" component.
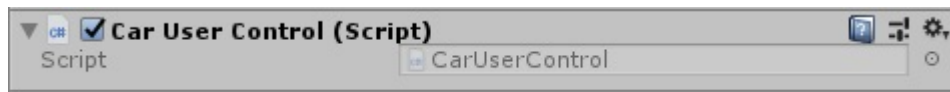
4 Select the "light_brakes" game object.

a Add the "Brake Light" component and set the "Car" property to the top level "Sedan" game object.

5 Select the "ParticleBurnoutSmoke" game object.
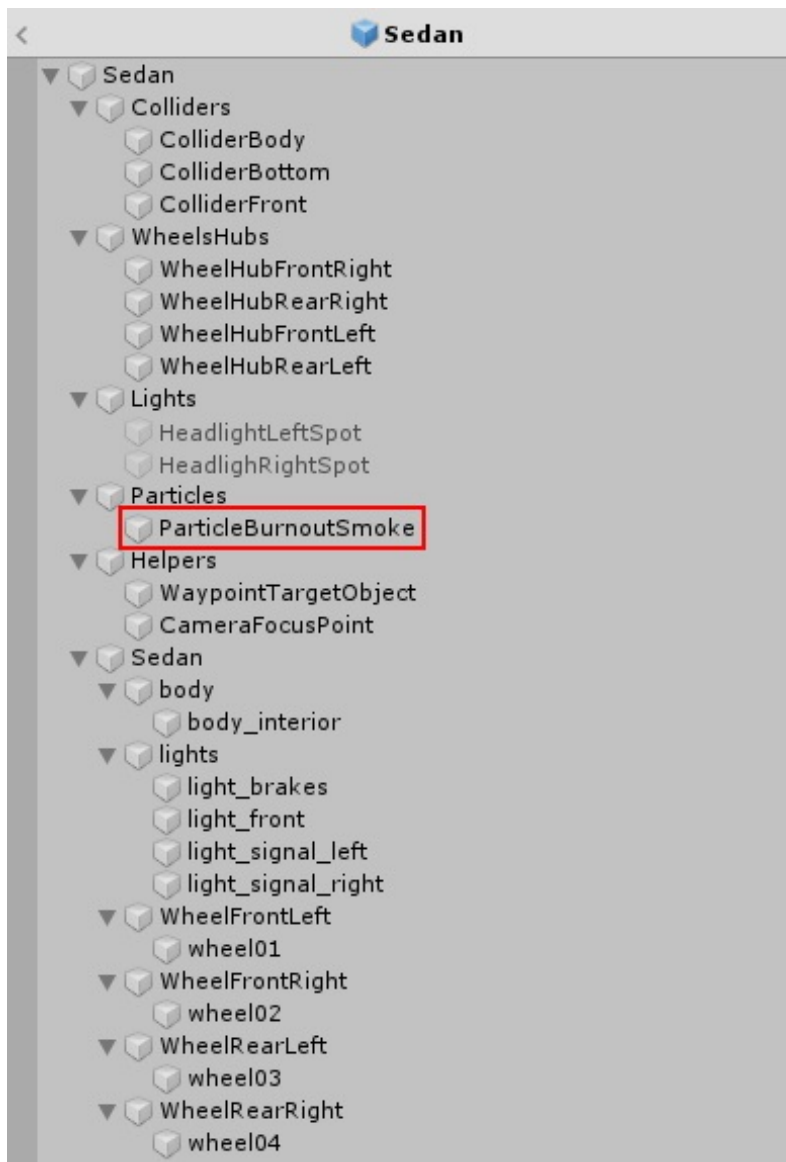
a Enable the game object.

b Set the Particle System component's Renderer Material property to "ParticleSmokeWhite".
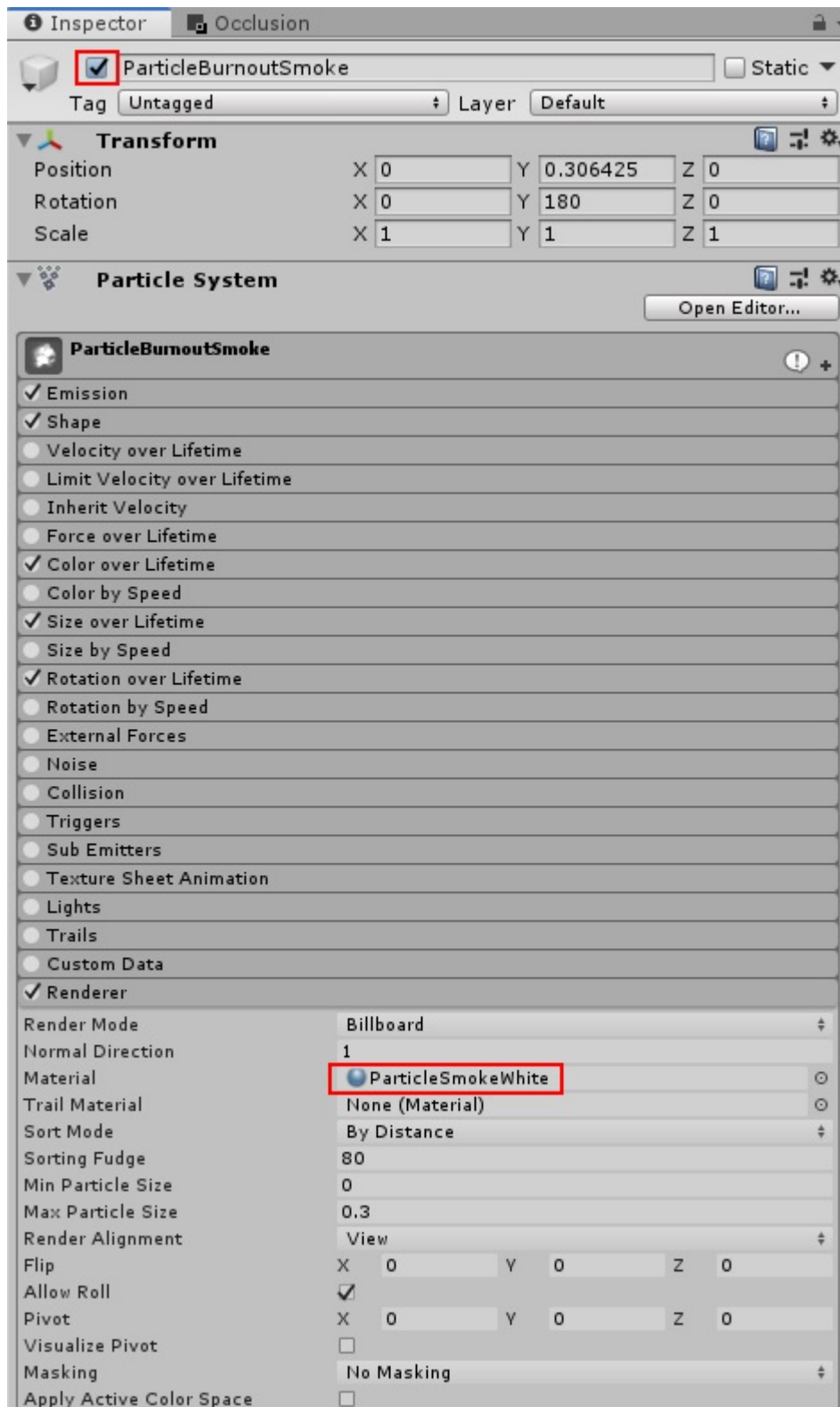
6　Exit Prefab edit mode.

Setting Up the Camera

1   Delete any existing cameras in the scene.

2   Add the Sedan prefab to the scene.

3   Add the "MultipurposeCameraRig" prefab (located in "Standard Assets/Cameras/Prefabs") to the
scene with the following settings.
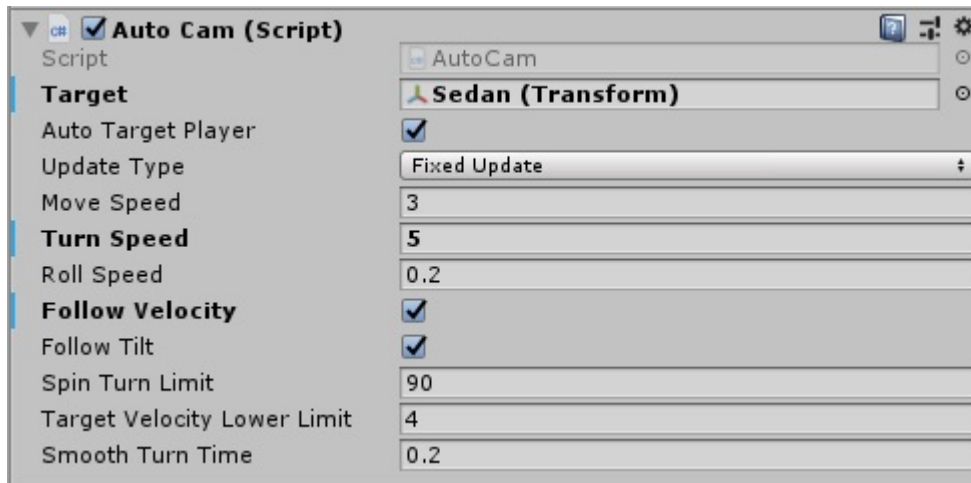
Running the Scene

Add in an imported RoadRunner scene and click play to drive around in it.

Export to Unreal Using Datasmith (.udatasmith) File

Unreal Overview

RoadRunner can export scenes to Unreal. The Unreal export option exports a Datasmith (.udatasmith) file.

On the Unreal side, a plugin is provided to help import the Datasmith file. The plugin handles these processes:

- Setting up materials

  - Reads in material data and maps the data to a new instance of one of the base materials included with the plugin.
  - For transparent material, selects between the translucent and masked blend modes based on the transparency of the diffuse color.
- Adjusting the colliders in the imported static meshes

  - During import, newly created static mesh assets have their `Collision Complexity` property set to `Use Complex Collision As Simple`.
- Unreal software requirements: Unreal Version 4.24+.
- Only `C++` based Unreal projects are supported.

Installing the Plugin

Follow the instructions in this section to install the Unreal plugin.

1 See "Downloading Plugins" on page 5-51 for instructions for downloading the latest version of the plugin.

2 Extract the RoadRunner Plugins ZIP file and locate the `RoadRunnerDatasmith`, `RoadRunnerImporter`, `RoadRunnerMaterials`, and `RoadRunnerRuntime` folders under `Unreal/Plugins`.

Note The Unreal plugin folder also contains a `RoadRunnerCarla` integration plugin. Do not copy this folder if you are not using CARLA.

---

3 Copy the folders into the `Plugins` folder under the project directory. If a `Plugins` folder does not exist, create one.

Name	Type	Size
Config	File folder	
Content	File folder	
Intermediate	File folder	
Plugins	File folder	
Saved	File folder	
packageTest.uproject	Unreal Engine Proj...	1 KB

4 Rebuild the plugin.

    a Generate the project files. This feature only works on Windows platform.

        • Windows — Right-click the `.uproject` file and select **Generate Visual Studio project files**.

    b Open the project and build the plugins by clicking **Yes**.
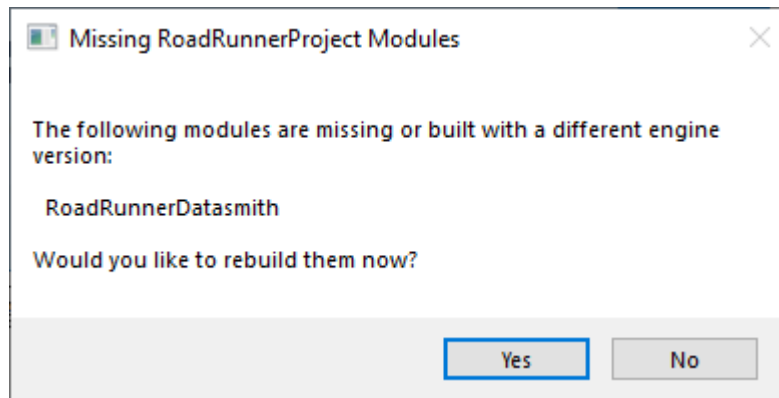
5 The plugin appears in Unreal under **Edit > Plugins**. If it does not appear in that menu, check that the **Enabled** parameter is selected.

6 In the Unreal editor, add Datasmith Importer plugin. Unreal requires a restart to add the plugin. The Datasmith plugin appears as a new icon on the Unreal toolstrip.

Plugin Contents

- `RoadRunnerDatasmith` module:

  - Dataprep asset that handles metadata post-processing
  - Imports signal data and timing

- `RoadRunnerRuntime` module:

  - Contains component to control traffic signal visuals

- `RoadRunnerMaterials` plugin:

  - Base materials to create instances from

Exporting from RoadRunner to Unreal

Follow these steps to export a scene from RoadRunner to Unreal:

1 Open your scene in RoadRunner.

2 Export the scene using the Unreal option. Select **File > Export > Unreal Datasmith (.udatasmith)** from the menu bar.

3 In the Export Unreal dialog box, set the mesh merging and tiling options, and then click **Export**.

4 Select **Browse** to open the File dialog box and set the exported file's name and path. The Datasmith file exports to the specified folder.

- You can split the mesh by segmentation type. Meshes have `<segmentation type>Node` appended to their names.

- If the **Export To Tiles** option is selected, meshes are split per tile. Props are grouped by the tile they are in.

  - By default, only one file is exported. Tiles are stored in separate nodes.
  - If **Export Individual Tiles** is enabled, each tile are stored in its own Datasmith file.

Importing into Unreal

To import the scene into Unreal use the Datasmith icon on the Unreal Editor toolstrip. A window, Datasmith Import, opens and enables you to select a file with the `.udatasmith` extension. You can select the destination folder in the current project.

When the Unreal Import Options Dialog Box Opens
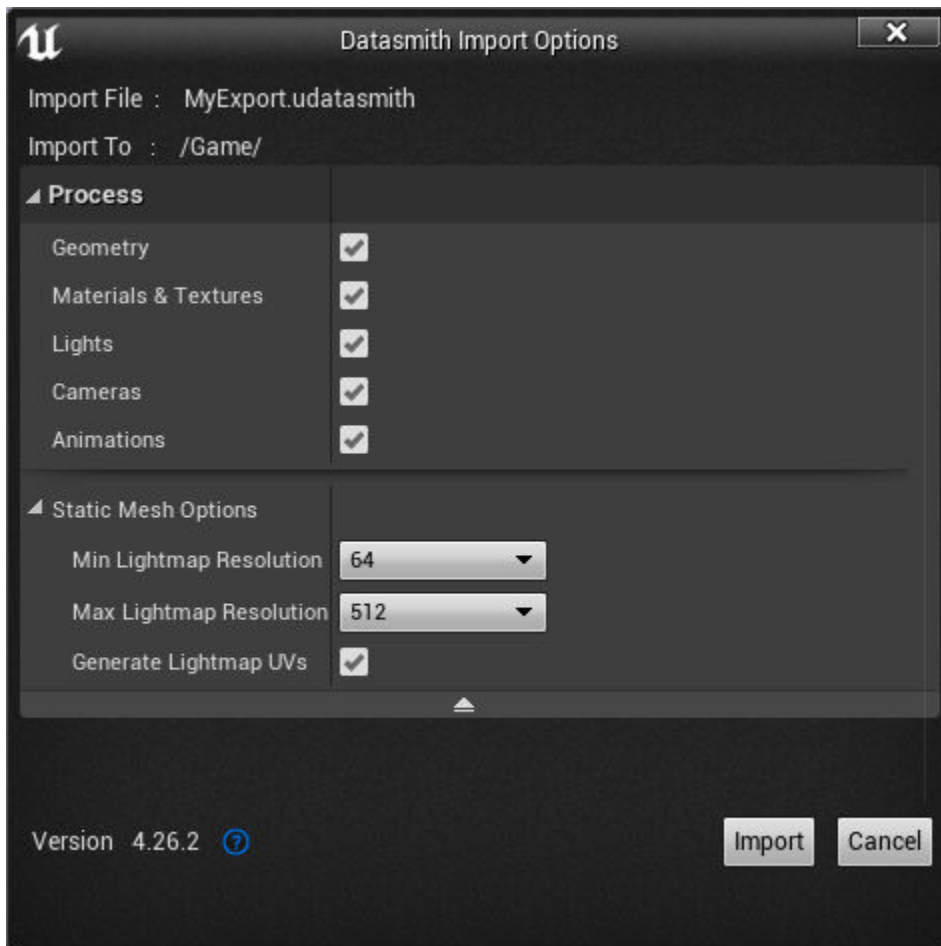
The Datasmith Import Options dialog box enables you to select which properties to import into the Unreal Editor. These options include:

- Geometry
- Materials
- Lights
- Cameras
- Animations

sRGB Textures

Unreal Engine does not support 16-bit sRGB textures. Therefore, textures appear to be washed out, unless you convert the texture files to 8-bit sRGB textures.

Exporting from RoadRunner to Unreal using Datasmith Road

Exporting a RoadRunner scene to Unreal using Datasmith Road enables you to export ASAM OpenDRIVE data along with the Datasmith file and also adds the metadata to Datasmith export. The

metadata stores signalization and ASAM OpenDRIVE IDs. The Datasmith Road export option reduces the import time for significantly large scenes into Unreal.

Follow these steps to export a scene from RoadRunner to Unreal using Datasmith Road :

1   Open your scene in RoadRunner.

2   Export the scene using the Unreal option. Select **File** > **Export** > **Datasmith Road (.udatasmith, .xodr)** from the menu bar.

3   In the Export Datasmith Road dialog box, set the mesh merging and tiling options, and then click **Export**.
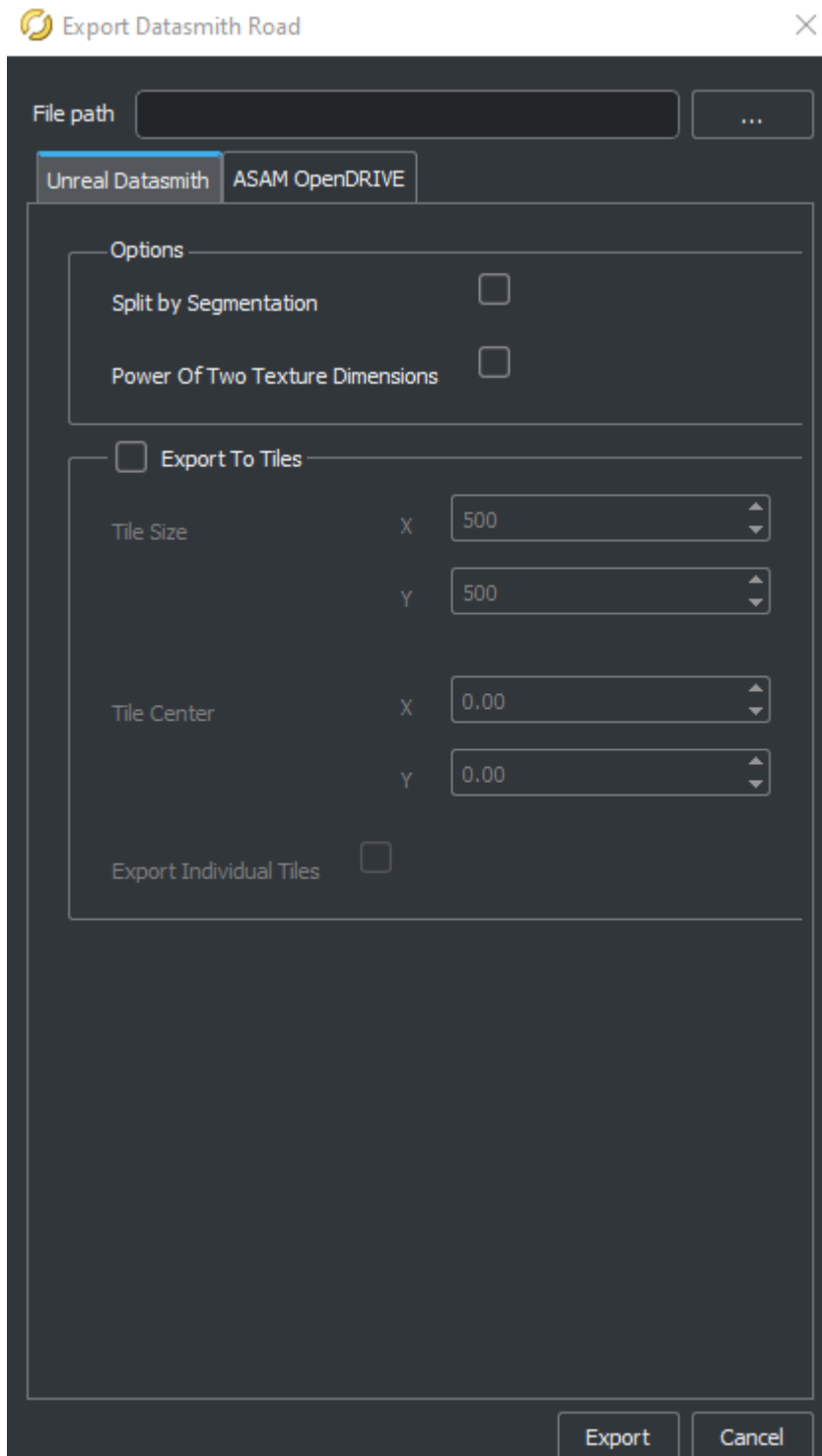
4 Select **Browse** to open the File dialog box and set the exported file's name and path. The Datasmith file exports to the specified folder.

- You can split the mesh by segmentation type. Meshes have `<segmentation type>Node` appended to their names.

- If the **Export To Tiles** option is selected, meshes are split per tile. Props are grouped by the tile they are in.

  - By default, only one file is exported. Tiles are stored in separate nodes.
  - If **Export Individual Tiles** is enabled, each tile are stored in its own Datasmith file.

Importing into Unreal using Datasmith Road

To import the scene into Unreal using Datasmith Road option follow these steps:

1 Right-click in the Content Browser window in Unreal. In the menu, select **Show Plugin Content** .
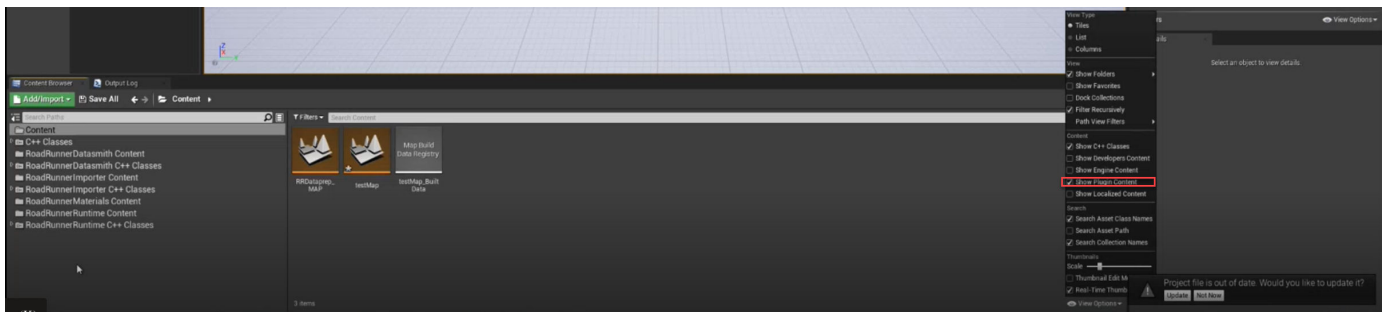
2 Click **RoadRunner Datasmith Content** . Then select the **Dataprep Asset** to handle importing with Datasmith and all the post processing steps.
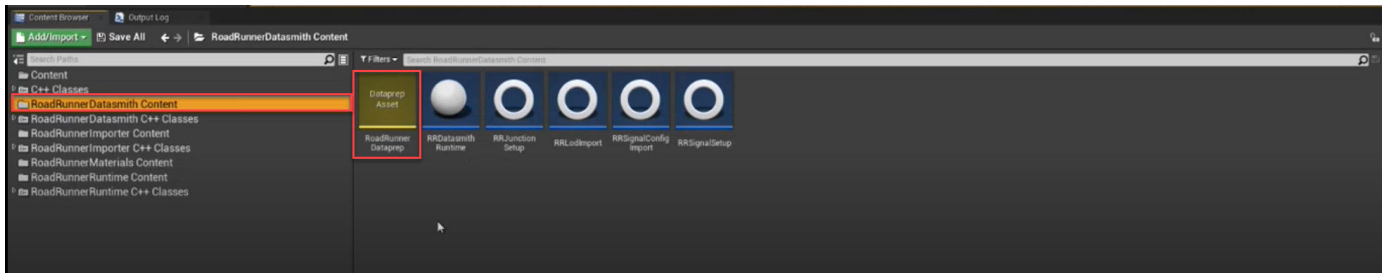
3 Click the **Import** button to import a scene into Unreal. A window, Datasmith Import, opens and enables you to select a file with the `.udatasmith` extension. You can select the destination folder in the current project. This loads the selected scene into Unreal. for example, this image loads the `FourWaySignal` scene file, which is one of the scenes present in the `Scenes` folder of a RoadRunner project.

4 Click **Execute** in the Editor toolstrip to run the post-processing steps for the imported scene.

5 Click **Commit** in the Editor toolstrip to commit these changes to the scene.

6 Click **Run** in the Editor toolstrip to test out the imported scene. For example, in the `FourWaySignal` scene imported with the Unreal using Datasmith Road option, the traffic signal lights are lit up because the plugin now allows signals to be controlled by their ASAM OpenDRIVE IDs.

Known Issues

In exported scenes, terrain sensors, or any other actor that uses the Unreal Engine line tracing API, might not detect hits near the road markings that have nonconvex shapes. To enable these detections, before exporting, float the road markings slightly above the road surface. From the RoadRunner menu, select **Tools > LOD Settings**. Then, in the LOD Settings dialog box, set **Marking Construction Type** to Floating and increase **Floating Marking Offset** to 0.005 meters. For more details on floating lane markings, see "Customize Levels of Detail in Exported Scenes" on page 5-111.

Limitations

- Exporting scenes to Datasmith format does not support **Split by Segmentation** and **Overhead Tile Rendering** LODs options simultaneously. By default, RoadRunner disables **Overhead Tile Rendering** LODs option.
- Exporting scenes to Datasmith format does not support **Prop Packing** LODs.

Export to Unreal Using Filmbox (.fbx) File

Unreal Overview

RoadRunner can export scenes to Unreal. The Unreal export option exports a Filmbox (.fbx) file and generates an additional XML file to hold extra data. The XML file holds data for materials and traffic signals in the scene.

On the Unreal side, a plugin is provided to help import the FBX file by using the information stored in the XML file. The plugin handles the following:

- Setting up materials

    - Material data is read in from the XML file and maps the data into new instance of one of the base materials included with the plugin.

    - Transparent materials will choose between the translucent and masked blend modes based on the transparency of the diffuse color.

- Adjusting the colliders in the imported static meshes

    - During import, newly created static mesh assets have their "Collision Complexity" property set to "Use Complex Collision As Simple".

- Setting up the traffic signal components:

    - Signal data is read in from the XML file and creates a component in the blueprint with the light bulb names set up during import.

    - The traffic signals will cycle through their phases during play mode.

    - The UUIDs prefixed in the scene components for prop instances are needed to reference the static mesh component in the traffic signal script during play mode, so signals will not work if their names are changed.

- Unreal software requirements: Unreal Version 4.17+

- Only `C++` based Unreal projects are supported.

Installing the Plugin

Follow the instructions in this section to install the Unreal plugin.

1 See "Downloading Plugins" on page 5-51 for instructions for downloading the latest version of the plugin.

2 Extract the RoadRunner Plugins zip file and locate the `RoadRunnerImporter`, `RoadRunnerRuntime`, and `RoadRunnerMaterials` folders under `Unreal/Plugins`.

> **Note** The Unreal plugin folder now also contains a `RoadRunnerCarla` integration plugin. Do not copy this folder if you are not using CARLA.
>
> The `RoadRunnerMaterials` plugin. Do not copy this folder if you are not using CARLA.

3 Copy the `RoadRunnerImporter`, `RoadRunnerRuntime`, and `RoadRunnerMaterials` folders into the `Plugins` folder under the project directory. If a `Plugins` folder does not exist, create one.

Name	Type	Size
Config	File folder	
Content	File folder	
Intermediate	File folder	
Plugins	File folder	
Saved	File folder	
packageTest.uproject	Unreal Engine Proj...	1 KB

4 Rebuild the plugin.

   a Generate the project files.

- Windows - Right-click the .uproject file and select "Generate Visual Studio project files."
- Linux - Run this code at the command line:

  ```
  $UE4_ROOT/GenerateProjectFiles.sh -project="<Path to .uproject file>" -game -engine
  ```

  Set UE4_ROOT to your Unreal Engine install directory.

   b Open the project and build the plugins by clicking **Yes**.

Missing CarlaDemo Modules

The following modules are missing or built with a different engine version:

    RoadRunnerImporter
    RoadRunnerRuntime

Would you like to rebuild them now?

                                       [Yes]   [No]
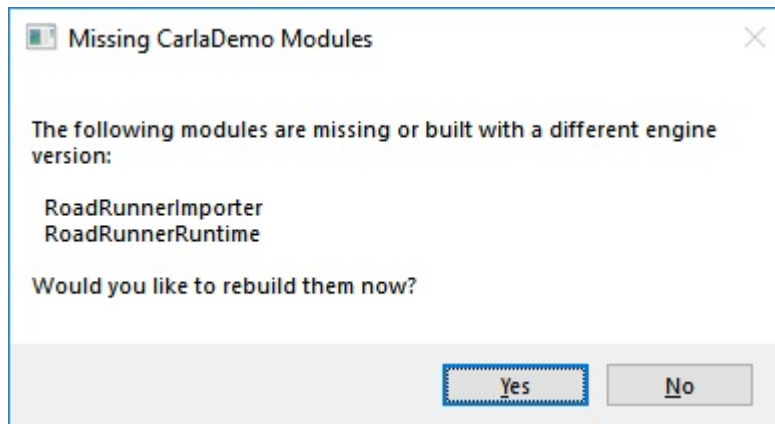
5 The plugin shows up under **Edit > Plugins**. If it does not appear in that menu, check that the **Enabled** parameter is selected.

> **Note** Ensure that the `RoadRunnerMaterials` plugin is enabled.

Plugin Contents

- RoadRunnerImporter module:

  - Overrides the default FBX importer when the metadata file is present
  - Option to overwrite default materials with new materials using the metadata file
  - Import signal data and timing
- RoadRunnerRuntime module:

- Contains component to control traffic signal visuals
- RoadRunnerMaterials plugin:
  - Base materials to create instances from

Exporting from RoadRunner to Unreal

Follow these steps to export a scene from RoadRunner to Unreal:

1 Open your scene in RoadRunner.

2 Export the scene using the Unreal option. Select **File > Export > Unreal (.fbx + .xml)** from the menu bar.

3 In the Export Unreal dialog box, set the mesh merging and tiling options, and then click **Export**.
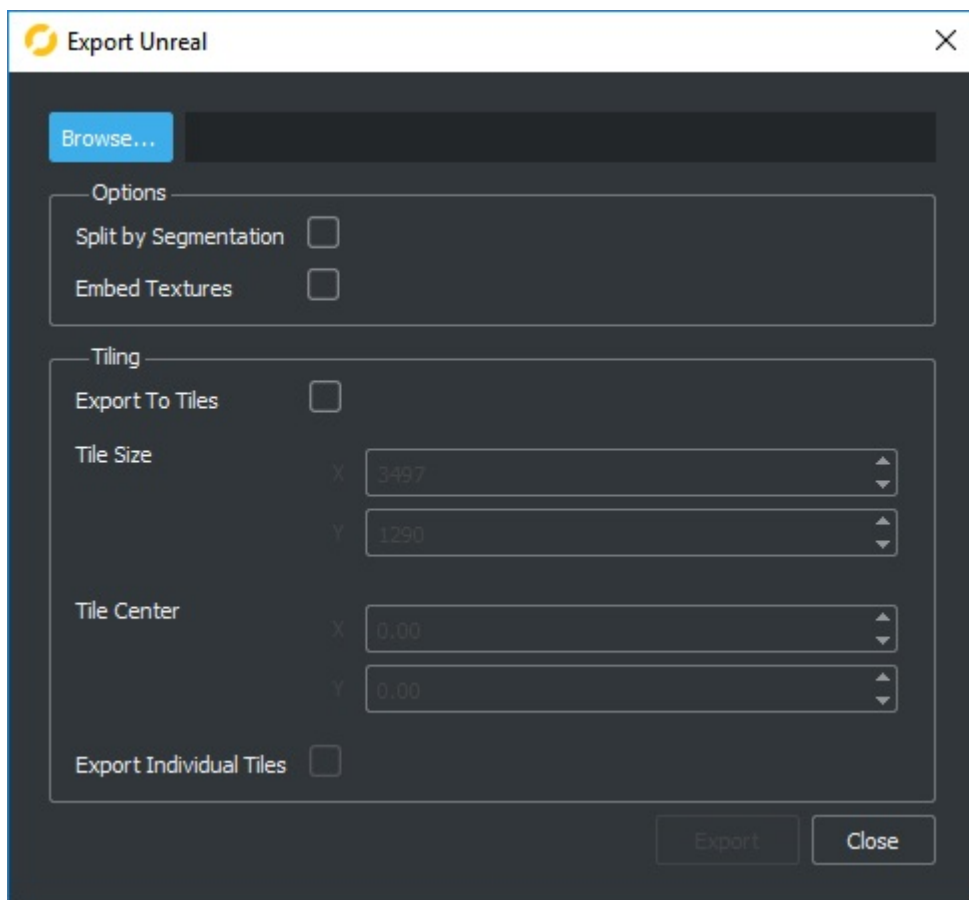
4 Browse to open the file dialog box to set the exported file's name and path. The FBX, textures, and XML files are exported to the same folder.

- The mesh can be split by segmentation type. Meshes have "<segmentation type>Node" appended to their names.
- If the **Export To Tiles** option is selected, meshes are split per tile. Props are grouped by the tile they are in.

- By default, only one file will be exported. Tiles will be stored in separate nodes.
- If **Export Individual Tiles** is enabled, each tile will be stored in its own FBX file.

Importing into Unreal

There are multiple ways to import the scene into Unreal:

- Drag the file into the Content Browser.
- Use the "Import" button and select the FBX file.

The plugin checks if there is a RoadRunner XML file associated with the imported file and imports as normal if a corresponding XML file is not found.

Selecting **File > Import Into Level** does not use the exported RoadRunner XML and uses the Unreal importer instead.

Prop Instances are prefixed by their UUID so that the traffic signal controller has a way to identify which signals it controls.

When the RoadRunner Import Options Dialog Box Opens
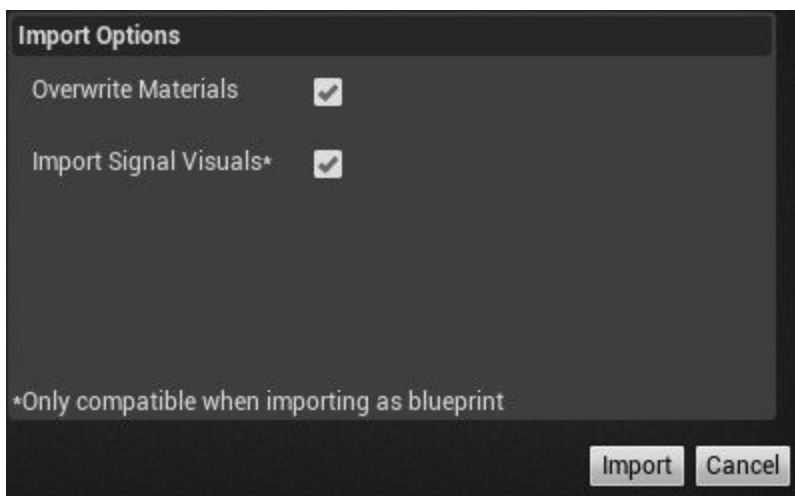
- Overwrite Materials

  - Overrides the default material importing
  - Needs to be unchecked if you want to set the materials to "Use Existing" in the next dialog box
- Import Signal Visuals

  - Functional only when "Create one Blueprint asset" is selected in the next dialog box

When the FBX Scene Import Options Dialog Box Opens

1   Set **Scene > Hierarchy Type** to `Create one blueprint asset` (selected by default).

Note   Only the `Create one blueprint asset` import option works with materials, signals, and transparency sorting. The `Create one Actor with Components` and `Create Level`

`Actors` options import only materials. If your scene contains traffic signals, use the `Create one blueprint asset` import option.

2   Select **Texture > Invert Normal Maps** if needed. This texture option inverts the orientation of normal maps, because Unreal uses the DirectX normal map format, which inverts the *y*-axis relative to the OpenGL format.
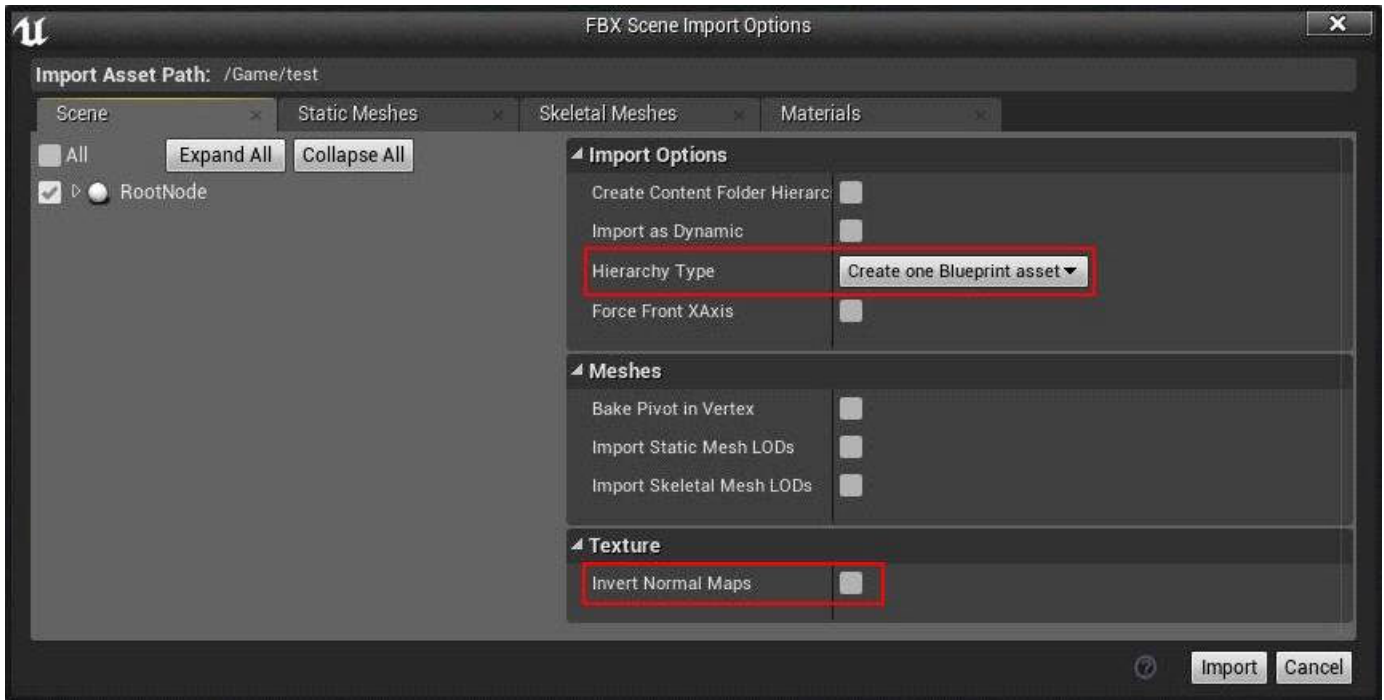
3   Set **Static Meshes > Normal Import Method** to `Import Normals`. Unreal generates its own normals, by default, to best suit its rendering needs. Because the Unreal method uses smoothing groups to calculate normals, which are not exported by RoadRunner, some geometries have hard edges when they should be smooth. Selecting this option enables smooth edges by importing the normals from the `.fbx` file, instead of using the default normals generated by Unreal.
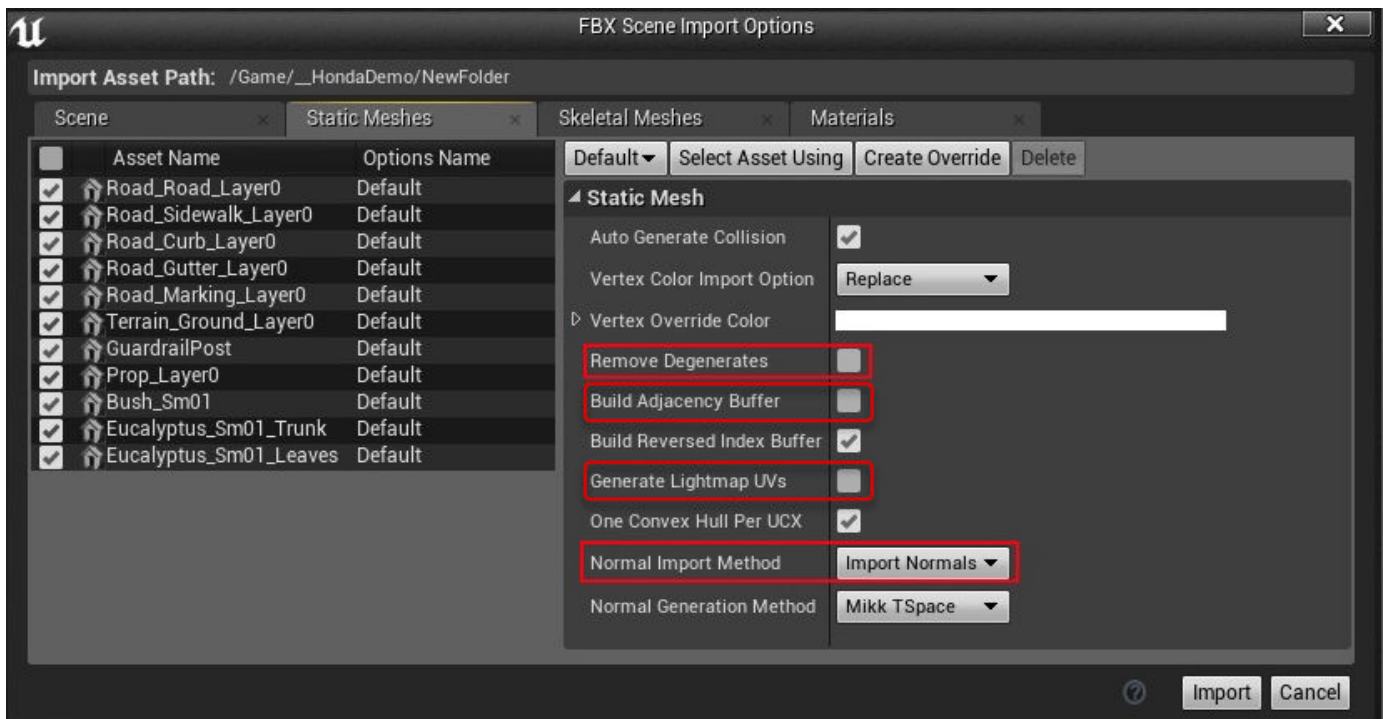
4. (Optional) Clear the **Remove Degenerates** parameter. Due to scaling between Maya® and Unreal, Unreal considers the mesh data for some props in RoadRunner to be very small, which causes Unreal to remove the geometry. Clearing this parameter prevents Unreal from unexpectedly removing these props.

5. (Optional) Clear the **Build Adjacency Buffer**. Selecting this parameter is required for PNT Tessellation, but it slows down the import process. Clearing this parameter for large meshes improves the performance of the import process.

6. (Optional) Clear the **Generate Lightmap UVs** parameter. The default lightmap resolution is too small for large terrains. Clearing this parameter enables the terrain to be lit dynamically.

7. Click **Import**.

About Importing Traffic Signals into Unreal

If traffic signals were set up in RoadRunner, they are imported into Unreal as RoadRunnerTrafficJunction components. These controllers are automatically created during import and included in the created blueprint.

The RoadRunnerTrafficJunction component handles the logic for switching between signal states. UUIDs are used to match to specific game objects in the scene.

FBX Details

The FBX file automatically splits the mesh by transparency sorting layer. This is due to Unreal storing "Translucency Sort Priority" on the static mesh component.

sRGB Textures

Unreal Engine does not support 16-bit sRGB textures. Therefore, textures appear to be washed out, unless the texture files are converted to 8-bit sRGB textures.

Large Scene Optimizations

Using the "Create one Actor with Components" option can be more efficient. However, signals will not be set up.

Importing Without the Plugin

This section covers fixes handled automatically by the RoadRunner plugin.

Fix Stretched Textures

Meshes with UV coordinates far away from the 0 to 1 range can cause issues in Unreal. On Static Mesh assets with this issue, the "Use Full Precision UVs" option can be set to fix it.
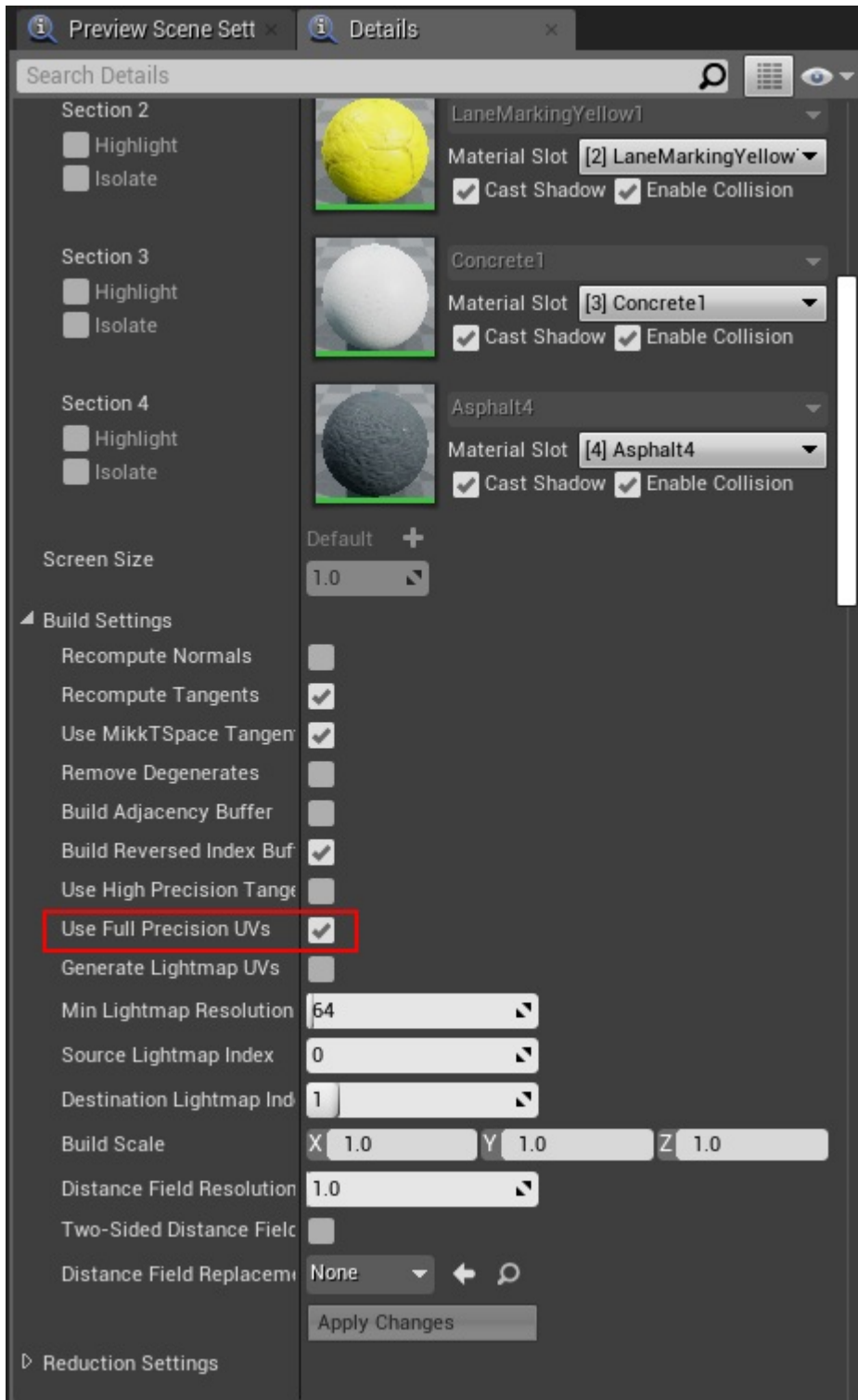
Fix Objects Floating Above the Road

Static Mesh assets need to have their Collision Complexity property set to "Use Complex Collision As Simple". Otherwise, collision boxes need to be manually added.
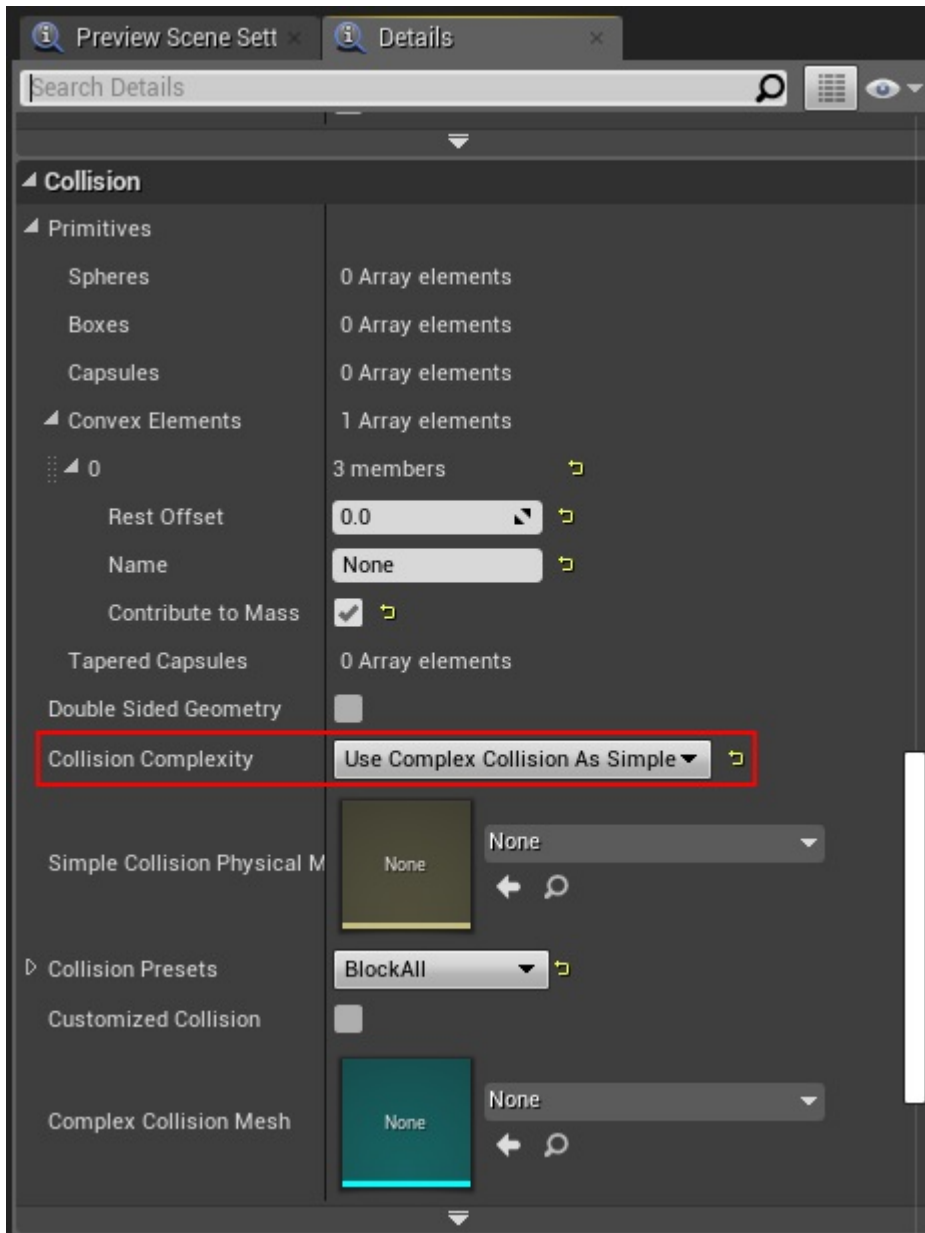
Known Issues

In exported scenes, terrain sensors or any other actor that uses the Unreal Engine line tracing API, might not detect hits near the road markings that have nonconvex shapes. To enable these detections, before exporting, float the road markings slightly above the road surface. From the RoadRunner menu, select **Tools > LOD Settings**. Then, in the LOD Settings dialog box, set **Marking Construction Type** to `Floating` and increase **Floating Marking Offset** to approximately `0.005` meters. For more details on floating lane markings, see "Customize Levels of Detail in Exported Scenes" on page 5-111.

Export to CARLA

CARLA Export Overview

RoadRunner can export scenes to the CARLA simulator. The CARLA export provides two options:

- **CARLA**: Exports an Unreal Datasmith (`.udatasmith`) file and an ASAM OpenDRIVE (`.xodr`) file.
- **CARLA Filmbox** : Exports a Filmbox (`.fbx`) file, an XML for some metadata, and an ASAM OpenDRIVE (`.xodr`) file. The XML file holds data for materials in the scene.

On the CARLA or Unreal side, a plugin is provided to help import the exported scene from RoadRunner.

For the scene exported using the **CARLA plugin**, the plugin provided on the Unreal side handles the following:

- Setting up materials

  - Material data is read in from the Datasmith (`.udatasmith`) file and maps the data into a new instance of one of the base materials included with the plugin.
  - Transparent materials will choose between the translucent and masked blend modes based on the transparency of the diffuse color.
- Adjusting the colliders in the imported static meshes

  - During import, newly created static mesh assets have their "Collision Complexity" option set to "Use Complex Collision As Simple".
- Setting up the traffic signal visuals

  - Traffic signal logic is hooked up to the simulator.
- Software requirements

  - CARLA 0.9.13

For the scene exported using the **CARLA Filmbox plugin**, the plugin provided on the Unreal side helps to import the FBX file by using the information stored in the XML file. The plugin handles the following:

- Setting up materials

  - Material data is read in from the XML file and maps the data into a new instance of one of the base materials included with the plugin.
  - Certain materials will instantiate from one of the CARLA materials.
  - Transparent materials will choose between the translucent and masked blend modes based on the transparency of the diffuse color.
- Adjusting the colliders in the imported static meshes

  - During import, newly created static mesh assets have their "Collision Complexity" option set to "Use Complex Collision As Simple".
- Setting up the traffic signal visuals

  - Traffic signal logic is not hooked up to the simulator.

- Software requirements

  - CARLA 0.9.13

Installing the Plugins

Follow the instructions in this section to install the Unreal plugin:

1 Build CARLA from its source. For more information, see the Windows build page of the Building CARLA instructions.

2 See "Downloading Plugins" on page 5-51 for instructions for downloading the latest version of the plugin.

3 Extract the RoadRunner Plugins zip file and locate the `RoadRunnerImporter`, `RoadRunnerCarlaIntegration`, `RoadRunnerRuntime`, `RoadRunnerDatasmith`, `RoadRunnerCarlaDatasmith`, and `RoadRunnerMaterials` folders under "Unreal/Plugins".

4 Copy the `RoadRunnerImporter`, `RoadRunnerCarlaIntegration`, `RoadRunnerRuntime`, `RoadRunnerDatasmith`, `RoadRunnerCarlaDatasmith`, and `RoadRunnerMaterials` folders into the `Plugins` folder under the `CarlaUE4` project directory, located at `<carla>/Unreal/CarlaUE4/Plugins` (next to the `Carla` folder).

5 Rebuild the plugin. First, generate the project files.

- If you are on Windows, right-click the `.uproject` file and select `Generate Visual Studio project files`.

- If you are on Linux, run this code at the command line:

  ```
  $UE4_ROOT/GenerateProjectFiles.sh -project="<CarlaFolderPath>/Unreal/CarlaUE4/CarlaUE4.uproject" -game -engine
  ```

  Set `UE4_ROOT` to your Unreal Engine install directory.

Then, open the project and build the plugins. If you are on Windows, run "make launch" in **x64 Native Tools Command Prompt for VS 2019** to compile the plugin and launch the editor.

6 The plugins shows up under **Edit > Plugins**. If it does not appear in that menu, check that the **Enabled** check box is on.
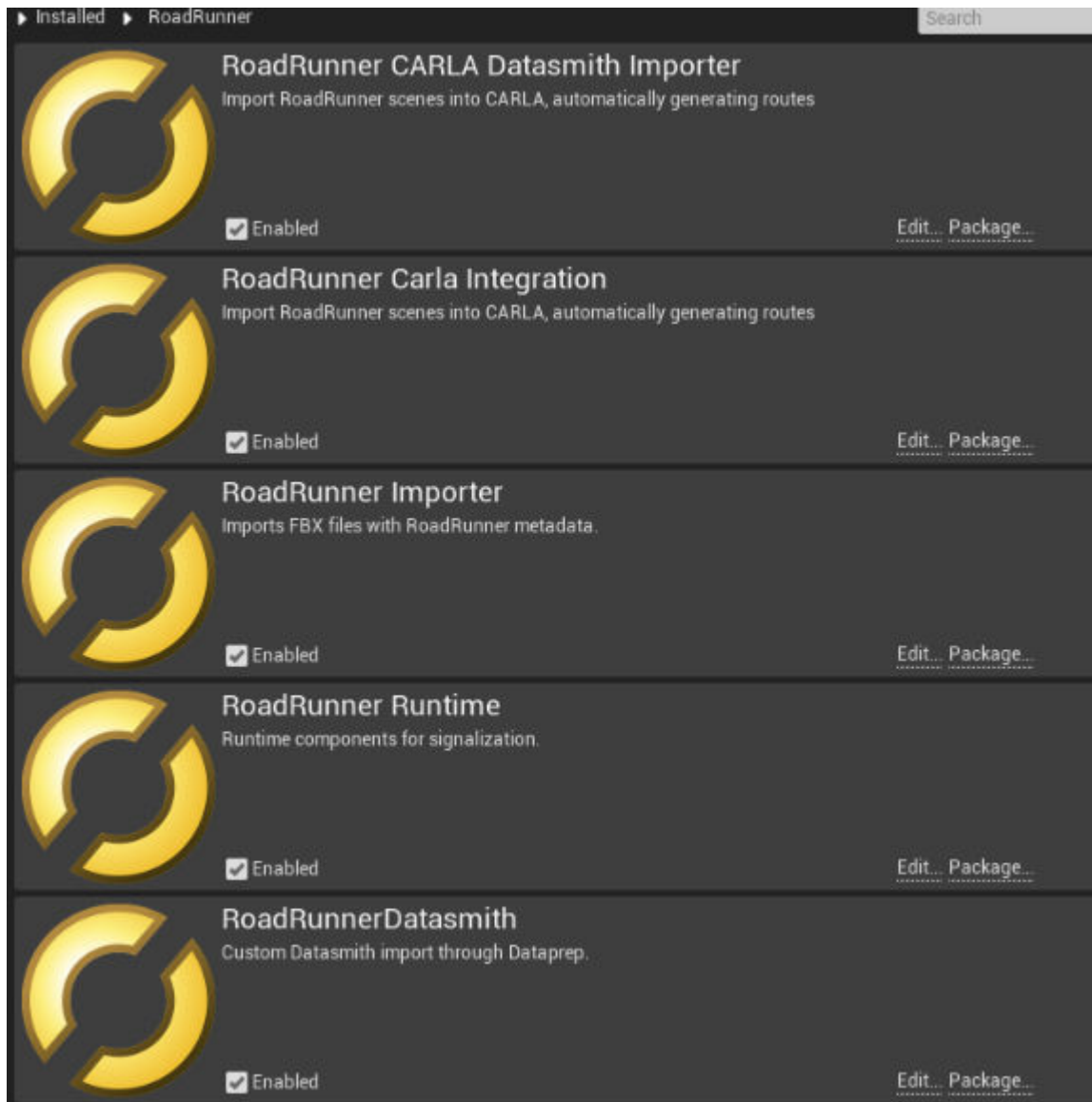
Plugin Contents

- RoadRunnerImporter module:

  - Overrides the default FBX importer when the metadata file is present
  - Option to overwrite default materials with new materials using the metadata file
  - Import signal data and timing

- RoadRunnerRuntime module:

  - Contains component to control traffic signal visuals

- RoadRunnerCarlaIntegration module:

  - Creates a new map and imports the FBX into the level
  - Moves static mesh assets based on segmentation type

- Creates materials instantiated from CARLA materials for weather effects
- Generates the routes from the ASAM OpenDRIVE file
- RoadRunnerMaterials plugin:

  - Base materials to create instances from
- RoadRunnerDatasmith plugin:

  - Dataprep asset that handles metadata post-processing
  - Imports signal data and timing
- RoadRunnerCARLADatasmith plugin:

  - Imports RoadRunner scenes into CARLA, automatically setting up traffic signals.

Exporting from RoadRunner to CARLA

Export Using CARLA Plugin

CARLA plugin (`.udatasmith + .xodr`). is the recommended method to export to CARLA. Exporting a RoadRunner scene to CARLA using CARLA (`.udatasmith + .xodr`) plugin enables you to export ASAM OpenDRIVE data along with the Datasmith file and also adds the metadata to Datasmith export. The metadata stores signalization and ASAM OpenDRIVE IDs. The CARLA export option reduces the import time for significantly large scenes into CARLA.

Follow these steps to export a scene from RoadRunner to Unreal using CARLA (`.udatasmith + .xodr`):

1 Open your scene in RoadRunner.
2 Export the scene using the CARLA (`.udatasmith + .xodr`) option. Select **File > Export > CARLA (.udatasmith, .xodr)** from the menu bar.
3 In the Export CARLA Road dialog box, set the mesh merging and tiling options, and then click **Export**.
4 Select **Browse** to open the File dialog box and set the exported file's name and path. The Datasmith file exports to the specified folder.

   - You can split the mesh by segmentation type. Meshes have *<segmentation type>*Node appended to their names.
   - If the **Export To Tiles** option is selected, meshes are split per tile. Props are grouped by the tile they are in.

     - By default, only one file is exported. Tiles are stored in separate nodes.
     - If **Export Individual Tiles** is enabled, each tile are stored in its own Datasmith file.

Export Using CARLA Filmbox

If you want to use export using the older pipeline, use the CARLA plugin (`.fbx + .rrdata.xml + .xodr`) option. Follow the steps below to export a scene from RoadRunner to CARLA using the CARLA Filmbox plugin (`.fbx + .rrdata.xml + .xodr`) option:

1 Open your scene in RoadRunner.
2 Export the scene using the CARLA option. Select **File > Export > CARLA Filmbox(.fbx + .xml + .xodr)** from the menu bar.

3   In the Export CARLA dialog box, set the mesh tiling on the **FBX** tab and the ASAM OpenDRIVE options on the **OpenDRIVE** tab as needed. Then, click **Export**.

4   Browse to open the file dialog box to set the exported file's name and path. The FBX, textures, XML, and ASAM OpenDRIVE files are exported to the same folder.

   • The mesh can be split by segmentation type. Meshes have "<segmentation type>Node" appended to their names.

   • If the **Export To Tiles** option is selected, meshes are split per tile and props are grouped by the tile they are in.

      • By default, only one file is exported. Tiles are stored in separate nodes.

      • If **Export Individual Tiles** is enabled, each tile is stored in its own FBX file.

   • **Note**   The plugin does not fully support the **Export Individual Tiles** option.

Importing into CARLA

Import to CARLA

Follow these steps to import the scene into CARLA if you exported the RoadRunner scene using CARLA (`.udatasmith + .xodr`) option:

1   Make a copy of the **BaseMap** and save it to the `Maps` folder. Rename the new map.
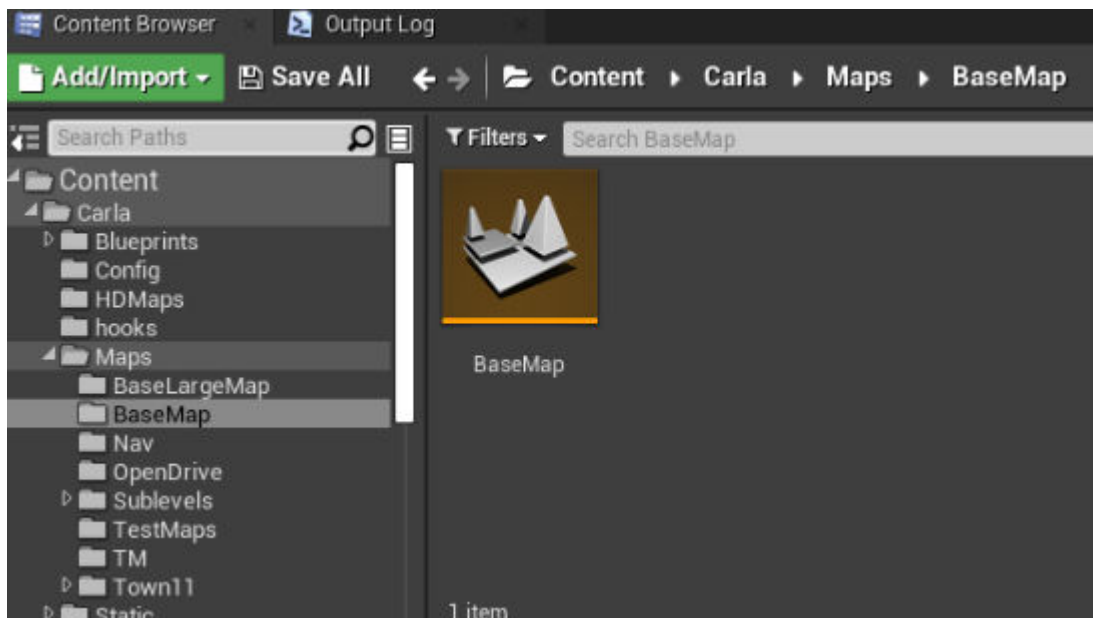
2   Open the new map that you created in the previous step.

3   Right-click in the **Content Browser** window in Unreal. In the menu, select **Show Plugin Content** .
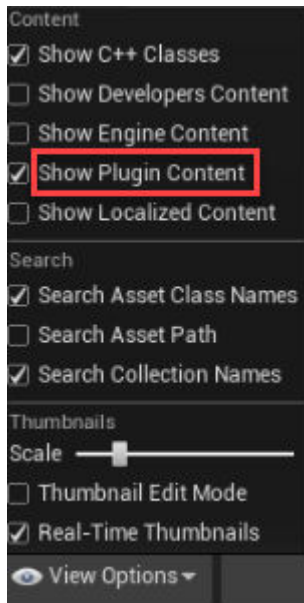
4 Click **RoadRunner CARLADatasmith Content** . Then select the **RRCARLADataprep Asset** to handle importing with Datasmith and all the post processing steps.
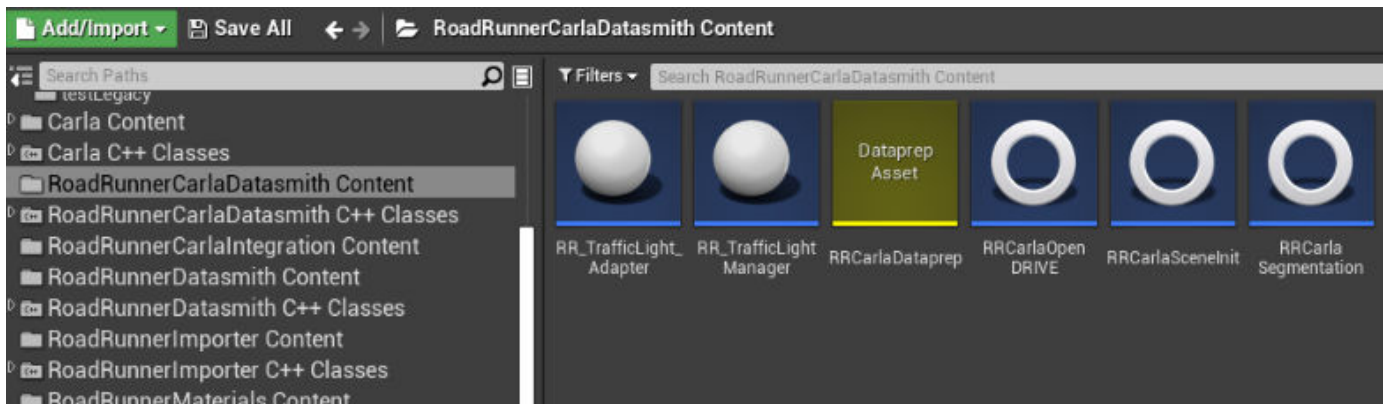
5 Click the **Import** button to import a scene into Unreal. For example, this image loads the `FourWaySignal` scene file, which is one of the scenes present in the `Scenes` folder of a RoadRunner project.

6 Click **Execute** in the Editor toolstrip to run the post-processing steps for the imported scene.

7 Click **Commit** in the Editor toolstrip to commit these changes to the scene.

8 Visualize the routes in the scene. Add an **Open Drive Actor** to the scene.
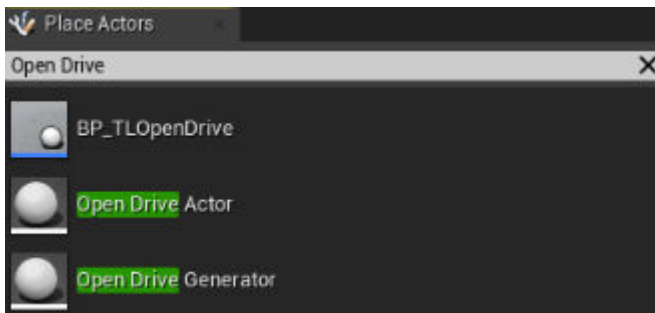
Select **Add Spawners** and click **Generate Routes**.
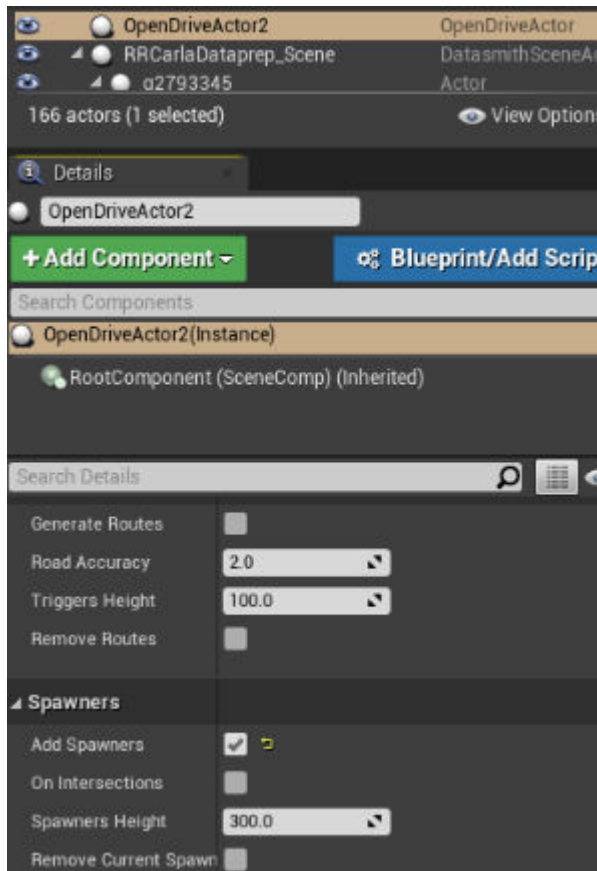
9 The scene is now imported and ready to simulate on.

Import to CARLA (Filmbox)

Follow these steps if you used the CARLA Filmbox plugin while exporting the RoadRunner scene.

- Drag the file into the Content Browser.
- Use the "Import" button and select the FBX file.

The plugin checks if there is a RoadRunner XML file associated with the imported file, and imports as normal if a corresponding XML file is not found.

Selecting **File** > **Import Into Level** does not use the exported RoadRunner XML and uses the Unreal importer instead.

When the RoadRunner Import Options Dialog Box Opens
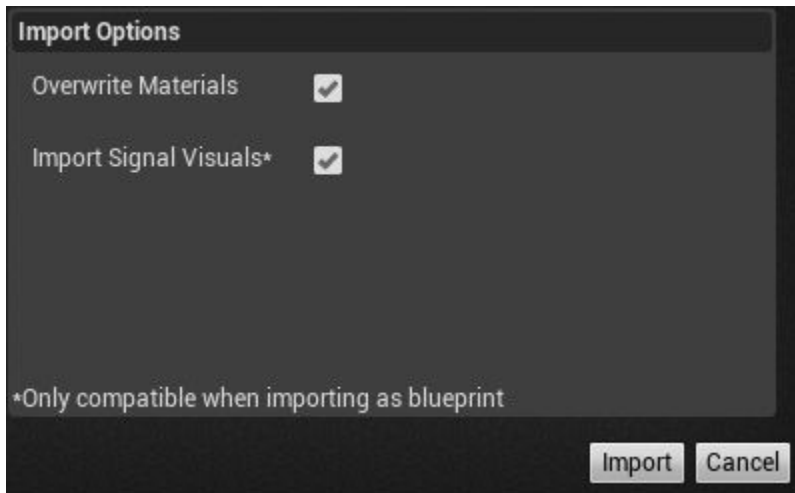
- Overwrite Materials

  - Overrides the default material importing. Instances from CARLA materials for roads and foliage.
  - Needs to be unchecked if you want to set the materials to "Use Existing" in the next dialog box.
- Import Signal Visuals

  - Only functional when the "Create one Blueprint asset" option is selected in the next dialog box.

Note The Importing Signal Visuals option does not have any effect on the traffic simulation.

When the FBX Scene Import Options Dialog Box Opens

1 Set **Scene > Hierarchy Type** to "Create One Blueprint Asset" (selected by default).

Note Only the "Create one Blueprint asset" import option works with materials, signals, and transparency sorting. The "Create one Actor with Components" and "Create Level Actors" options import only materials.

---

2 Select **Invert Normal Maps**, if needed.
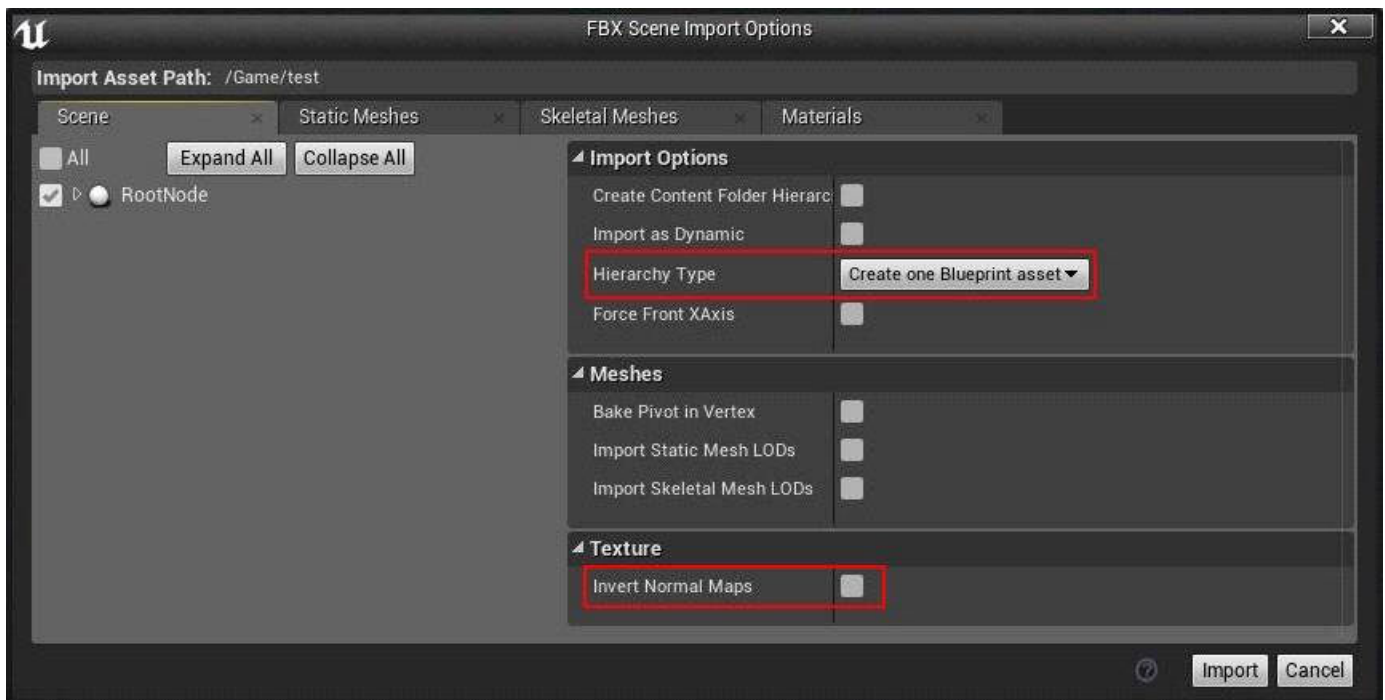
3 Set **Static Meshes > Normal Import Method** to "Import Normals".
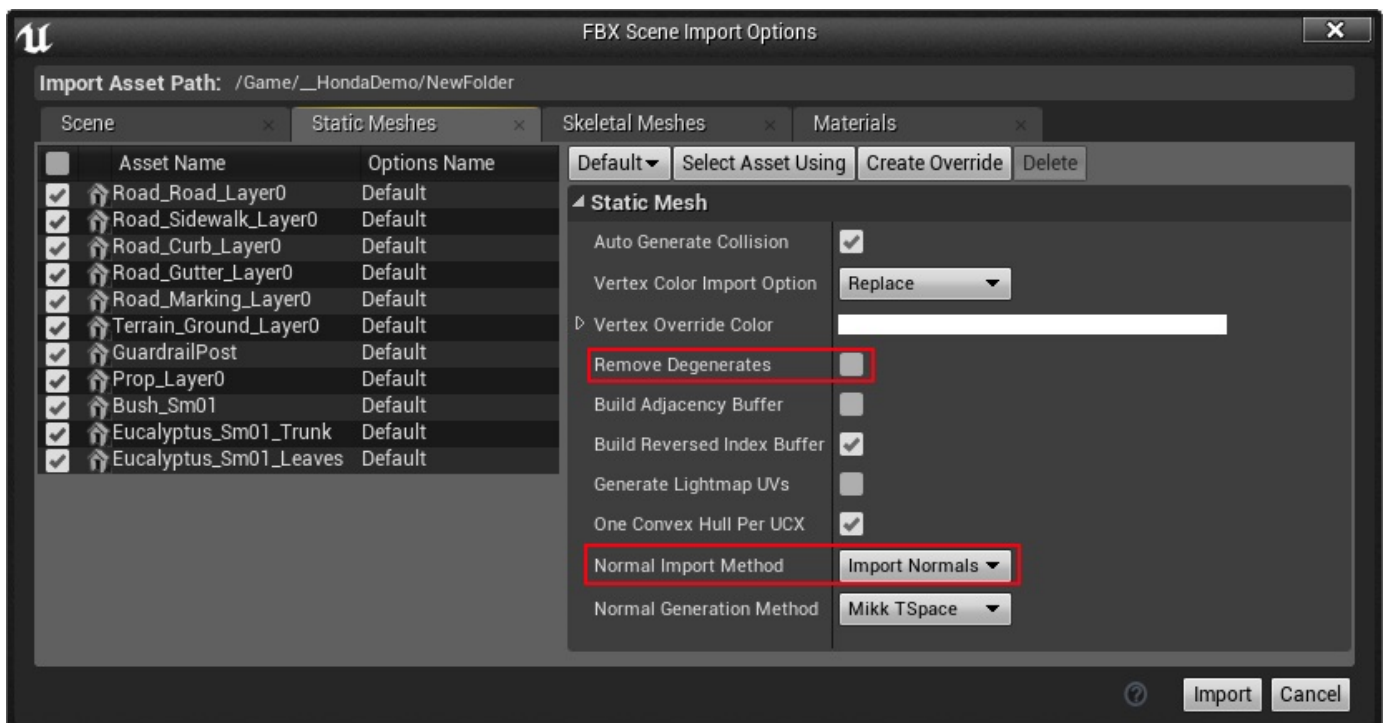
4 (Optional) Uncheck **Remove Degenerates**, which can help for some props created in a larger scale.

5 Click **Import**.

About Importing Traffic Signals into Unreal

If traffic signals were set up in RoadRunner, they are imported into Unreal as RoadRunnerTrafficJunction components. These controllers are automatically created during import and included in the created blueprint.

The RoadRunnerTrafficJunction component handles the logic for switching between signal states. UUIDs are used to match to specific game objects in the scene.

FBX Details

The FBX file will automatically split the mesh by segmentation and transparency sorting layer due to the following:

- Segmentation: CARLA determines segmentation by static mesh assets.
- Transparency sorting: Unreal stores the "Translucency Sort Priority" value on the static mesh component.

Testing the map

1 Click **Play** in the editor (the first time you click **Play** takes extra time to build the map).
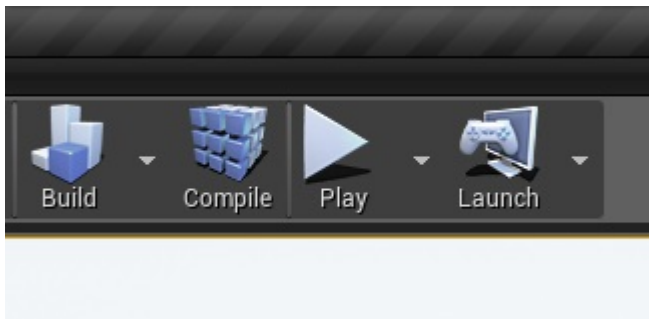
2 Run the example Python® scripts.
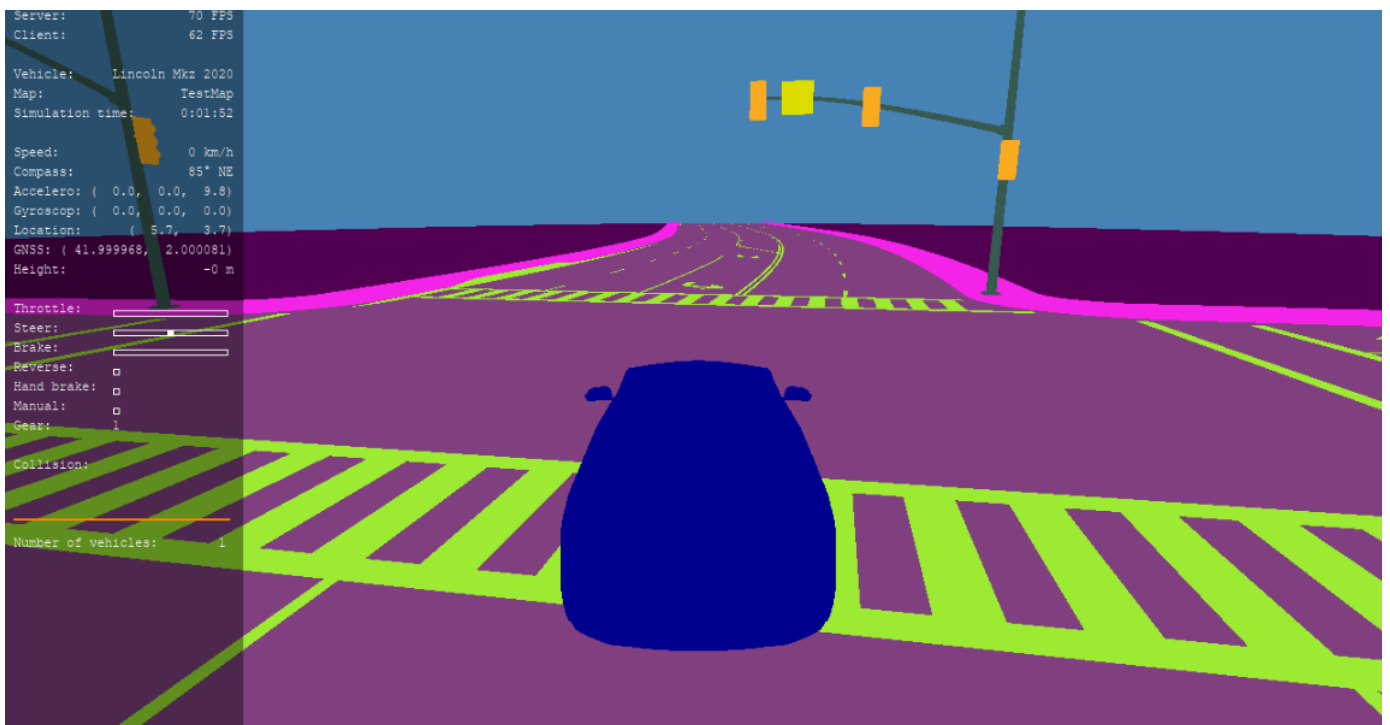
Export to VTD

RoadRunner has a combination exporter for exporting scenes to VIRES Virtual Test Drive (VTD)

RoadRunner provides a VTD export option that exports an ASAM OpenDRIVE (`.xodr`) file for the road network and an OpenSceneGraph (`.ive`) file for the visual scene.

Exporting to VTD

From the menu, select **File > Export > VTD (.xodr + .ive)**.

Export Options (ASAM OpenDRIVE)
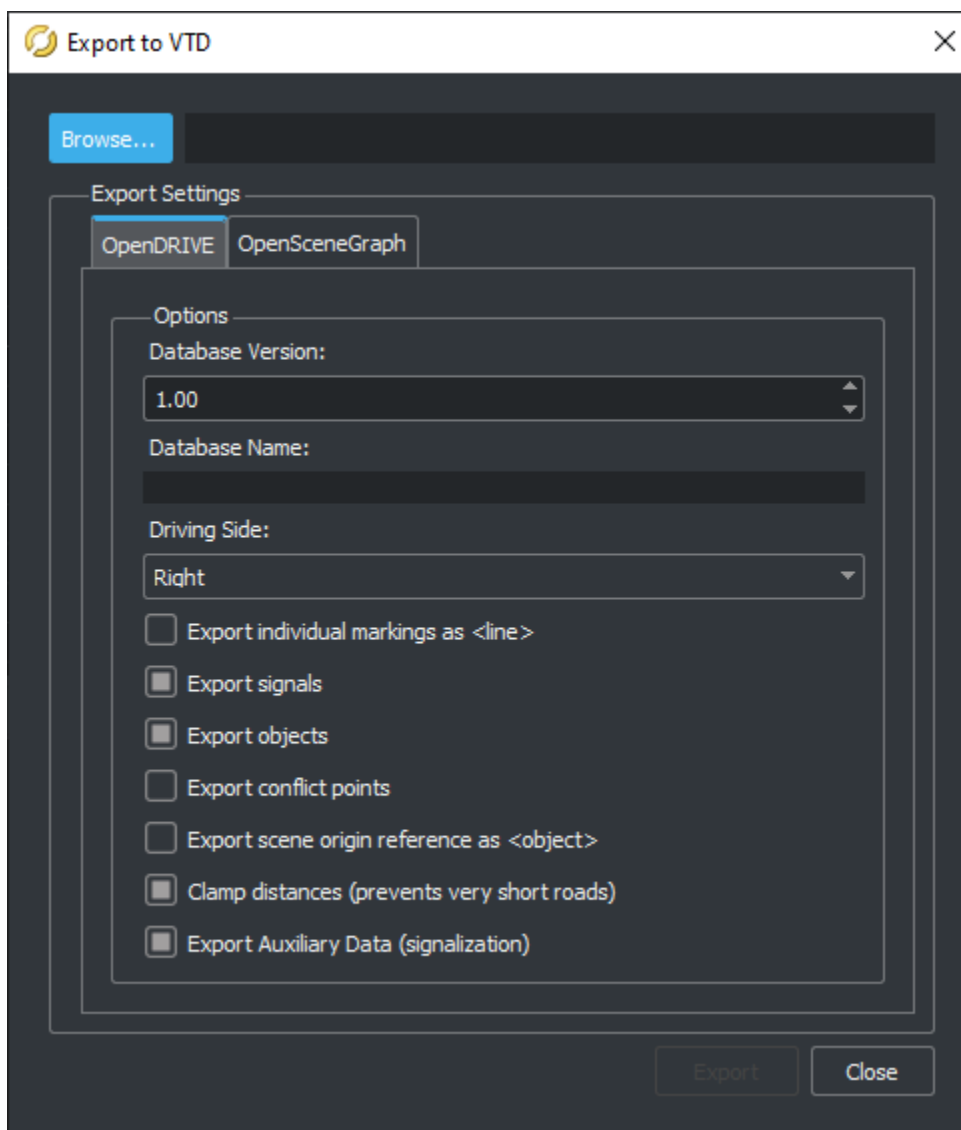
Export Options (OpenSceneGraph)
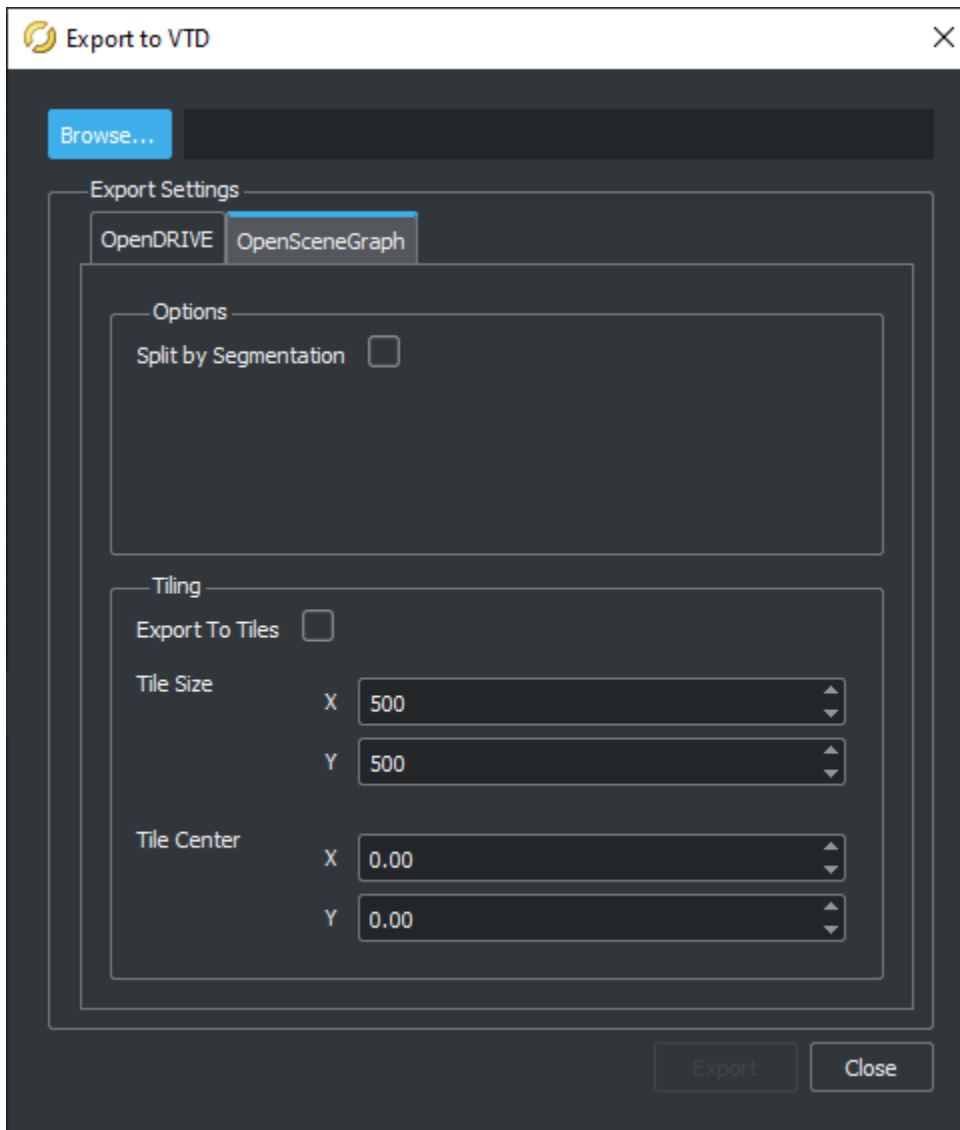
Split by Segmentation

This option will split meshes by their segmentation type on page 5-49.

Tiling

The **Export To Tiles** option splits the meshes per tile. This parameter also groups props by the tile they are in.

- By default, only one file is exported. Tiles are stored in separate nodes.
- VTD does not support scenes tiled to separate files.

Import into VTD

Once exported from RoadRunner, the scene can be imported into VTD.

1 (Optional) Convert the OpenSceneGraph IVE file to an OSGB file by using OpenSceneGraph version 3.2.3.

2 Copy the exported files (ASAM OpenDRIVE and IVE files) to your current project in the `Databases` folder. Placing the files in a separate folder is recommended.

3 Open VTD.
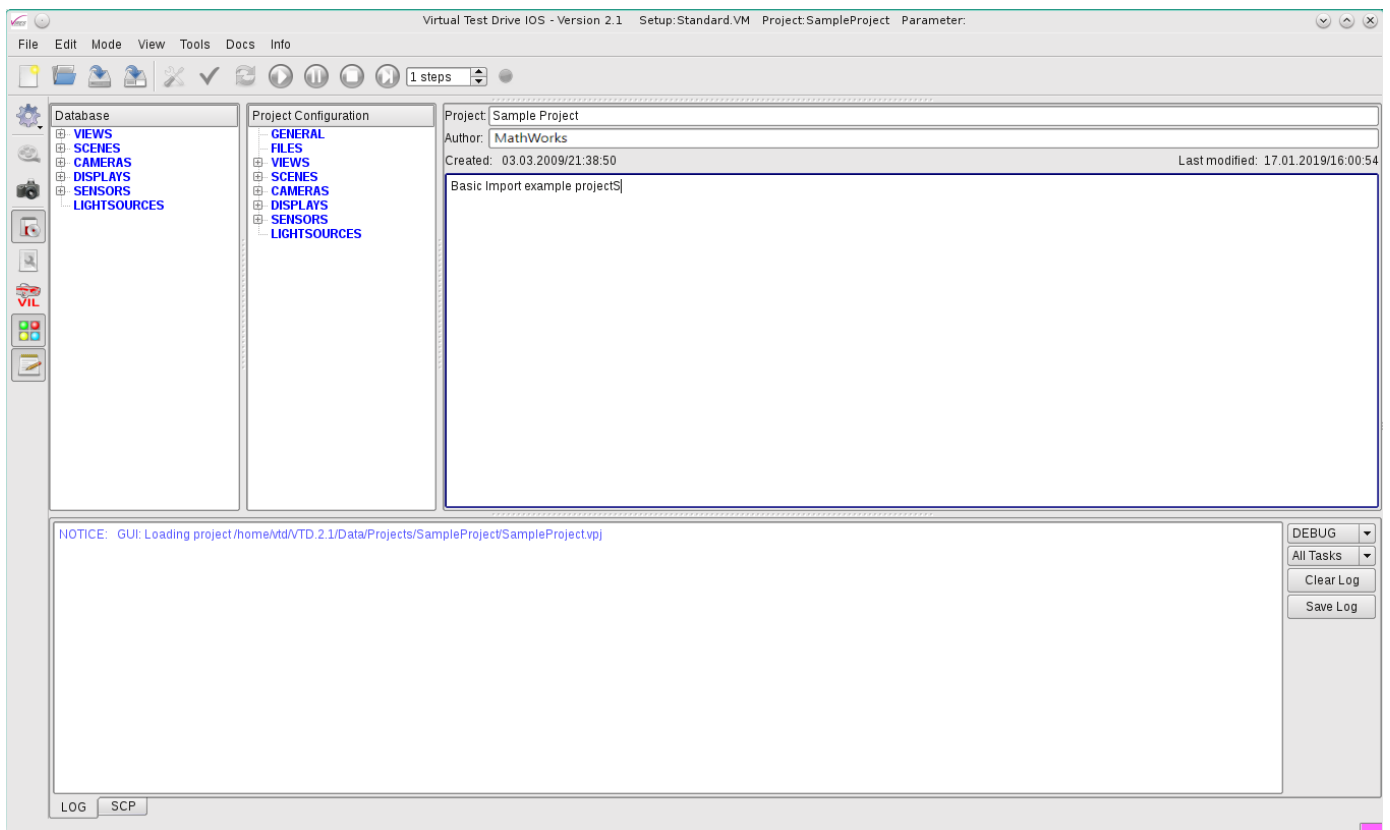
4 Open the OpenDRIVE Scenario Editor and click **New**.
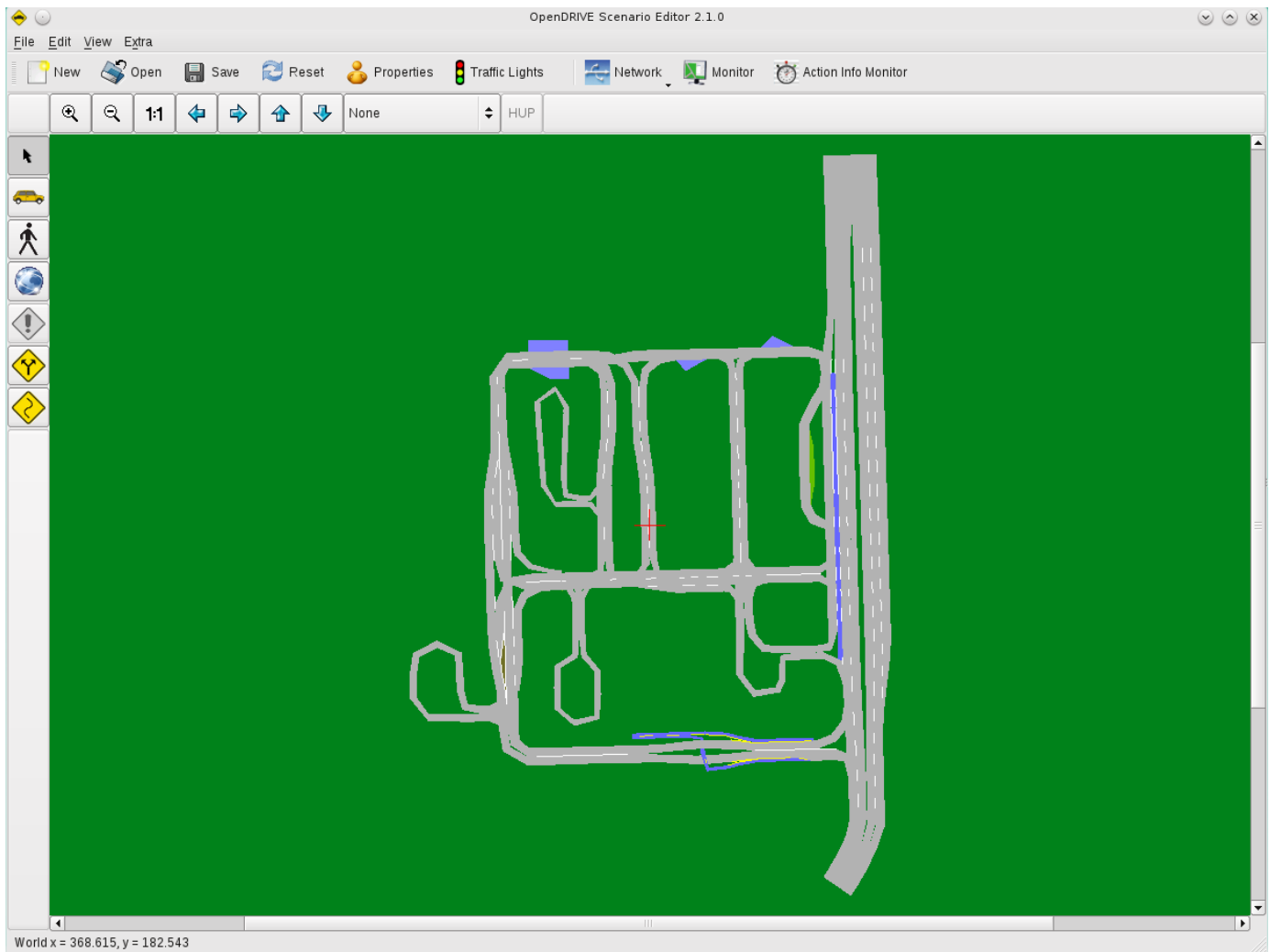
5   Click the **Properties** button.

6   Select the location of the **Layout File** (ASAM OpenDRIVE) and **Visual Database** (IVE).
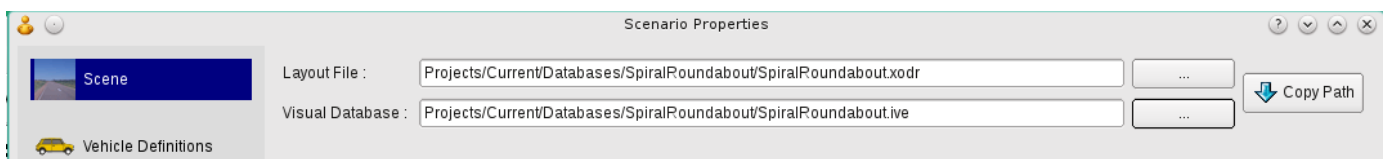
7   Insert at least an ego vehicle.

8   Save the scenario.

9   Select the scenario in VTD.
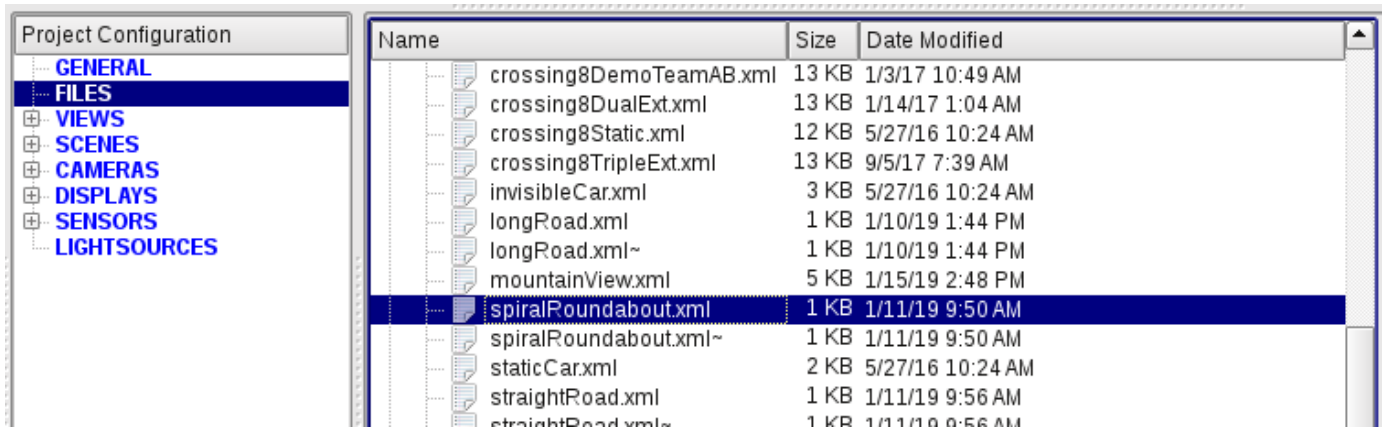
10 Run the scenario.

Limitations

Refer to "Export to OpenSceneGraph" on page 5-7 for further details on limitations.

Customize Levels of Detail in Exported Scenes

To improve simulation performance in exported scenes, you can set different levels of detail (LODs) to render at specified distances away from the simulator camera. For example, you can specify lower LODs at distances thousands of meters away, where the reduction in scene quality is acceptable. Export of LODs is supported for FBX files only.

Each LOD you add linearly increases the time it takes to export a scene. However, the substantial performance gains you can achieve by specifying differing LODs can make up for the increased export cost.

Set Levels of Detail in Scene

Open a scene that contains multiple LODs. From the **File** menu, select **Open Scene**. Then, select FourWaySignal.rrscene, which is one of the prebuilt scenes included with new RoadRunner projects.
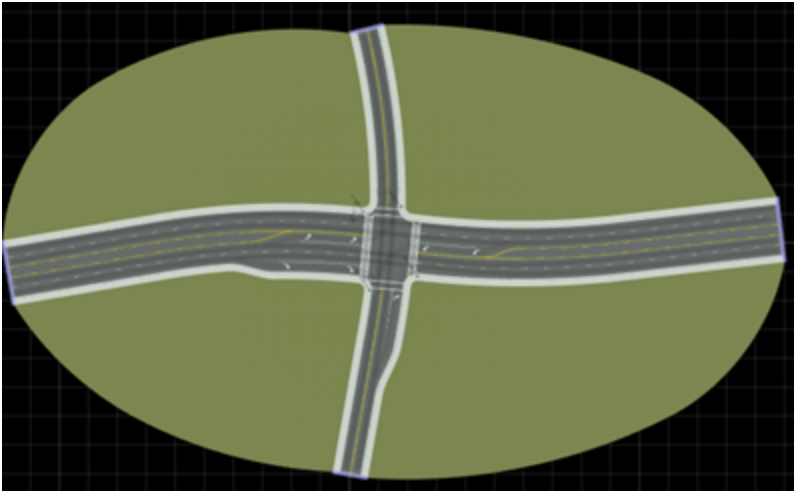
Open the LOD settings for the scene. From the **Tools** menu, select **LOD Settings**. In the left pane of the LOD Settings dialog box, the scene has three LODs: LOD 0, LOD 1, and LOD 2.
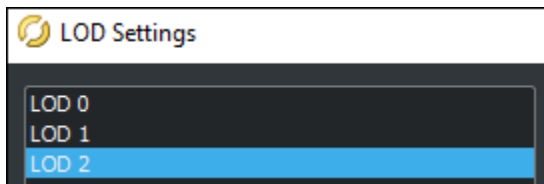
You can add new LODs by clicking **Add LOD**. Each LOD you add clones the settings from the previous LOD. RoadRunner does not limit the number of LODs you can add, but each LOD increases export time. The size of the export also increases, because RoadRunner exports a different rendering of the scene at each LOD.

Select the different LODs and notice the change in LOD settings in the right pane. The **LOD Distance** parameter controls the LOD settings to use at distances relative to the simulator camera. This table describes the **LOD Distance** settings for the FourWaySignal scene.

Level of Detail	LOD Distance	Description
LOD 0	1000.0	Use the defined LOD settings at ranges from 0 to 1000 meters away from the camera.
LOD 1	2000.0	Use the defined LOD settings at ranges (1000,2000] meters away from the camera.
LOD 2	3000.0	Use the defined LOD settings at ranges 3000 or more meters away from the camera.

The other dialog box parameters control scene triangulation, overhead scene rendering, and prop packing.

Export Highest Levels of Detail from a Scene

Open a scene that contains multiple LODs. From the **File** menu, select **Open Scene**. Then, select `FourWaySignal.rrscene`, which is one of the prebuilt scenes included with new RoadRunner projects.
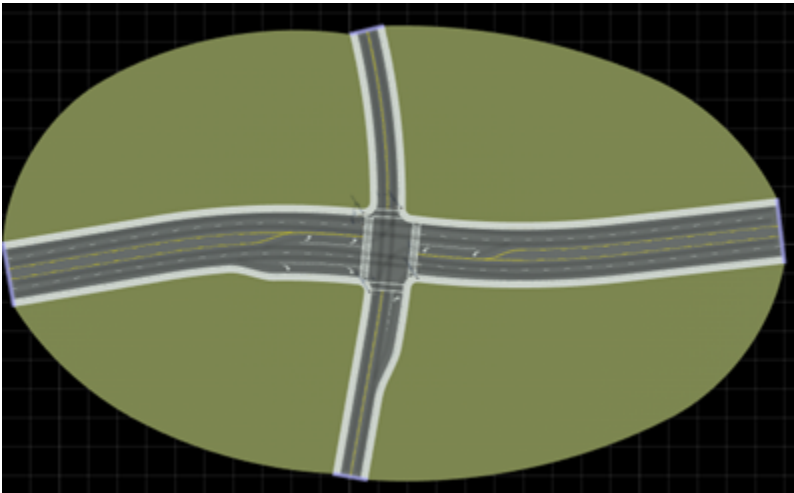
Click the **Scene Export Preview Tool** button .

You can export only the highest LODs by selecting **Export Only Highest LODs** from the **Attributes** pane. Selecting this option allows you to export only the highest LODs for all props in the scene. This parameter reduces the export time of the scene since other LODs are striped off from the scene, leaving only the highest LODs to be exported.

Export Only Highest LODs is also available as a menu option when exporting a scene to AutoCAD, Filmbox, glTF, OpenFlight, OpenSceneGraph, USD, CARLA, Metamoto, Unity, Unreal, Datasmith, and VTD. The STL and OBJ file formats use this option by default.

Modify Triangulation Settings

The LOD Settings dialog box includes several parameters that control the number of triangles used to render the scene geometry. Increasing these values reduces the amount of scene triangulation, which speeds up rendering performance at the expense of scene quality.

In the `FourWaySignal` scene, view how the triangulation decreases in the scene at the different LOD distances. First, press **F3** to display the shaded wireframe of the scene on the scene editing canvas. Then, in the LOD Settings dialog box, select **Preview Live** and change between LODs in the left pane. The amount of triangulation decreases significantly at each successive LOD.

LOD 0	LOD 1	LOD 2
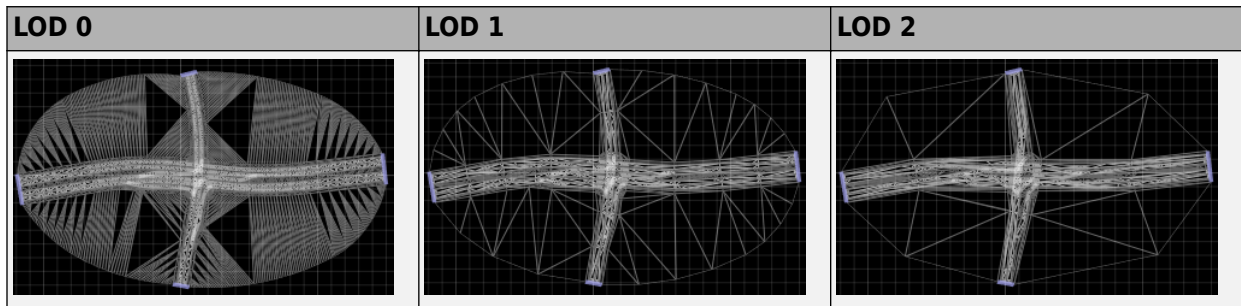
Press **F3** again to turn off the shaded wireframe.

Positional Tolerance Parameter

The **Positional Tolerance** parameter controls the maximum deviation in meters between sampled polylines and their underlying analytical representation. As this value decreases, the number of samples and triangles produced increases. This table shows the difference in triangulation at varying **Positional Tolerance** values.

Positional Tolerance = 0.005	Positional Tolerance = 0.01	Positional Tolerance = 0.1
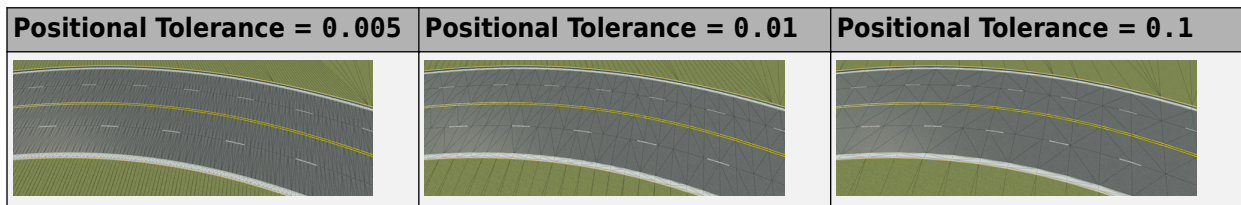
Max Length Parameter

The **Max Length** parameter controls the maximum length between any two samples in the curve triangulation. This table shows the difference in triangulation at varying **Max Length** values.

Max Length = 50.0	Max Length = 10.0	Max Length = 1.0
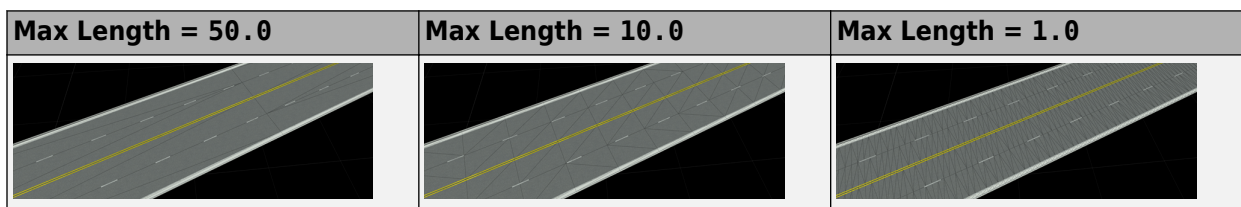
Height Field Sampling Parameters

The LOD Settings dialog box contains several parameters that affect the amount of triangulation used to render elevation changes in the scene.

The **Height Tolerance** parameter controls whether to include elevation samples in terrain triangulation relative to the elevation of neighboring samples. Roads are not included in the terrain triangulation. Height tolerance is in meters. As the tolerance decreases, the difference between samples and the underlying elevation field with respect to elevation decreases. Typically, a low tolerance produces a large number of triangles. This table shows the difference in triangulation at varying **Height Tolerance** values.

Height Tolerance = 0.5	Height Tolerance = 0.05	Height Tolerance = 0.005
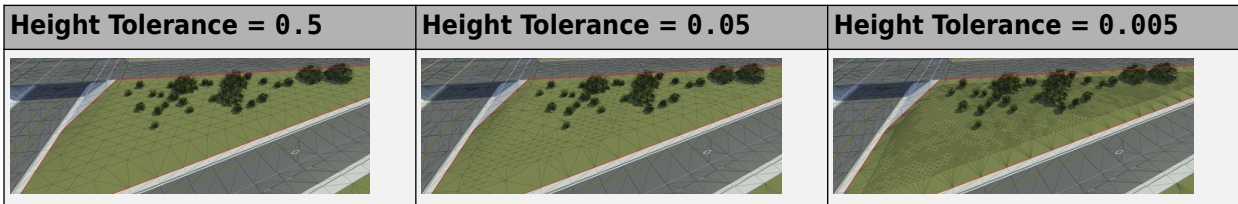
The **Height Min Spacing** parameter controls the minimum distance, in meters, between samples and their neighbors. In general, the lower the tolerance, the closer the samples are allowed to be. Typically, a lower tolerance produces a greater number of triangles. This table shows the difference in triangulation at varying **Height Min Spacing** values.

Height Min Spacing = 5.0	Height Min Spacing = 1.0	Height Min Spacing = 0.1
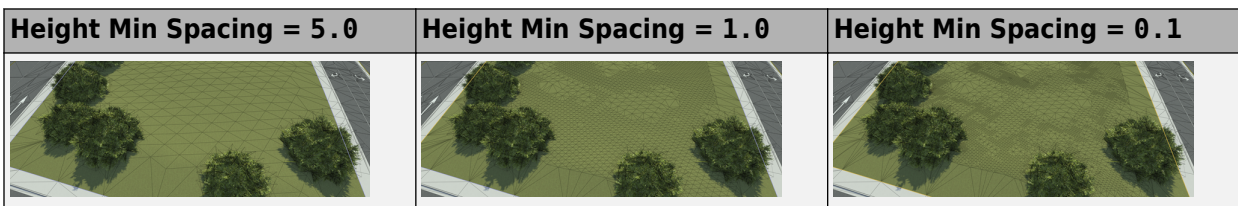
The **Height Max Spacing** parameter controls the maximum distance, in meters, between samples and their neighbors. In general, the lower the tolerance, the closer the samples are required to be. Typically, a lower tolerance produces a greater number of triangles. This table shows the difference in triangulation at varying **Height Max Spacing** values.

Height Max Spacing = 5.0	Height Max Spacing = 2.0	Height Max Spacing = 0.5
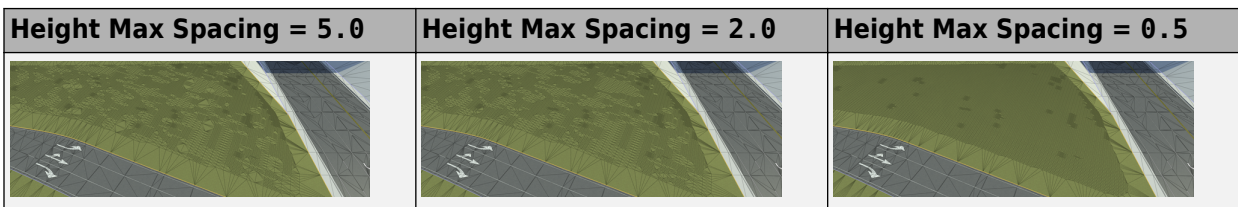
The **Road to Surface Blend Range** parameter controls the amount of distance, in meters, between the surface triangles and any adjacent edge. Typically, a lower blend range produces a greater number of triangles. This table shows the difference in triangulation at varying **Road to Surface Blend Range** values.

Road to Surface Blend Range = 5.0	Road to Surface Blend Range = 1.0
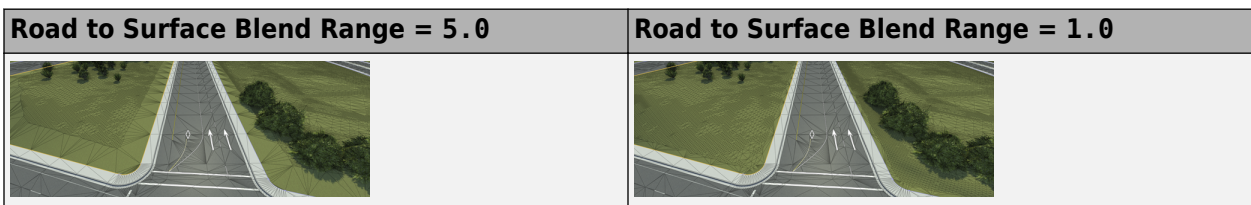
The **Uniform Road Sampling** parameter enables uniform tessellation and specifies sampling distance for roads and lanes. This allows samples to be taken from between the lane cross sections and lane boundaries. It also adds the ability to sample roads based on uniform values. Check the

Uniform Road Sampling button and specify the **Road Sample Spacing** to sample roads on uniform values.

Marking Projection Parameters

Every marking in RoadRunner is projected onto the underlying road or terrain surface to improve visual quality and avoid rendering issues such as z-fighting. The images in this table show the result of projecting markings onto a complex road surface.

Consider "SLOW" road marking stencils projected onto a road that has an exaggerated height bump.
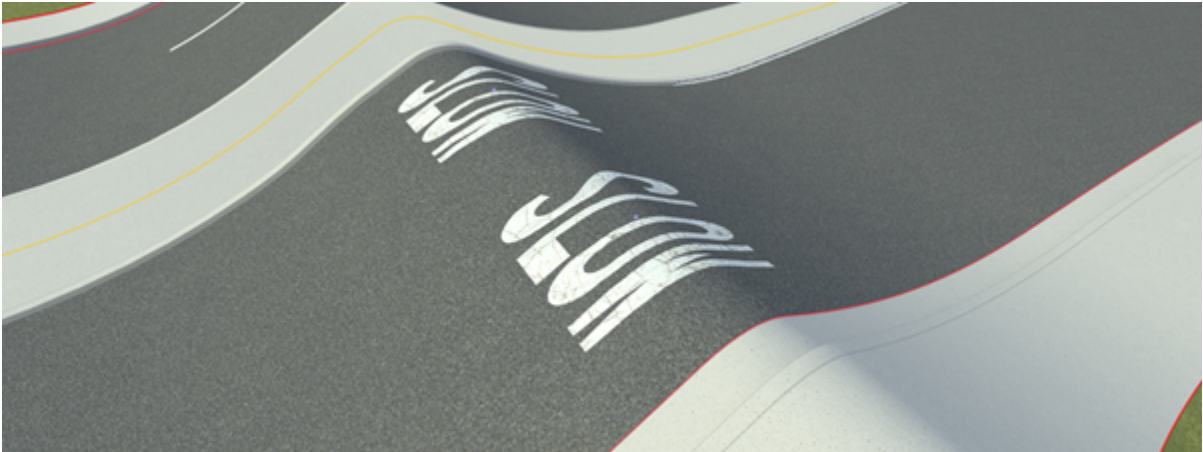
When you select one of the stencils by using the **Marking Point Tool** , the original flat stencil does not line up well with the underlying surface.
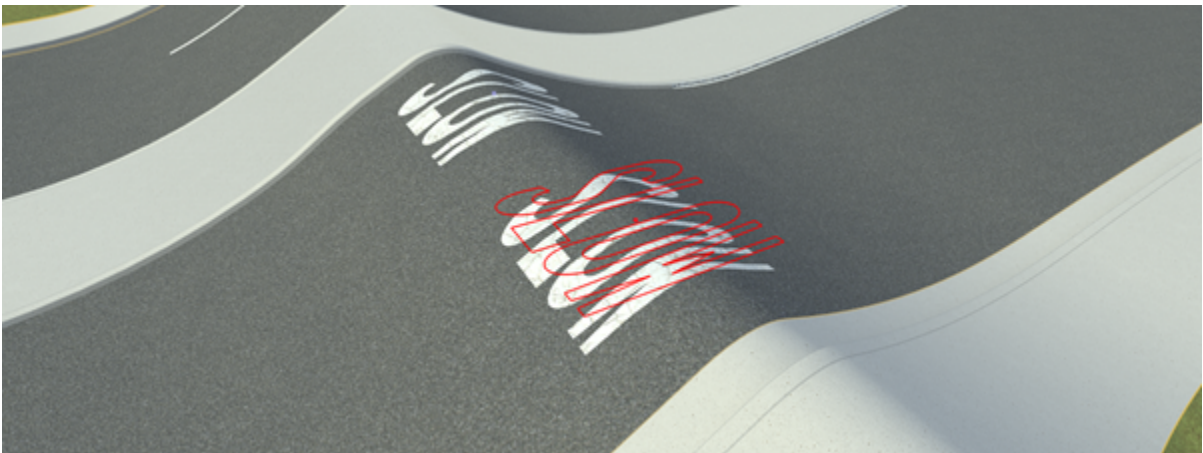
After adjusting the marking projection parameters in the LOD Settings dialog box, the triangulation of the marking outline matches the marking on the road surface.
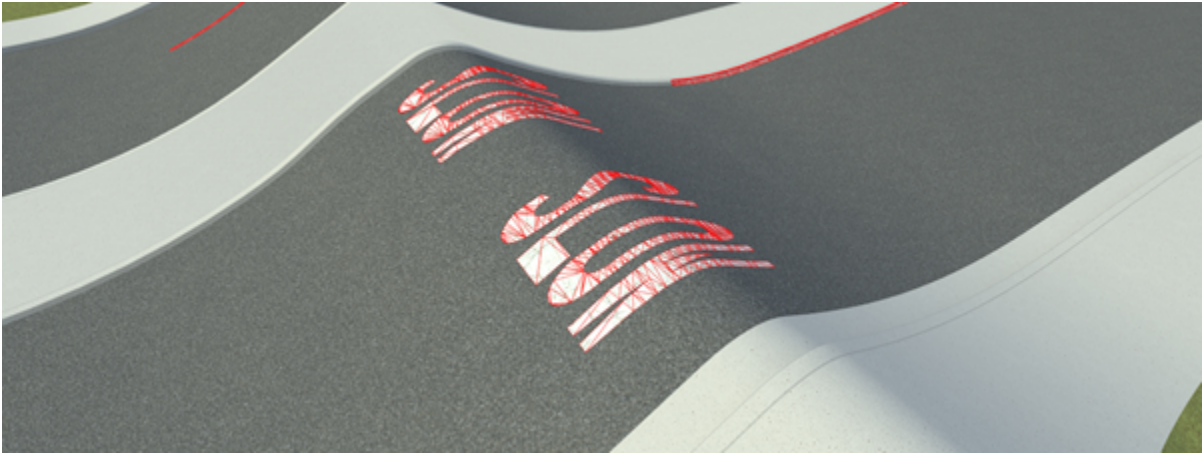
The **Marking Construction Type** parameter controls whether to stitch markings into the underlying road and terrain surface or to float the markings a small distance above the surface once they are projected. Floating markings, also known as decals, produce fewer triangles overall and allow for a more regular road surface triangulation. However, floating markings might require more handling once exported from RoadRunner to avoid z-fighting.
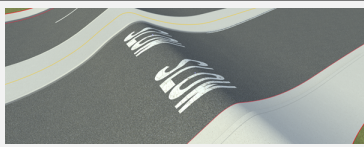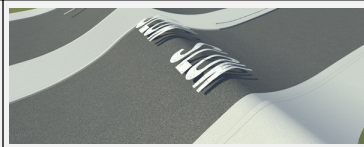
This image shows the sample markings produced when you set **Marking Construction Type** to `Cut Out`. The asphalt surface is triangulated around the "SLOW" stencil markings.
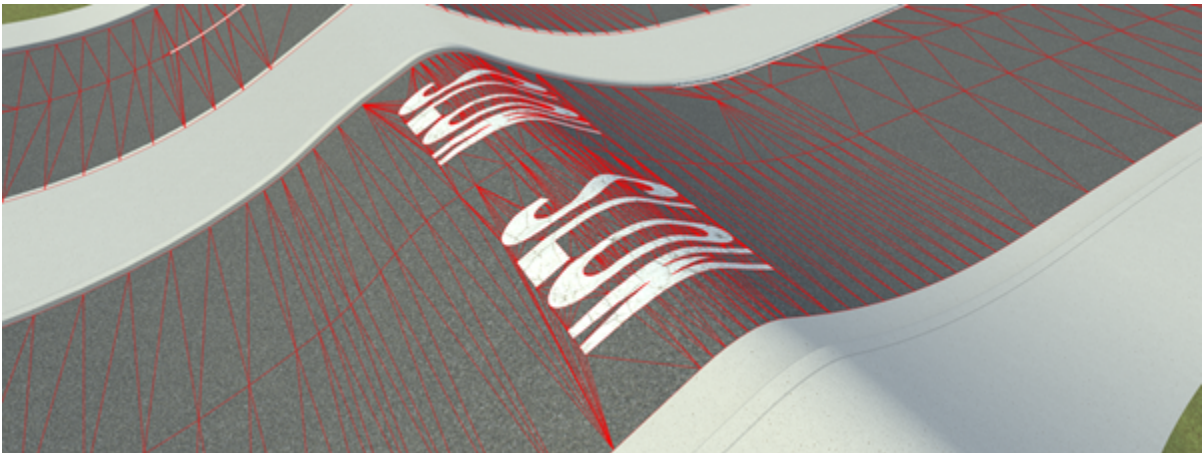
This image shows the sample markings produced when you set **Marking Construction Type** to `Floating`. The asphalt surface triangulation passes beneath the marking geometry.
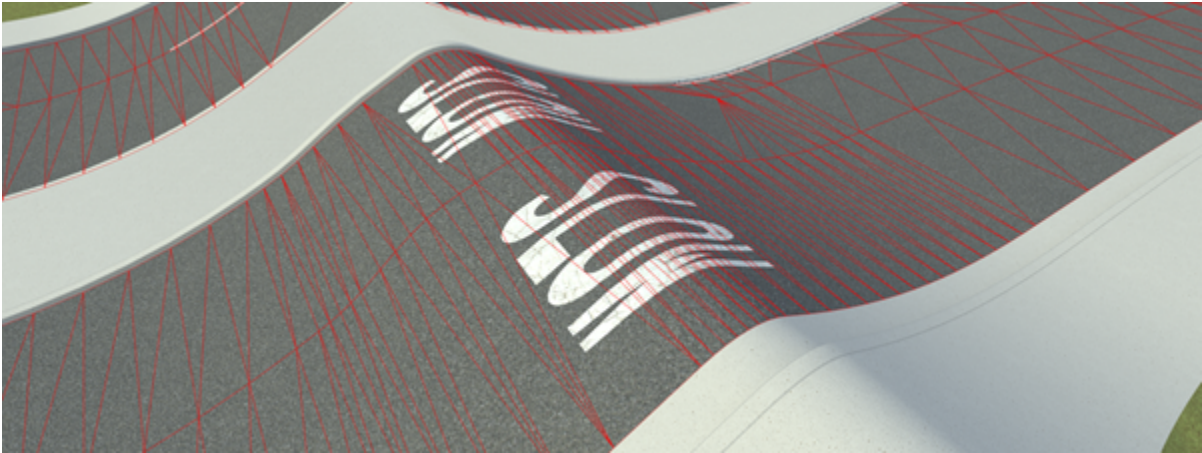
The **Floating Marking Offset** parameter controls how much to float a marking above the underlying surface. The lower the value, the smaller the gap between the markings and the surface. With lower values, the likelihood of rendering artifacts, such as *z*-fighting, increases.

Floating Marking Offset = 0.01	Floating Marking Offset = 0.5

Modify Scene Rendering

To further improve scene performance, you can select **Overhead Tile Rendering** to flatten the scene into a single texture image per exported tile. RoadRunner flattens roads, lanes, curbs, and terrain into this image but excludes props. Selecting this option can dramatically improve simulation performance. However, the reduction in scene quality is high, so reserve this optimization for high LOD distances only.

In the LOD Settings dialog box for the `FourWaySignal` scene, select `LOD 2`. This LOD has **Overhead Tile Rendering** enabled. You can select additional options to customize the tile rendering:

- **Texture Resolution** — Set the pixel resolution of the textures. Decrease this value to speed up simulation performance at the expense of texture image quality.

- **Use Simplified Mesh** — Simplify the mesh geometry of the texture image into a grid, which reduces the amount of triangles used to render the tile.

- **Simplified Grid Resolution** — Set the number of squares in the triangulation grid. Reduce this value to reduce the number of triangles used to construct the grid at the expense of texture image quality. To enable this option, you must select **Use Simplified Mesh**.

Select **Use Simplified Mesh** and leave **Simplified Grid Resolution** set to `10` to render the tiles using 10-by-10 mesh grids. Click **OK**.

Preview how the texture image looks by clicking the **Scene Export Preview Tool**.

1   Click the **Scene Export Preview Tool** button . The scene editing canvas displays the four tiles that are exported from the scene.
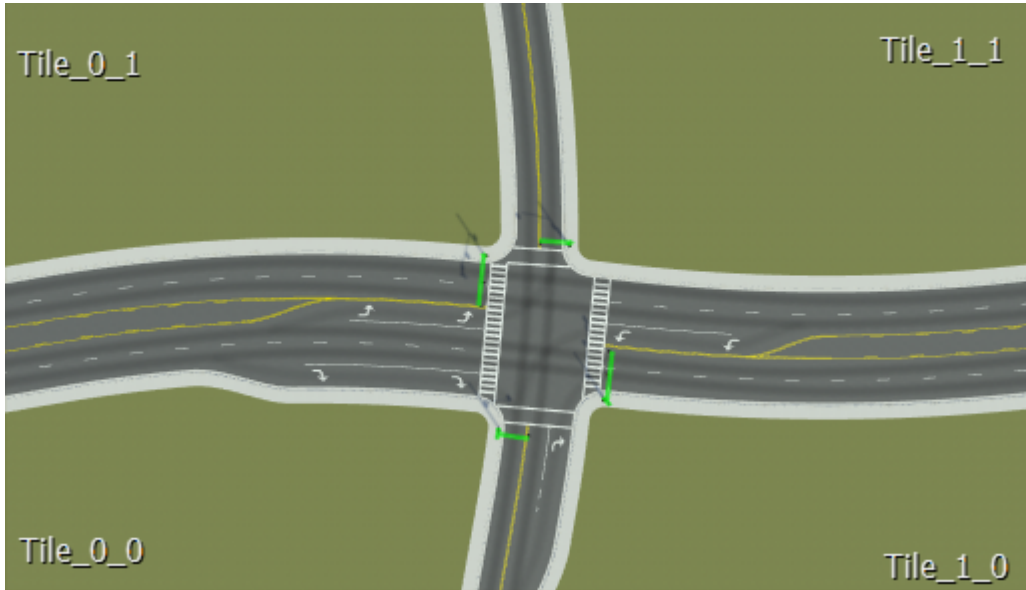
2   In the **Attributes** pane, select **Visualize Camera LOD Distance**.

3   Set the **Camera LOD Distance** to 3000, which is the distance at which LOD 2 starts.

4   Click **Refresh Scene**. RoadRunner combines scene objects into one texture image per tile. Only the traffic signal props at the intersection remain selectable. The scene quality is noticeably diminished but in simulators that use this scene, this image is used only when the scene is viewed from 3000 meters away.
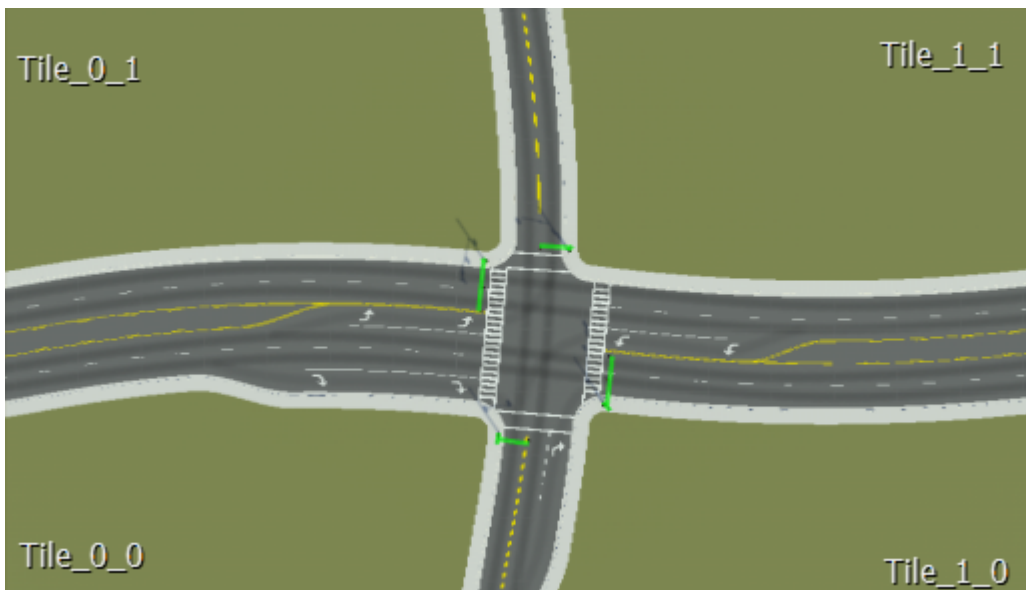
5   Select the black corner of one of the tiles to view the mesh grid of the tile. The tile is rendered using a simple 10-by-10 triangular mesh.
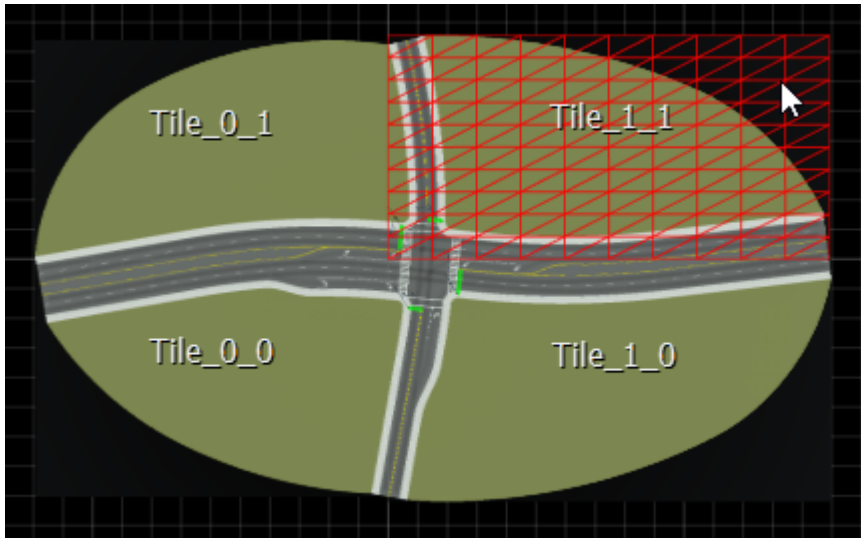
Pack Props

To improve simulation performance even more, you can pack all props into a single prop model per export tile. With packed props, all textures in a tile are combined into a maximum of two atlas textures: one for opaque materials and one for transparent materials. This optimization can greatly reduce the amount of state changes that the rendering engine of the target simulator uses. Prop packing reduces the amount of texture detail for props and lowers visual quality, so reserve this optimization for high LOD distances only.

View the effects of prop packing on the FourWaySignal scene.

1 Open the LOD Settings dialog box for the FourWaySignal scene. From the **Tools** menu, select **LOD Settings**.

2 In the left pane of the dialog box, select LOD 2.

3 Select **Pack Props**.

4 (Optional) Modify the **Atlas Resolution** parameter. This option changes the number of pixels used to render props. Decrease this value to improve simulation performance at the cost of visual quality.

5 Click **OK**.

6

Click the **Scene Export Preview Tool** button .

7 Focus the camera on the prop assembly in Tile_1_0. This assembly contains a "No U-Turn" sign and traffic lights. In the **Attributes** pane, check that **Visualize Camera LOD Distance** is selected and toggle the **Camera LOD Distance** attribute above and below the LOD 2 threshold of 3000 meters. When the camera distance is at LOD 2, prop packing is enabled and the textures of the prop assembly combine into one model. RoadRunner removes the sign face and simplifies the traffic light textures.

5-119

Prop Packing Disabled	Prop Packing Enabled
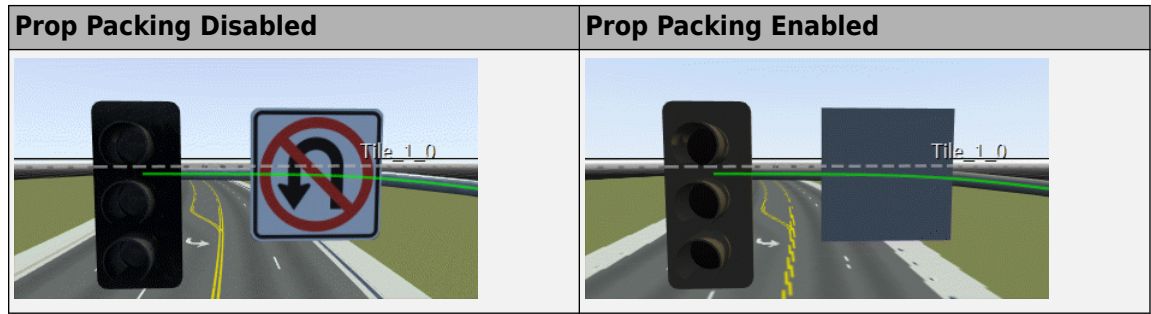
To view all the packed props of a tile, you can select them from the hierarchy of mesh nodes in each tile. In the **Attributes** pane of the **Scene Export Preview Tool**, under **Mesh Node Tree**, navigate to `Tile_1_0`. Then, select `PropLodGroup_1`.
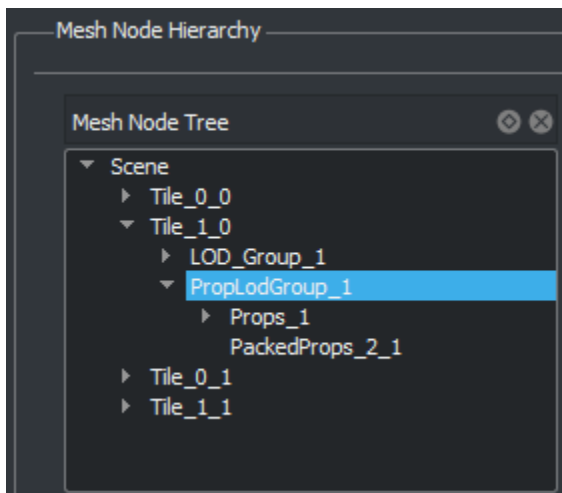
The `Props_1` group lists every individual prop on the tile, with each prop having its own mesh. Expand this group, select a prop, and view the corresponding mesh selected in the scene editing canvas. For example, this image shows a selected traffic light post prop.
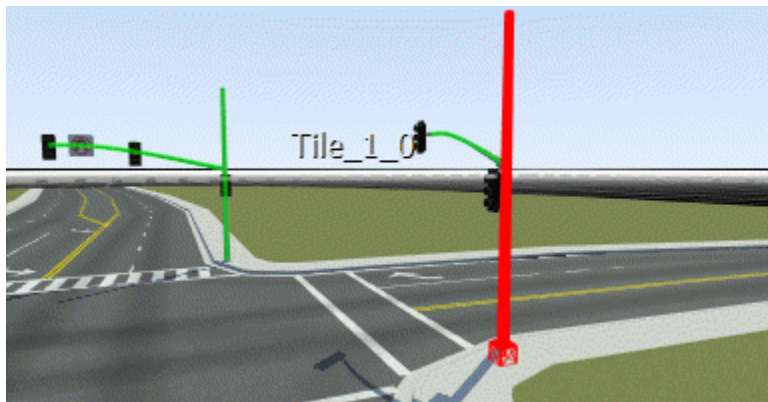
Select the `PackedProps_2_1` mesh node. In the scene editing canvas, all meshes are now collapsed into a single selectable mesh.
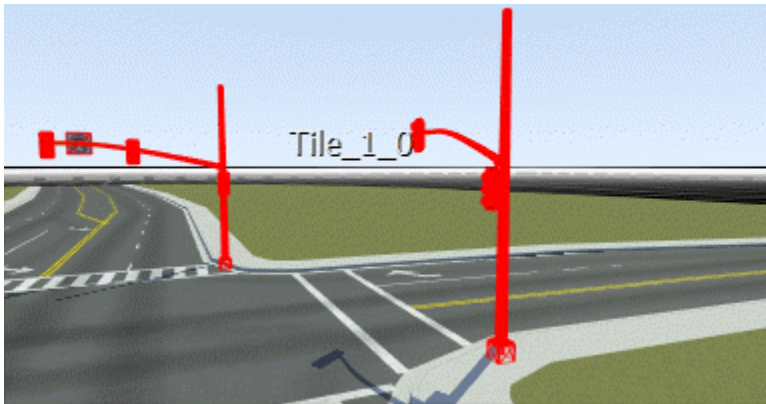
Visualize Performance Improvements

As an additional form of verification, you can visualize the number of draw calls that an external simulator needs to make to render the scene at varying LODs.

From the **View** menu, select **Sensor**, then **Draw Calls**. With this sensor selected, RoadRunner segments the scene by color, where each color represents a separate draw call.

In the FourWaySignal scene, with the **Scene Export Preview Tool** still enabled, set the **Camera LOD Distance** attribute below 3000 meters. At these close distances, the simulators make a high number of draw calls to render the roads, terrain, and props.
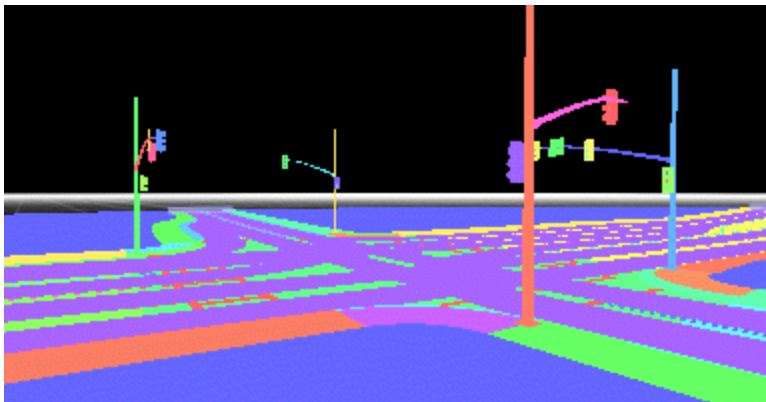
Set the **Camera LOD Distance** attribute above 3000 meters. At this distance, prop packing and tile rendering are enabled, so the simulators make significantly fewer draw calls to render the scene.
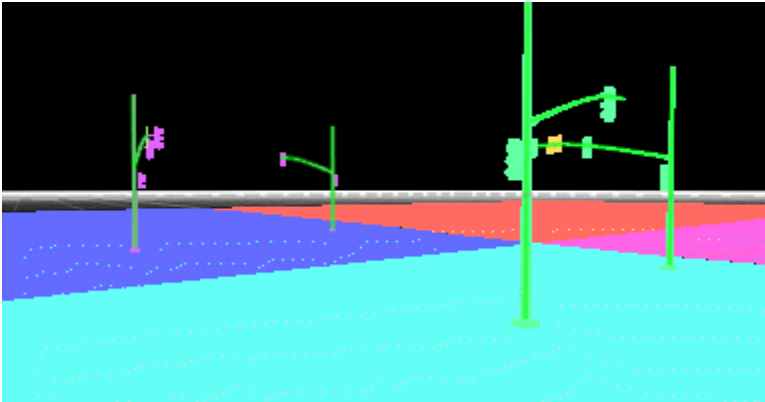
Export Scene

RoadRunner supports the exporting of multiple LODs to FBX files only. If you export to a format other than FBX, then RoadRunner uses the LOD settings specified by `LOD 0`.

Export the scene as an FBX file. From the **Export** menu, select **Filmbox (.fbx)**.

Open the folder to which you exported the scene. By default, scenes export to the `Exports` folder of the current RoadRunner project.

The exported FBX data includes the texture atlases used at the varying levels of detail. For more details on the exported FBX data, see "Export to FBX" on page 5-3.

See Also
Scene Export Preview Tool

Related Examples

Export Custom Formats

You can export RoadRunner scenes to a variety of file formats. These exporting options are available from the **File** menu, under **Export**. You can also configure this menu to include custom export options that combine the different file formats that RoadRunner supports. To specify a custom configuration, follow these steps:

1 Create an XML file that configures the details of the custom export format.

2 Save the XML file to the `Project` folder of the RoadRunner project that you want to contain the export option and name it `ExportConfigurations.xml`.

Create Export Configuration XML File

To enable custom exports, you must specify an export configuration XML file similar to the one shown here. This sample file specifies a custom option that exports an FBX file, ASAM OpenDRIVE file, GeoJSON file, and RoadRunner scene metadata file together in one zip file.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<ExportConfigurations>
  <Configuration name="My Custom Format" zip="true" extension="zip">
    <Format name="Filmbox" extension="fbx">
      <Option name="ExportToTiles" value="true" type="bool"/>
      <Option name="ExportIndividualTiles" value="true" type="bool"/>
      <Option name="TileSize" value="1000,1000" type="vec2"/>
      <Option name="Center" value="0,0" type="vec2"/>
      <Option name="SpecularMapAsRoughnessMap" value="true" type="bool"/>
      <Option name="Offset" value="0,0,-72.226989" type="vec3"/>
    </Format>
    <Format name="OpenDRIVE" extension="xodr">
      <Option name="ExportIndividualRoadMarkings" value="false" type="bool"/>
      <Option name="ExportSignals" value="true" type="bool"/>
      <Option name="ExportObjects" value="true" type="bool"/>
      <Option name="ExportConflictPoints" value="false" type="bool"/>
      <Option name="ExportSceneReference" value="false" type="bool"/>
      <Option name="QuantizeRoads" value="true" type="bool"/>
      <Option name="VerticalOffset" value="-72.226989" type="double"/>
    </Format>
    <Format name="GeoJSON" extension="geojson"/>
    <Format name="Scene Metadata" extension="xml"/>
  </Configuration>
</ExportConfigurations>
```

<ExportConfigurations> Element

The top-level element of the XML file, `<ExportConfigurations>`, is required. This element contains all the custom export configurations that appear in the **File** menu, under **Export**.

<Configuration> Elements

The `<ExportConfigurations>` element can contain one or more `<Configuration>` elements. Each `<Configuration>` element corresponds to a custom export format that appears in the **File** menu, under **Export**. This table describes the attributes that you can specify for this element.

<Configuration> Attribute	Optional or Required	Description
name	Required	Name of the exported file, specified as a string.
extension	Required	Extension of the exported file, specified as a string.
zip	Optional	Option to export the custom format as a zipped file, specified as `false` or `true`. If you specify `zip` as `true`, then specify `extension` as a zip file extension, such as `zip` or `rar`. **Default**: `false`

<Format> Elements

Each `<Configuration>` element can contain one or more `<Format>` elements. These elements correspond to the file formats that RoadRunner exports as part of the custom configuration. This element requires a `name` attribute, which specifies the name of a file format that RoadRunner supports. This element also includes an optional `extension` attribute, which specifies the extension of the exported file. If you do not specify this attribute, then the exported file has the default extension for that file format.

This table lists the format names that you can specify for this attribute and their corresponding valid extensions. These attributes are case sensitive. If you specify an invalid attribute, then the export dialog box does not include a tab for that format.

name Attribute	Valid extension Attributes	Description	Export Format Details
AutoCAD	dxf (default)	Export scene to an AutoCAD file.	"Export to AutoCAD" on page 5-2
Filmbox	fbx (default)	Export scene to a Filmbox file.	"Export to FBX" on page 5-3
glTF	gltf (default)	Export scene to a GL Transmission Format (glTF) file.	"Export to glTF" on page 5-5
OpenFlight	flt (default)	Export scene to an OpenFlight file.	"Export to OpenFlight" on page 5-6
OpenSceneGraph	osg (default)	Export scene to an OpenSceneGraph file.	"Export to OpenSceneGraph" on page 5-7
USD	usd (default)	Export scene to a Universal Scene Description (USD) file.	"Export to USD" on page 5-15
Wavefront	obj (default)	Export scene to a Wavefront OBJ file.	"Export to Wavefront OBJ" on page 5-8

name Attribute	Valid extension Attributes	Description	Export Format Details
OpenDRIVE	xodr (default)	Export scene to an ASAM OpenDRIVE file.	"Export to ASAM OpenDRIVE" on page 5-24
Apollo	xml (default)	Export scene to the Baidu Apollo file format.	"Export to Apollo" on page 5-54
GeoJSON	geojson (default)	Export scene to a GeoJSON file.	"Export to GeoJSON" on page 5-9
Scene Metadata	xml (default)	Export scene metadata to an XML file.	"RoadRunner Metadata Export" on page 5-52

<Option> Elements

Each <Format> element can optionally include <Option> elements. These elements specify options for configuring your export formats. If you do not specify an option, then RoadRunner assigns the default value to that option. Each <Option> element that you do specify must include these attributes:

• name — Name of option
• value — Value of option
• type — Data type of option

This table describes the option attributes that you can specify for mesh export formats, which apply to these formats:

• AutoCAD
• Filmbox
• glTF
• OpenSceneGraph
• USD
• Wavefront

name Attribute	Description	value Attribute	type Attribute
SplitBySegmentation	Split meshes by their segmentation type. For more details, see "Segmentation" on page 5-49.	false (default) | true	bool
PowerOfTwoTextures	Resize the dimensions of exported textures by rounding them up to the next highest power of two.	false (default) | true	bool
ExportToTiles	Split meshes per tile.	false (default) | true	bool

name Attribute	Description	value Attribute	type Attribute
TileSize	Specify the size of exported tiles, when `ExportToTiles` is `true`.	`0,0` (default) | two-element real-valued vector	vec2
ExportIndividualTiles	Export tiles to separate files.	`false` (default) | `true`	bool
EmbedTextures	Embed the exported textures inside the exported file.	`false` (default) | `true`	bool
LodAsPercentage	Specify levels of detail by percentage.	`false` (default) | `true`	bool
CollapseMesh	Export the scene as a single mesh rather than as multiple nodes.	`false` (default) | `true`	bool
SplitByPass	Split meshes by triangulation pass, for example, separating markings or damage from the road surface.	`false` (default) | `true`	bool
SplitByMaterial	Split meshes by material.	`false` (default) | `true`	bool
SpecularMapAsRoughnessMap	Use the specular map as the roughness map.	`false` (default) | `true`	bool
UnityLods	Export levels of detail for use with Unity.	`false` (default) | `true`	bool
Offset	Specify additional (*x*,*y*,*z*) offset for exported scene.	`0,0,0` (default) | three-element real-valued vector	vec3

This table describes the option attributes that you can specify for mesh export formats, which apply to these formats:

- `OpenDRIVE`
- `Apollo`

name Attribute	Description	value Attribute	type Attribute
QuantizeRoads	Perform quantization to prevent very short roads or lanes.	`true` (default) | `false`	bool
DatabaseVersion	Specify an identifier for the exported scene, which is useful for versioning exports of the same scene.	`1.0` | positive real scalar	double

name Attribute	Description	value Attribute	type Attribute
DatabaseName	Specify the name of the exported scene.	"" | string	string
ExportIndividualRoadMarkings	Export road markings using the `<line>` definition in ASAM OpenDRIVE.	false (default) | true	bool
ExportSignals	Export all signals and signs mapped to junctions as `<signal>` entries.	true (default) | false	bool
ExportObjects	Export all props as `<object>` entries.	true (default) | false	bool
ExportConflictPoints	Export conflict points, that is, point in junctions at which connecting roads overlap.	false (default) | true	bool
ExportSceneReference	Include `<object>` element at (0,0) in exported file to use as a reference point.	false (default) | true	bool
ReduceFileSize	Remove new lines from exported file to reduce file size.	false (default) | true	bool
VerticalOffset	Specify vertical offset for exported scene.	0 | nonnegative real scalar	double

For the `GeoJSON` format, `ReduceFileSize` is the only valid option.

The `Scene Metadata` format does not have any options.

Save Export Configuration File to Project

Save the XML file containing your export configurations to the `Project` folder of your project and give it the name `ExportConfigurations.xml`. If you do not save the file to this folder and with this name, then RoadRunner does not recognize the custom export formats.

After saving the file, restart RoadRunner and open a scene in the project that has the `ExportConfigurations.xml` file. In the **File** menu, under **Export**, the custom format options are now included at the bottom of the list. This figure shows a sample custom format option.
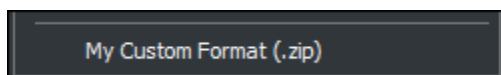
See Also

Related Examples

- "Export Scenes"
- "RoadRunner Project and Scene System" on page 2-2

Export to STL

RoadRunner can export scenes to the STL file format.

STL Export

From the **File** menu, select **Export**, then **STL (.stl)** to open the Export STL dialog box. Then, specify a path to which to export the file, and click **Export**.
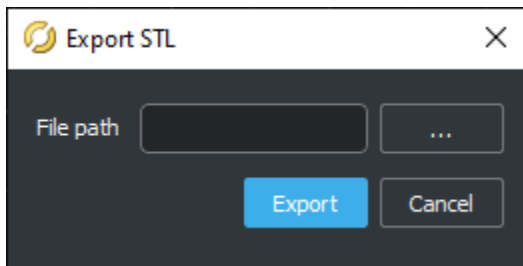
Advanced Details

STL files support both ASCII and binary file formats. The ASCII version of an STL file is larger than the equivalent binary STL file. RoadRunner exports the scenes to only the binary STL file format, but supports importing from both binary and ASCII-formatted STL files. For more information on the STL file formats, see the File Format page about STL files.

Limitations

- An STL file does not contain any color or texture information. Therefore, the exported geometries of some assets (mainly trees) may appear wrapped.

- The units in STL files are arbitrary and there is no scale information. RoadRunner uses meters as the unit of measurement for the exported STL files.

Programmatic Scene Interfaces

Control RoadRunner Programmatically Using gRPC API

RoadRunner provides an API that enables you to control the RoadRunner UI programmatically. For example, using this API, you can:

- Create, load, and save RoadRunner scenes, scenarios, and projects.
- Import ASAM OpenDRIVE files into scenes.
- Export scenes and scenarios to one of the file formats that RoadRunner supports.

RoadRunner enables you to compile versions of the API in various programming languages and call them in the language you choose. Alternatively, you can use a precompiled version of the API that enables you to control RoadRunner from the command line.

How The RoadRunner API Works

The RoadRunner API is built using the open-source gRPC framework. This framework uses a client-server architecture in which a client application remotely controls a server application using a set of remote procedure call (RPC) methods. In RoadRunner:

- Your locally installed version of RoadRunner is the server application.
- The RoadRunner API provides the RPC methods used to remotely control RoadRunner.
- The programs you write to call the RPC methods are the client applications. The gRPC framework is language-neutral and platform-neutral. You can write clients that call the RoadRunner API in any platform and language that gRPC supports. For details on what languages and platforms gRPC supports, see the gRPC Documentation.

This diagram shows a simplified layout of the API architecture. In this diagram, a Python client uses the `LoadScene` method to load a scene in RoadRunner.
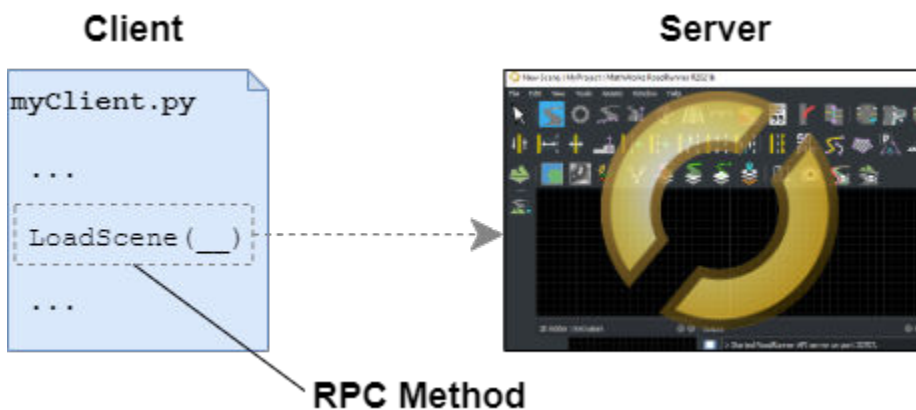
How The RoadRunner API Sends and Receives Data

The RPC methods of the RoadRunner API are defined in a gRPC *service*. Each time you call a method that is part of a gRPC service, that method:

- Sends a request to a server.
- Receives a response back from the server.

This diagram shows this request-response format for a call of the `LoadScene` method. In the client, the input to the method, `LoadSceneRequest`, is a request that the client sends the RoadRunner application server. RoadRunner processes this request, loads a scene, and sends back a response, `LoadSceneResponse`.
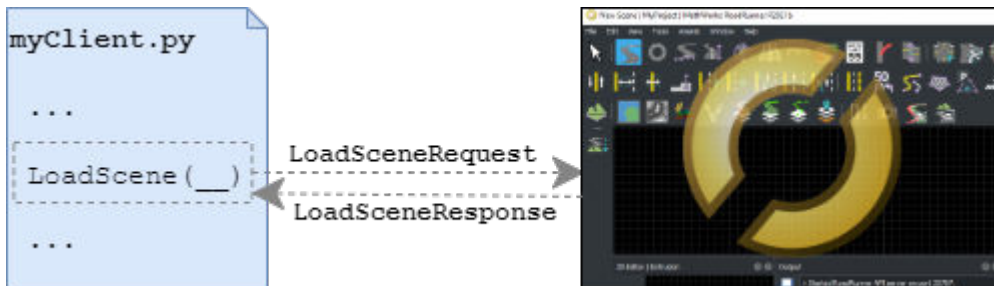
The data in these requests and responses is structured as messages that are defined using the *protocol buffer* (protobuf) schema. The protobuf schema is a language-neutral format developed by Google, and is optimized for fast and efficient data transfer. The RoadRunner server can send and receive millions of protobuf messages from these API calls simultaneously while maintaining real-time updates of RoadRunner.

Messages in the protobuf schema are defined in text files with a `.proto` extension. These messages contain name-value fields that define:

- The names of the fields that you can specify in the messages.
- The data types of the fields. For example, you can specify fields as Boolean values, strings, or as other protobuf messages.

Consider the schema for the `LoadScene` RPC method, as defined in the `roadrunner_service.proto` file.

```
// Load scene
rpc LoadScene (LoadSceneRequest) returns (LoadSceneResponse) {}
```

The schema for the request message, `LoadSceneRequest`, and response message, `LoadSceneResponse`, are defined in the `roadrunner_service_messages.proto` file. `LoadSceneRequest` takes in one required input, `file_path`, which is a string that specifies the path of the file to load.

```
message LoadSceneRequest
{
  // Scene file to load (required)
  string file_path = 1;
}
```

After RoadRunner processes the request (tries to load the scene), the RoadRunner API server sends back an empty `LoadSceneResponse` message in response.

```
message LoadSceneResponse
{
}
```

Since the protobuf schema is language-neutral, the syntax used to call the methods and format your message requests depends on the programming language you use to write your client applications.

Connect to RoadRunner API Server

To use the RoadRunner API, you must first establish a network connection with the RoadRunner API server. This server is a part of your local RoadRunner installation and starts running when you open a project.

To programmatically open RoadRunner and start the API server, call the `AppRoadRunner` executable from your local RoadRunner installation. This executable contains command-line options that enable you to specify:

- The project that RoadRunner opens to
- The IP network port that the RoadRunner API server runs on

This command-line code shows how to open RoadRunner from its default installation location on Windows. RoadRunner opens to a project located at `C:\RR\MyProject` on IP network port `54321`.

```
cd "C:\Program Files\RoadRunner R2022b\bin\win64"
AppRoadRunner --projectPath C:\RR\MyProject --apiPort 54321
```

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

> Started RoadRunner API server on port 54321.

Use RoadRunner API from Command Line

RoadRunner provides a precompiled helper command, `CmdRoadRunnerApi`, that enables you to call RoadRunner RPC methods from the command line. This helper command is located in the same folder as the `AppRoadRunner` executable.

This code calls the `LoadScene` method to load the prebuilt `FourWaySignal` scene from the open project.

```
CmdRoadRunnerApi "LoadScene(file_path='FourWaySignal')" --serverAddress=localhost:54321
```
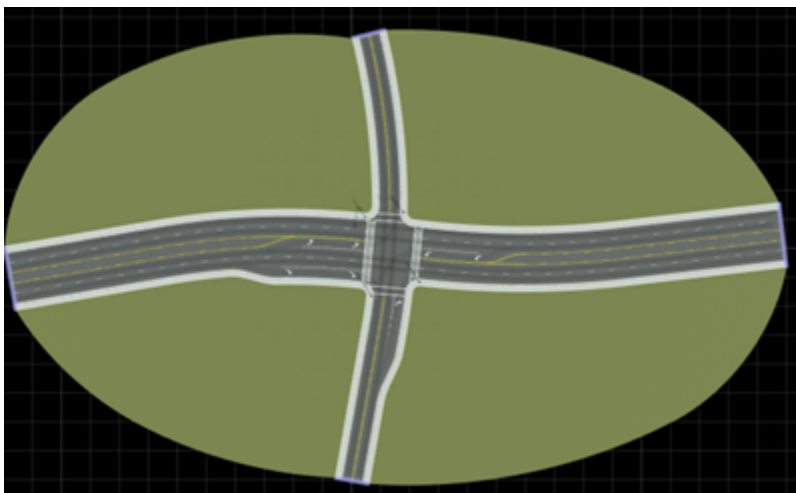
Use RoadRunner API in Various Programming Languages

For additional flexibility in using the RoadRunner API, you can compile the API into a language supported by gRPC, and then write client applications in that language to control RoadRunner programmatically.

Compile RoadRunner API

To compile the RoadRunner API in your desired programming language, you must first copy the protobuf files that define the API into a writable folder. These files are located in your local RoadRunner installation. You can then use the protobuf compiler, along with the gRPC plugin for your desired programming language, to compile language-specific versions, or bindings, of the RoadRunner API. For example, this diagram shows the generation of Python and C++ bindings.
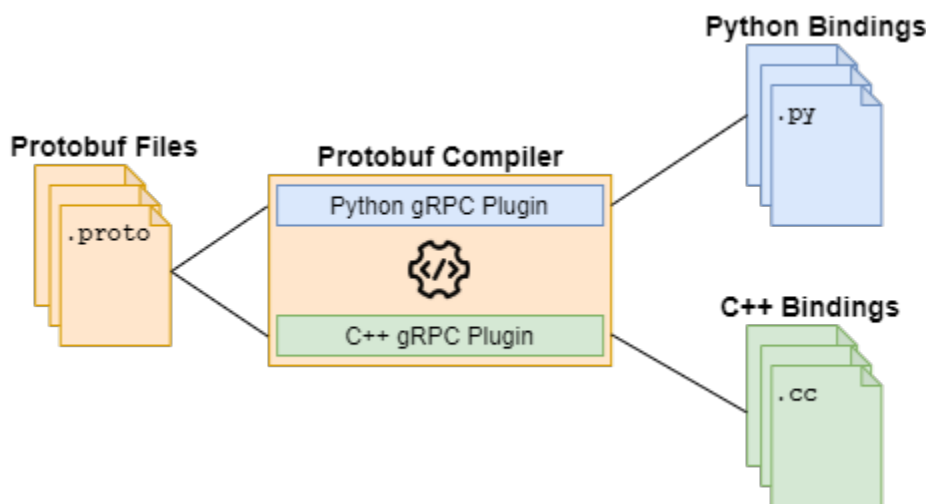
For details on compiling the protobuf files, see "Compile Protocol Buffers for RoadRunner gRPC API" on page 6-19.

Create RoadRunner API Clients

The clients that you write to programmatically control RoadRunner typically contain code that performs these steps:

1   Import gRPC code from the compiled bindings.
2   Establish a local network connection to the RoadRunner API server.
3   Use the imported gRPC code to create a RoadRunner API object. This object is called a stub.
4   Call the RPC methods from this stub to control RoadRunner over the local network.

This simple Python client shows an example of calling the `LoadScene` method.

```
import grpc
from mathworks.roadrunner import roadrunner_service_messages_pb2
from mathworks.roadrunner import roadrunner_service_pb2_grpc

with grpc.insecure_channel("localhost:54321") as channel:
    api = roadrunner_service_pb2_grpc.RoadRunnerServiceStub(channel)
    loadSceneRequest = roadrunner_service_messages_pb2.LoadSceneRequest()
    loadSceneRequest.file_path = "FourWaySignal"
    api.LoadScene(loadSceneRequest)
```

After importing the compiled gRPC Python bindings, this client establishes a connection over a gRPC channel and creates a stub for the RoadRunner service API. The gRPC channel for the API uses insecure channel credentials and has no encryption or authentication. Since RoadRunner typically runs on a local machine and is not connecting to external networks, the security risk is low. Then, the client calls `LoadScene` from the API stub, which loads the prebuilt `FourWaySignal` scene from the currently open project.

You can have multiple clients calling RPC methods simultaneously, as long they are connected to RoadRunner on the same network port. For example, in this diagram, both a Python client and C++ client are calling `LoadScene` over network port 54321.
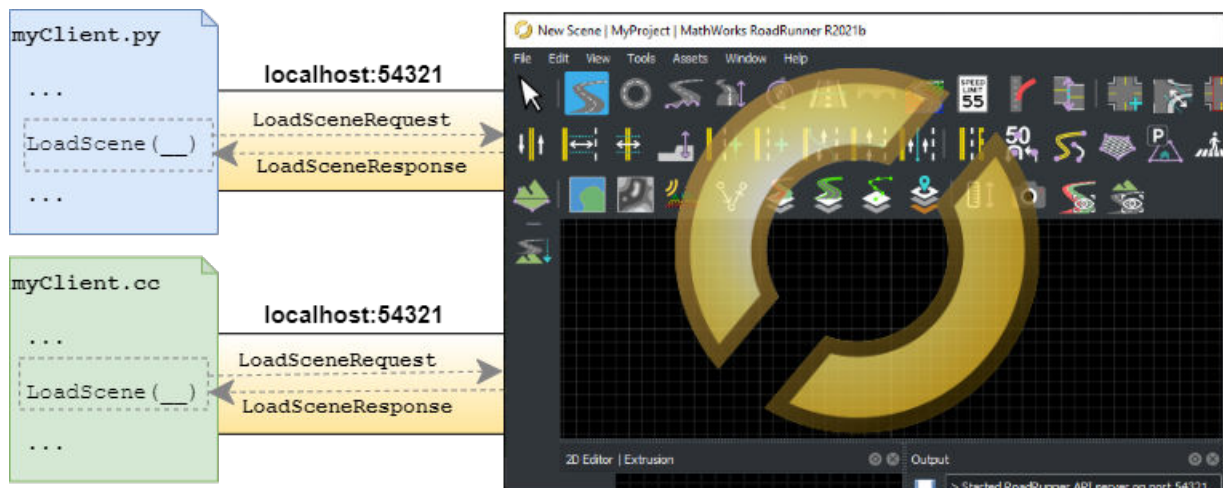
See Also
AppRoadRunner

Related Examples

- "Convert Scenes Between Formats Using gRPC API" on page 6-8
- "Export Multiple Scenes Using gRPC API" on page 6-14
- "Generate Scenario Variations Using gRPC API" (RoadRunner Scenario)
- "Export Multiple Scenarios Using gRPC API" (RoadRunner Scenario)
- "Reuse Scenarios in Multiple Scenes Using gRPC API" (RoadRunner Scenario)

More About

- "Compile Protocol Buffers for RoadRunner gRPC API" on page 6-19
- "Create gRPC Python Client for Controlling RoadRunner Programmatically" on page 6-23
- "Create gRPC C++ Client for Controlling RoadRunner Programmatically" on page 6-28

Convert Scenes Between Formats Using gRPC API

This example shows how to import RoadRunner scenes from one file format and export those scenes to a different format. In this example, you import ASAM OpenDRIVE files into scenes, save them to a project, and export the scenes to Filmbox FBX files 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" on page 6-2.

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" on page 2-2.
- (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.

Import and Export Single Scene

Preview the programmatic operations by importing one ASAM OpenDRIVE file into a RoadRunner scene and exporting the scene to an FBX file. To perform these operations, use the `CmdRoadRunnerApi` helper command. This helper command is located in the same folder as the `AppRoadRunner` executable file.

Import ASAM OpenDRIVE File

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

```
CmdRoadRunnerApi "Import(file_path='C:\ODR\MyOpenDRIVEFile.xodr')" --serverAddress localhost:54321
```

If you do not have an ASAM OpenDRIVE file, you can download a sample file from the ASAM OpenDRIVE website.

This example shows an ASAM OpenDRIVE file generated by loading one of the prebuilt scenes from the current project, exporting it to ASAM OpenDRIVE, and then importing the exported file into a new scene. The code to generate this file is shown here:

```
CmdRoadRunnerApi "LoadScene(file_path='FourWaySignal')" --serverAddress localhost:54321
CmdRoadRunnerApi "Export(file_path='FourWaySignal.xodr' format_name='opendrive')" --serverAddress localhost:54321
CmdRoadRunnerApi "NewScene()" --serverAddress localhost:54321
CmdRoadRunnerApi "Import(file_path='C:\RR\MyProject\Exports\FourWaySignal.xodr')" --serverAddress localhost:54321
```
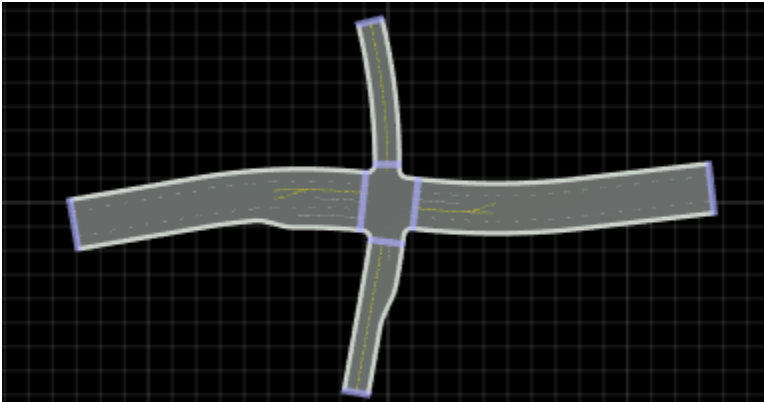
Export to FBX

Export the scene created from the imported ASAM OpenDRIVE file to the Filmbox FBX file format, and specify the option to split meshes by their segmentation type.

- For the exported file name, specify the same name as the current scene, but with the extension `.fbx`. By default, RoadRunner exports the scene to the `Exports` folder of the open project.
- For the `serverAddress` option, replace the example port number with the `apiPort` value you specified when opening RoadRunner.

```
CmdRoadRunnerApi "Export(file_path='FourWaySignal.fbx' format_name='filmbox' ^
filmbox_settings.split_by_segmentation.value='true')" --serverAddress localhost:54321
```

Navigate to the `Exports` folder. List the folder contents and verify that the folder contains the `.fbx` file and associated texture image files.

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

```
Asphalt1_Diff.png
Asphalt1_Norm.png
Asphalt1_Spec.png
Concrete1_Diff.png
Concrete1_Norm.png
Concrete1_Spec.png
FourWaySignal.fbx
...
```

Import and Export Multiple Scenes

To import all ASAM OpenDRIVE files in a folder and export them to the FBX format, you can write a script that calls the `CmdRoadRunnerApi` command in a loop.

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

- `RRPATH` — Update to the `bin/platform` folder path of your local RoadRunner installation.
- `PROJECT` — Update to the path of your RoadRunner project.

- PORT — Update to the IP network port that you want to connect to.
- OPENDRIVEFOLDER — Update to the folder path that contains the ASAM OpenDRIVE files you want to import.

---

bulk_opendrive_import_fbx_export.bat (Windows)

```
@echo off
SetLocal EnableDelayedExpansion
REM Bulk-import ASAM OpenDRIVE files into RoadRunner scenes and export scenes to FBX.

set RRPATH=C:\Program Files\RoadRunner R2022b\bin\win64&
set PROJECT=C:\RR\MyProject&
set PORT=54321&
set OPENDRIVEFOLDER="C:\RR\ODR"&

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 OpenDRIVE files into new scene. Export FBX files to subfolders under "Exports" folder of project.
for %%F in (%OPENDRIVEFOLDER%\*.xodr) do (
  set FILENAME=%%~nF
  set SCENENAME=!FILENAME!_OpenDRIVE
  set EXPORTPATH=!PROJECT!\Exports\!FILENAME!\!FILENAME!.fbx
  CmdRoadRunnerApi "NewScene()" --serverAddress=localhost:!PORT!
  CmdRoadRunnerApi "Import(file_path='%%F')" --serverAddress localhost:!PORT!
  CmdRoadRunnerApi "Export(file_path='!EXPORTPATH!' format_name='filmbox')" --serverAddress localhost:!PORT!
  CmdRoadRunnerApi "SaveScene(file_path='!SCENENAME!')" --serverAddress=localhost:!PORT!
)

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

---

bulk_opendrive_import_fbx_export.bash (Linux)

```
#!/bin/bash
# Bulk-import ASAM OpenDRIVE files into RoadRunner scenes and export scenes to FBX.

RRPATH="/usr/local/RoadRunner_R2022b/bin/glnxa64"
PROJECT="/local/RR/MyProject"
PORT=54321
OPENDRIVEFOLDER="/local/RR/ODR"

# Open RoadRunner
cd "$RRPATH"
./AppRoadRunner --projectPath="$PROJECT" --apiPort="$PORT" &

# Load OpenDRIVE files into new scene. Export FBX files to subfolders under "Exports" folder of project.
for OPENDRIVEFILE in $OPENDRIVEFOLDER/*.xodr
do
  FILENAME=$(basename $OPENDRIVEFILE .xodr)
  SCENENAME=$FILENAME"_OpenDRIVE"
  EXPORTPATH=$PROJECT"/Exports/"$FILENAME"/"$FILENAME".fbx"
  ./CmdRoadRunnerApi "NewScene()" --serverAddress=localhost:$PORT
  ./CmdRoadRunnerApi "Import(file_path='$OPENDRIVEFILE')" --serverAddress localhost:$PORT
  ./CmdRoadRunnerApi "Export(file_path='$EXPORTPATH' format_name='filmbox')" --serverAddress localhost:$PORT
  ./CmdRoadRunnerApi "SaveScene(file_path='$SCENENAME')" --serverAddress=localhost:$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 ASAM OpenDRIVE file contained in the folder into a new scene.

3 Exports each scene to the Filmbox FBX format using the default settings. For each exported scene, RoadRunner exports the FBX file and associated texture image files to subfolders under the `Exports` folder of the project.

4 Saves the scene to the `Scenes` folder. Each scene has the same name as the ASAM OpenDRIVE file but with the suffix `_OpenDRIVE` and the extension `.rrscene` instead of `.xodr`.

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.

```
bulk_opendrive_import_fbx_export
```

To verify that the script has exported the scenes, navigate to the `Exports` folder of the specified project and list the folder contents. For example, if you specified a project at path `"C:\RR\MyProject"`, run these commands.

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

If your project included only the default scenes, then the `Exports` folder contains a subfolder for each scene.

```
FourWaySignal
FourWayStop
SanAntonio
T_Intersection
```

Open one of the subfolders and verify that the folder contains the exported Filmbox file and associated texture image files.

```
cd FourWaySignal
dir
```

```
Asphalt1_Diff.png
Asphalt1_Norm.png
Asphalt1_Spec.png
Concrete1_Diff.png
Concrete1_Norm.png
Concrete1_Spec.png
FourWaySignal.fbx
...
```

Navigate to the `Scenes` folder of the project and list the contents to verify that RoadRunner saved ASAM OpenDRIVE versions of the scenes.

```
cd ..\..\Scenes
dir
```

```
FourWaySignal.rrscene
FourWaySignal_OpenDRIVE.rrscene
FourWayStop.rrscene
FourWayStop_OpenDRIVE.rrscene
...
```

Extend RoadRunner Import and Export Options

To customize the script further, you can specify nondefault import and export settings or specify other file formats. For more details on supported formats, see the `Import` and `Export` RPC methods.

For additional flexibility in importing and exporting scenes, 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" on page 6-19. You can then write client applications in those languages. For sample clients, see these examples:

- "Create gRPC Python Client for Controlling RoadRunner Programmatically" on page 6-23
- "Create gRPC C++ Client for Controlling RoadRunner Programmatically" on page 6-28

See Also

AppRoadRunner | Import | Export | import_settings.proto | export_settings.proto

Related Examples

- "Control RoadRunner Programmatically Using gRPC API" on page 6-2
- "Export Multiple Scenes Using gRPC API" on page 6-14

See Also

Export Multiple Scenes Using gRPC API

This example shows how to bulk-export scenes in a RoadRunner project to one of the file formats supported by RoadRunner. In this example, you export scenes to the ASAM OpenDRIVE 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" on page 6-2.

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" on page 2-2.

- (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.

Export Single Scene

Preview the programmatic operations by exporting one scene from the current project to ASAM OpenDRIVE. To perform these operations, you use the `CmdRoadRunnerApi` helper command. This helper command is located in the same folder as the `AppRoadRunner` executable file.

Load the `FourWaySignal` scene by using the `LoadScene` RPC method. This scene is included by default in the `Scenes` 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 this option.

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


Export the scene to the ASAM OpenDRIVE version 1.6 format by using the `Export` RPC method.

- For the exported file name, specify the same name as the scene, but with the extension `.xodr`. By default, RoadRunner exports the scene to the `Exports` folder of the opened project.
- In the `serverAddress` option, replace the port number with the `apiPort` value you specified when opening RoadRunner.

```
CmdRoadRunnerApi "Export(file_path='FourWaySignal.xodr' format_name='opendrive' ^
open_drive_settings.open_drive_version='1.6')" --serverAddress=localhost:54321
```

Navigate to the `Exports` folder and open the exported ASAM OpenDRIVE file. Verify that the `<header>` tag contains the specified file version.

```
cd "C:\RR\MyProject\Exports"
FourWaySignal.xodr
```

```
<OpenDRIVE>
    <header revMajor="1" revMinor="6" ...>
    ...
```

Export Multiple Scenes

To export 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 `bulk_opendrive_export.bat` (Windows) or `bulk_opendrive_export.bash` (Linux) and modify these variables:

- `RRPATH` — Update to the `bin/platform` folder path of your local RoadRunner installation.
- `PROJECT` — Update to the path of your RoadRunner project.
- `PORT` — Update to the IP network port that you want to connect to.

bulk_opendrive_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 scenes from "Scenes" folder and export to "Exports" folder.
for %%F in (%PROJECT%\Scenes\*.rrscene) do (
  set SCENENAME=%%~nF
  set EXPORTPATH=!PROJECT!\Exports\!SCENENAME!.xodr
  CmdRoadRunnerApi "LoadScene(file_path='%%F')" --serverAddress=localhost:!PORT!
  CmdRoadRunnerApi "Export(file_path='!EXPORTPATH!' format_name='OpenDRIVE')" --serverAddress=localhost:!PORT!
)

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

```
bulk_opendrive_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" &

# Load scenes from "Scenes" folder and export to "Exports" folder.
for SCENE in $PROJECT/Scenes/*.rrscene
do
  SCENENAME=$(basename $SCENE .rrscene)
  EXPORTPATH=$PROJECT/Exports/$SCENENAME.xodr
  ./CmdRoadRunnerApi "LoadScene(file_path='$SCENE')" --serverAddress=localhost:$PORT
  ./CmdRoadRunnerApi "Export(file_path='$EXPORTPATH' format_name='OpenDRIVE')" --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 Exports each scene to ASAM OpenDRIVE using the default settings. RoadRunner exports the file to the `Exports` folder of the project and gives it the same name as the scene, but with the extension `.xodr`.

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_opendrive_export
```

To verify that the script has exported the scenes, 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
```

If your project included only the default scenes, then your folder contains only the ASAM OpenDRIVE files and corresponding GeoJSON files.

```
FourWaySignal.xodr
FourWaySignal.geojson
FourWayStop.xodr
FourWayStop.geojson
...
```

Extend RoadRunner Export Options

To customize the script further, you can specify nondefault export settings or specify other file formats. For more details on supported formats, see the `Export` RPC method.

For additional flexibility in exporting scenes, 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" on page 6-19. You can then write client applications in those languages. For sample clients, see these examples:

- "Create gRPC Python Client for Controlling RoadRunner Programmatically" on page 6-23
- "Create gRPC C++ Client for Controlling RoadRunner Programmatically" on page 6-28

See Also
AppRoadRunner | Import | Export | import_settings.proto | export_settings.proto

Related Examples
- "Control RoadRunner Programmatically Using gRPC API" on page 6-2
- "Convert Scenes Between Formats Using gRPC API" on page 6-8

Compile Protocol Buffers for RoadRunner gRPC API

The RoadRunner API is built using an open-source, language-neutral framework called gRPC. With this framework, you can compile the protocol buffer (protobuf) schema files that define the API into one of several programming languages supported by gRPC and then use the compiled API in that language.

To compile the protobuf files, follow these steps:

1   Install the gRPC library and protobuf compiler, as well as any associated plugin for your desired programming language.

2   Copy the API protobuf files from your RoadRunner installation to a local, writable folder.

3   Use the compiler and gRPC plugin to generate the language-specific versions, or bindings, of the API, which you can call from the clients you write.

This diagram shows bindings generated for Python and C++, but you can generate bindings for any language that gRPC supports.
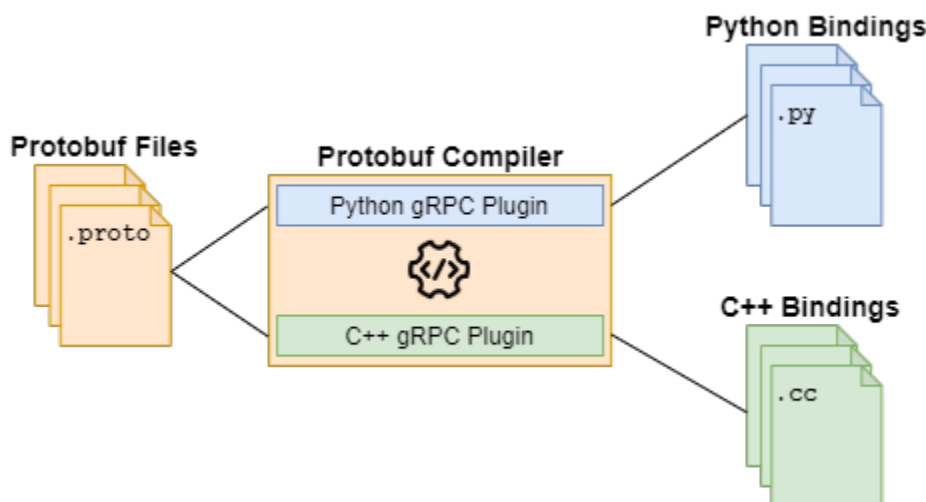
Verify Minimum Software Requirements

Before compiling, verify in the gRPC Documentation that gRPC supports:

•   The programming language to which you want to compile

•   The minimum version you are using for your programming language

•   The OS you are using

Install gRPC and Protobuf Compiler

To compile the RoadRunner API, install this software:

•   The gRPC runtime library for your language (minimum version 1.23.1). Follow the instructions on `https://github.com/grpc/grpc/`.

- The protocol compiler, `protoc` (minimum version 3.8.0), and the protobuf runtime for your language, which installs the language-specific plugin on `protoc`. Follow the instructions on `https://github.com/protocolbuffers/protobuf`.

For additional installation help, see the quick start guides for your language in the gRPC Documentation.

Copy Protobuf Files to Writable Folder

Your local installation of RoadRunner contains the protobuf files that you need to compile in this folder:

RRInstallFolder/bin/*platform*/Proto/

- *RRInstallFolder* is your local RoadRunner installation folder
- *platform* is the folder name for your OS platform.

This table shows the default protobuf location by platform.

Platform	Default Protobuf Location
Windows	`C:\Program Files\RoadRunner R2022b\bin\win64\Proto`
Linux Ubuntu	`/usr/local/RoadRunner_R2022b/bin/glnxa64/Proto`

Inside the `Proto` folder, the top-level `mathworks` folder contains the hierarchy of protobuf files. Copy this folder into the same folder that contains the clients that you intend to write. For example, this figure shows a sample folder containing the `mathworks` folder and a Python client named `bulk_opendrive_export.py`.
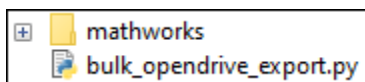
Select Protobuf Files to Compile

The protobuf files that you need to compile depend on the API that you intend to use. This table shows which files to compile and their locations within the `mathworks` root protobuf folder.

Note You installation might not include all protobuf files shown here. The exact protobuf files included in your local RoadRunner installation depend on the RoadRunner products for which you are licensed.

API	Required Products	Protobuf Files to Compile	Protobuf File Locations

RoadRunner service for working with scenes and scenarios. For more details, see:	RoadRunner
	
	RoadRunner Scenario (for scenario workflows)
		`roadrunner_service.proto`
		
		`roadrunner_service_messages.proto`
			`mathworks/roadrunner`

API	Required Products	Protobuf Files to Compile	Protobuf File Locations
• "gRPC API for Scenes and HD Maps" • "gRPC API for Scenarios" (RoadRunner Scenario)		`import_settings.proto`	
		`export_settings.proto`	
		`geometry.proto`	`mathworks/ scenario/common`
HD map interface for importing custom scene data. For more details, see "Build Scenes from Custom Data Using RoadRunner HD Map".	RoadRunner RoadRunner Scene Builder	`hd_map_header.proto`	`mathworks/ scenario/scene/hd`
		`hd_map.proto`	
		`hd_lanes.proto`	
		`hd_lane_markings.proto`	
		`hd_junctions.proto`	
		`common_attributes.proto`	
		`geometry.proto`	`mathworks/ scenario/common`

Compile Protobuf Files

To compile the protobuf files, follow the tutorials for your programming language in the Google protocol buffer documentation: `https://developers.google.com/protocol-buffers`

In the typical process, you run a `protoc` command on the `.proto` files that you want to compile. The `protoc` compiler then generates language-specific bindings based off of those `.proto` files.

For example, this code shows how to compile protobuf files in Python, where:

• `PATH_TO_PYTHON_CLIENT` is the folder path in which you plan to store your Python client.
• `pathToProtos` is the path to the root protobuf folder, `mathworks`, which is relative to your client folder.
• `grpc.tools.protoc` is the compile command from an installed `protoc` compiler

```
import os
import subprocess

pathToClient = r"PATH_TO_PYTHON_CLIENT"
pathToProtos = pathToClient

# Get list of all protobuf files
protoFiles = list()
for root, dirs, files in os.walk(pathToProtos):
    for file in files:
        if file.endswith(".proto"):
            protoFiles.append(os.path.join(root,file))

# Generate protobuf compiler command
command = ('python -m grpc.tools.protoc --proto_path="' + pathToProtos + \
'" --python_out="' + pathToClient + '" --grpc_python_out="' + pathToClient + '"')

for file in protoFiles:
    command += ' "' + file + '"'

print("Compiling protobuf files...")
print("Executing command:\n\n" + command + "\n")

out = subprocess.run(command, check=True)

print("Successfully compiled protobuf files. Generated Python files are located in '"
+ pathToClient + "'")
```

If you need additional help compiling the protocol buffer files into your desired programming language, Contact Support.

Write Clients

The compiled protobuf files, or bindings, contain the language-specific code needed to use the API. In the clients you write, you import these bindings into your code. To learn how to write clients for a few of the languages that gRPC supports, see these examples:

- "Create gRPC Python Client for Controlling RoadRunner Programmatically" on page 6-23
- "Create gRPC C++ Client for Controlling RoadRunner Programmatically" on page 6-28

See Also

Related Examples

- "Control RoadRunner Programmatically Using gRPC API" on page 6-2
- "Build Scenes from Custom Data Using RoadRunner HD Map"

Create gRPC Python Client for Controlling RoadRunner Programmatically

The RoadRunner API enables you to write client applications in multiple programming languages to control RoadRunner programmatically. This example shows how to write a simple Python client that exports all scenes in a project to ASAM OpenDRIVE.

Prerequisites

Before beginning this example, make sure that you meet these prerequisites:

- You are familiar with the language-neutral gRPC framework that the RoadRunner API is built on. For more details, see "Control RoadRunner Programmatically Using gRPC API" on page 6-2.
- You have compiled the protocol buffer (protobuf) schema files that define the RoadRunner API you are using into Python. For more details, see "Compile Protocol Buffers for RoadRunner gRPC API" on page 6-19.
- You have experience writing Python code and have Python installed in your local environment.

Create Client File

Create a Python file named `bulk_opendrive_export.py`. Save this file in the same folder as the Python bindings that you generated by compiling the RoadRunner API protocol buffer (protobuf) files.

This figure shows a sample folder structure, where `mathworks` is the root folder of the compiled bindings.
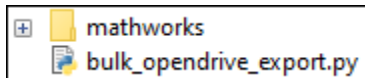
Write Client

Write the Python code for the client application by copying this code into the `bulk_opendrive_export.py` file:

Python Client Code to Copy

```python
"""
Copyright 2021 The MathWorks, Inc.
An example client for exporting RoadRunner scenes to ASAM OpenDRIVE files in bulk.
"""

import grpc
from mathworks.roadrunner import roadrunner_service_messages_pb2
from mathworks.roadrunner import roadrunner_service_pb2_grpc

import os
import sys

def bulk_opendrive_export(project,port,export_folder="OpenDRIVE"):
    """
    Export all RoadRunner scenes in a project to ASAM OpenDRIVE.

    Required arguments:
    project -- Path to RoadRunner project (string)
    port -- Network port on which RoadRunner service is running (string)

    Keyword arguments:
    export_folder -- Export folder path relative to "Exports" folder (string)
    """

    # Connect to RoadRunner API server
    with grpc.insecure_channel("localhost:" + port) as channel:
        api = roadrunner_service_pb2_grpc.RoadRunnerServiceStub(channel)

        # Get all scenes in project
        scenes = [scene for scene in os.listdir(project + "\Scenes") \
                    if scene.endswith(".rrscene")]

        # Load project
        load_project_request = roadrunner_service_messages_pb2.LoadProjectRequest()
        load_project_request.folder_path = project
        api.LoadProject(load_project_request)

        # Export scenes to ASAM OpenDRIVE
        for scene in scenes:
            export_request = roadrunner_service_messages_pb2.ExportRequest()
            file_name = scene.replace(".rrscene",".xodr")
            export_request.file_path = export_folder + "\\" + file_name
            export_request.format_name = "opendrive"
            api.Export(export_request)
            print("Exported scene " + scene + " to " + \
                    project + "\\Exports\\" + export_folder)

        # Exit RoadRunner
        exit_request = roadrunner_service_messages_pb2.ExitRequest()
        api.Exit(exit_request)

if __name__ == "__main__":
    bulk_opendrive_export(sys.argv[1],sys.argv[2],*sys.argv[3:])
```

This client defines a Python function that exports all scenes in a specified RoadRunner project to the ASAM OpenDRIVE file format. The table describes the code in this client application.

Code and Description

Import the gRPC Python library. If you previously compiled the protobuf files to Python, then you have already installed this library. If you do not have the library installed, see "Install gRPC and Protobuf Compiler" on page 6-19.

```
import grpc
```

Import the generated Python bindings for the RoadRunner API. This sample client uses the RoadRunner service API, which enables you to manage projects, scenes, and scenarios.

```
from mathworks.roadrunner import roadrunner_service_messages_pb2
from mathworks.roadrunner import roadrunner_service_pb2_grpc
```

The imported Python bindings are compiled versions of these protobuf files:

- `roadrunner_service.proto`
- `roadrunner_service_messages.proto`

The client imports the Python bindings from the `mathworks` folder that is in the same folder as your client.

```
import os
import sys
```

Import additional Python libraries required for your client. This client uses the `os` library to perform file operations and the `sys` library to accept command-line arguments.

```
def bulk_opendrive_export(project,port,export_folder="OpenDRIVE"):
    """
    Export all RoadRunner scenes in a project to ASAM OpenDRIVE.

    Required arguments:
    project -- Path to RoadRunner project (string)
    port -- Network port on which RoadRunner service is running (string)

    Keyword arguments:
    export_folder -- Export folder path relative to "Exports" folder (string)
    """
```

Define the client in a function or script. This client defines a function, `bulk_opendrive_export`, that exports all scenes in a specified RoadRunner project to ASAM OpenDRIVE. The function connects to RoadRunner at the specified IP network port, `port`.

By default, the function exports scenes to *RoadRunnerProject*/Exports/OpenDRIVE, where *RoadRunnerProject* is the project path specified by `project`. The `export_folder` keyword argument enables you to change the folder containing the exported files relative to *RoadRunnerProject*/Exports.

```
    # Connect to RoadRunner API server
    with grpc.insecure_channel("localhost:" + port) as channel:
```

Establish a local connection to the RoadRunner API server over the gRPC channel.

The gRPC channel for the API server uses insecure channel credentials and has no encryption or authentication. Since RoadRunner is running on a local machine and not connecting to external networks, the security risk is low.

This client connects to the RoadRunner API server using the IP port number that is specified as an input argument to the function.

Code and Description

```
        api = roadrunner_service_pb2_grpc.RoadRunnerServiceStub(channel)
```

Create an object for the RoadRunner service API. This object is called a stub. By using remote procedure call (RPC) methods of this stub, you can remotely control RoadRunner from your client.

```
        # Get all scenes in project
        scenes = [scene for scene in os.listdir(project + "\Scenes") \
                if scene.endswith(".rrscene")]

        # Load project
        load_project_request = roadrunner_service_messages_pb2.LoadProjectRequest()
        load_project_request.folder_path = project
        api.LoadProject(load_project_request)

        # Export scenes to ASAM OpenDRIVE
        for scene in scenes:
            export_request = roadrunner_service_messages_pb2.ExportRequest()
            file_name = scene.replace(".rrscene",".xodr")
            export_request.file_path = export_folder + "\\" + file_name
            export_request.format_name = "opendrive"
            api.Export(export_request)
            print("Exported scene " + scene + " to " + \
                    project + "\\Exports\\" + export_folder)

        # Exit RoadRunner
        exit_request = roadrunner_service_messages_pb2.ExitRequest()
        api.Exit(exit_request)
```

Call RPC methods from the API stub. These methods send protobuf-encoded message requests to the RoadRunner API server. RoadRunner then updates accordingly and returns an empty message in response to indicate that it has processed the request.

After collecting a list of all scenes in the specified project, this client performs these actions:

1   Loads the specified project by using the `LoadProject` method.

2   Exports each scene to the export folder by using the `Export` method.

3   Exits RoadRunner and shuts down the API server by using the `Exit` method.

```
if __name__ == "__main__":
    bulk_opendrive_export(sys.argv[1],sys.argv[2],*sys.argv[3:])
```

Enable the function defined in the client to be called with command-line arguments.

Call Client

After creating the client, you can then open RoadRunner and call the client from the command line.

Use `AppRoadRunner` to open RoadRunner to a project. For example, this code shows how to open RoadRunner from its default installation location on Windows. RoadRunner opens to a project located at `C:\RR\MyProject` on IP network port `54321`. RoadRunner opens to a new scene in the project.

```
cd "C:\Program Files\RoadRunner R2022b\bin\win64"
AppRoadRunner --projectPath C:\RR\MyProject --apiPort 54321
```

Navigate to the location of your client and call the `bulk_opendrive_export` function defined in the client. Specify the name of the project path and the API port that you used to open RoadRunner. By default, this client exports the ASAM OpenDRIVE files to a folder named `Exports/OpenDRIVE` within the project.

```
cd "Path\To\Client"
python bulk_opendrive_export.py C:\RR\MyProject 54321
```

The client opens RoadRunner, exports the scenes, and prints statements to confirm the export. For example:

```
Exported scene FourWaySignal to C:\RR\MyProject\Exports\OpenDRIVE.
```

Navigate to the folder containing the exported files to confirm that they have exported correctly.

```
cd "C:\RR\MyProject\Exports\OpenDRIVE"
ls
```

```
FourWaySignal.xodr
...
```

See Also
AppRoadRunner

Related Examples
- "Control RoadRunner Programmatically Using gRPC API" on page 6-2
- "Compile Protocol Buffers for RoadRunner gRPC API" on page 6-19
- "Create gRPC C++ Client for Controlling RoadRunner Programmatically" on page 6-28

Create gRPC C++ Client for Controlling RoadRunner Programmatically

The RoadRunner API enables you to write client applications in multiple programming languages to control RoadRunner programmatically. This example shows how to write a simple C++ client that exports all scenes in a project to ASAM OpenDRIVE.

Prerequisites

Before beginning this example, make sure that you meet these prerequisites:

- You are familiar with the language-neutral gRPC framework that the RoadRunner API is built on. For more details, see "Control RoadRunner Programmatically Using gRPC API" on page 6-2.
- You have compiled the protocol buffer (protobuf) schema files that define the RoadRunner API you are using into C++. For more details, see "Compile Protocol Buffers for RoadRunner gRPC API" on page 6-19.
- You have experience writing C++ code and have C++ installed in your local environment.

Create Client File

Create a C++ file named `BulkOpenDRIVEExport.cc`. Save this file in the same folder as the C++ bindings that you generated by compiling the RoadRunner API protocol buffer (protobuf) files.

This figure shows a sample folder structure, where `mathworks` is the root folder of the compiled bindings.
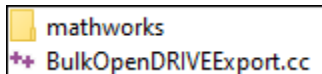
Write Client

Write the C++ code for the client application by copying this code into the `BulkOpenDRIVEExport.cc` file:

C++ Client Code to Copy

```cpp
/*
Copyright 2021 The MathWorks, Inc.
An example client for exporting RoadRunner scenes to ASAM OpenDRIVE files in bulk.
 */

#include "mathworks/roadrunner/roadrunner_service.grpc.pb.h"

#include <filesystem>
#include <grpcpp/grpcpp.h>
#include <iostream>
#include <string>
#include <vector>

using grpc::ClientContext;
using grpc::Status;

using namespace std;
using namespace mathworks::roadrunner;

// Call Signature: ./BulkOpenDRIVEExport C:/RR/MyProject 54321 OpenDRIVE"
int main(int argc, char **argv) {

  // Connect to RoadRunner API server
  unique_ptr<RoadRunnerService::Stub> api(
      RoadRunnerService::NewStub(grpc::CreateChannel(
          "localhost:" + string(argv[2]), grpc::InsecureChannelCredentials())));

  // Load Project
  string projectPath = argv[1];
  LoadProjectRequest loadProjectRequest;
  loadProjectRequest.set_folder_path(projectPath);
  LoadProjectResponse loadProjectResponse;
  ClientContext context;
  Status status = api->LoadProject(&context, loadProjectRequest, &loadProjectResponse);
  if (!status.ok()) {
    cout << status.error_code() << ": " << status.error_message() << endl;
    return -1;
  }

  // Get all scenes in project and export scenes to OpenDRIVE
  string exportFolder = argv[3];
  ExportRequest exportRequest;
  exportRequest.set_format_name("opendrive");
  for (const auto &file :
       filesystem::directory_iterator(projectPath + "/Scenes")) {
    const string fileName = filesystem::path(file.path()).filename().string();
    size_t extPos = fileName.find(".rrscene");
    if (extPos != string::npos) {
      exportRequest.set_file_path(exportFolder + "/" + fileName.substr(0,extPos) + ".xodr");
      ExportResponse exportResponse;
      ClientContext context;
      Status status = api->Export(&context, exportRequest, &exportResponse);
      if (!status.ok()) {
        cout << status.error_code() << ": " << status.error_message() << endl;
      }
      else {
          cout << "Exported scene " + fileName + " to " + projectPath + "/Exports/" + exportFolder << endl;
      }
    }
  }

  // Exit RoadRunner
  ExitRequest exitRequest;
  ExitResponse exitResponse;
  ClientContext context;
  Status status = api->Exit(&context, exitRequest, &exitResponse);
  if (!status.ok()) {
    cout << status.error_code() << ": " << status.error_message() << endl;
    return -1;
  }
```

```
   return 0;
}
```

This client defines a C++ function that exports all scenes in a specified RoadRunner project to the ASAM OpenDRIVE file format. The table describes the code in this client application.

Code and Description

Include the libraries that are required to call the client. These libraries include:

- The generated C++ bindings for the RoadRunner API. This sample client uses the RoadRunner service API, which enables you to manage projects, scenes, and scenarios. The imported C++ binding is a compiled version of the `roadrunner_service.proto` file. The client imports these bindings from the `mathworks` folder that is in the same folder as your client.

- The gRPC C++ library (`grpcpp/grpcpp.h`). If you previously compiled the protobuf files to C++, then you have already installed this library. If you do not have the library installed, see "Install gRPC and Protobuf Compiler" on page 6-19.

- Additional C++ libraries required for your client. This client uses the standard C++ libraries `<filesystem>`, `<iostream>`, `<string>`, and `<vector>`.

```
#include "mathworks/roadrunner/roadrunner_service.grpc.pb.h"

#include <filesystem>
#include <grpcpp/grpcpp.h>
#include <iostream>
#include <string>
#include <vector>

using grpc::ClientContext;
using grpc::Status;

using namespace std;
using namespace mathworks::roadrunner;
```

Define the client in a function or script. This client defines a function that exports all scenes in a RoadRunner project to ASAM OpenDRIVE. The function accepts three required command-line arguments.

1. The project from which you want to export scenes.
2. The IP network port used to establish a connection with the RoadRunner API server.
3. The folder to export scenes to, relative to the `Exports` folder of the specified project.

```
// Call Signature: ./BulkOpenDRIVEExport C:/RR/MyProject 54321 OpenDRIVE"
int main(int argc, char **argv) {
```

Create a pointer object for the RoadRunner service API. This object is called a stub. By using the remote procedure call (RPC) methods of this stub, you can remotely control RoadRunner from your client.

Use this stub to establish a local connection to your RoadRunner API server over the gRPC channel.

The gRPC channel for the API server uses insecure channel credentials and has no encryption or authentication. Since RoadRunner is running on a local machine and not connecting to external networks, the security risk is low.

This client connects to the RoadRunner API server using the IP port number that is specified as an input argument to the function.

```
  // Connect to RoadRunner API server
  unique_ptr<RoadRunnerService::Stub> api(
      RoadRunnerService::NewStub(grpc::CreateChannel(
          "localhost:" + string(argv[2]), grpc::InsecureChannelCredentials())));
```

Code and Description

Call RPC methods from the API stub. These methods send protobuf-encoded message requests to the RoadRunner API server. RoadRunner then updates accordingly and returns an empty message in response to indicate that it has processed the request.

The client performs these actions:

1   Loads the specified project by using the `LoadProject` method.

2   Exports each scene in the project to the export folder by using the `Export` method.

3   Exits RoadRunner and shuts down the API server by using the `Exit` method.

```
// Load Project
string projectPath = argv[1];
LoadProjectRequest loadProjectRequest;
loadProjectRequest.set_folder_path(projectPath);
LoadProjectResponse loadProjectResponse;
ClientContext context;
Status status = api->LoadProject(&context, loadProjectRequest, &loadProjectResponse);
if (!status.ok()) {
  cout << status.error_code() << ": " << status.error_message() << endl;
  return -1;
}

// Get all scenes in project and export scenes to OpenDRIVE
string exportFolder = argv[3];
ExportRequest exportRequest;
exportRequest.set_format_name("opendrive");
for (const auto &file :
     filesystem::directory_iterator(projectPath + "/Scenes")) {
  const string fileName = filesystem::path(file.path()).filename().string();
  size_t extPos = fileName.find(".rrscene");
  if (extPos != string::npos) {
    exportRequest.set_file_path(exportFolder + "/" + fileName.substr(0,extPos) + ".xodr");
    ExportResponse exportResponse;
    ClientContext context;
    Status status = api->Export(&context, exportRequest, &exportResponse);
    if (!status.ok()) {
      cout << status.error_code() << ": " << status.error_message() << endl;
    }
    else {
        cout << "Exported scene " + fileName + " to " + projectPath + "/Exports/" + exportFolder << endl;
    }
  }
}

// Exit RoadRunner
ExitRequest exitRequest;
ExitResponse exitResponse;
ClientContext context;
Status status = api->Exit(&context, exitRequest, &exitResponse);
if (!status.ok()) {
  cout << status.error_code() << ": " << status.error_message() << endl;
  return -1;
}

return 0;
}
```

Call Client

After creating the client, you can then open RoadRunner and call the client from the command line.

Use `AppRoadRunner` to open RoadRunner to a project. For example, this code shows how to open RoadRunner from its default installation location on Windows. RoadRunner opens to a project located at `C:\RR\MyProject` on IP network port `54321`. RoadRunner opens to a new scene in the project.

```
cd "C:\Program Files\RoadRunner R2022b\bin\win64"
AppRoadRunner --projectPath C:\RR\MyProject --apiPort 54321
```

Navigate to the location of your client and call the `BulkOpenDRIVEExport` function defined in the client. Specify the name of the project path and the API port that you used to open RoadRunner. By default, this client exports the ASAM OpenDRIVE files to a folder named `Exports/OpenDRIVE` within the project.

```
cd "Path\To\Client"
BulkOpenDRIVEExport C:\RR\MyProject 54321
```

The client opens RoadRunner, exports the scenes, and prints statements to confirm the export. For example:

```
Exported scene FourWaySignal.rrscene to C:/RR/MyProject/Exports/OpenDRIVE.
```

Navigate to the folder containing the exported files to confirm that they have exported correctly.

```
cd "C:\RR\MyProject\Exports\OpenDRIVE"
ls
```

```
FourWaySignal.xodr
...
```

See Also
AppRoadRunner

Related Examples
- "Control RoadRunner Programmatically Using gRPC API" on page 6-2
- "Compile Protocol Buffers for RoadRunner gRPC API" on page 6-19
- "Create gRPC Python Client for Controlling RoadRunner Programmatically" on page 6-23

Control RoadRunner Programmatically in Console Mode

RoadRunner provides programmatic interfaces for performing common workflow tasks such as opening, closing, and saving scenes and projects and importing and exporting scenes. You can use MATLAB functions or gRPC APIs for performing these common workflow tasks. The MATLAB functions operate through a MATLAB object that opens and sets up the main RoadRunner user interface so that you can control it programmatically. These MATLAB functions require an Automated Driving Toolbox™ license. The gRPC API uses remote procedure call (RPC) methods, defined in a set of protocol buffer (protobuf) schema files, that enable you to remotely control RoadRunner programmatically. Using the gRPC framework, you can compile these files into C++, Python, or another programming language supported by gRPC. You can then write client applications that call these methods for controlling RoadRunner in the language of your choice.

Using the MATLAB functions and gRPC APIs, you can launch RoadRunner in console mode also allowing RoadRunner to be run from the terminal in a non-graphical environment. This enables distributed workflows, for example, running RoadRunner from a remote server, and empowers you to use RoadRunner at a much larger scale. RoadRunner is often used as a part of a pipeline in simulators where it may be required to import a scene to RoadRunner and export it to a format that is compatible with other simulators. Using RoadRunner in console mode improves integration with such continuous integration(CI) systems. It also improves application performance by reducing the time it takes to load scenes in RoadRunner because you do not need to load and display scene graphics.

Like the interactive RoadRunner, RoadRunner Console can receive API commands sent from the programmatic interface. You can write your own clients by compiling the proto files, using the `CmdRoadRunnerApi` app, or the MATLAB functions for RoadRunner. If you are using the gRPC APIs, use `AppRoadRunner` with the `--nodisplay` argument to launch RoadRunner in a non-graphical environment. RoadRunner provides a precompiled helper command, `CmdRoadRunnerApi`, that enables you to call RoadRunner RPC methods from the command line. This helper command is located in the same folder as the `AppRoadRunner` executable. If you are using MATLAB, use the `roadrunner` object with the NoDisplay property to launch RoadRunner in a non-graphical environment.

The following examples show how you can export a RoadRunner scene in console mode using MATLAB functions and gRPC APIs.

Export RoadRunner Scene in Console Mode Using MATLAB

This example shows how to export a scene from a RoadRunner project to one of the file formats supported by RoadRunner. In this example, you export a scene to the ASAM OpenDRIVE® file format using MATLAB® functions.

To run this example, you must:

- Have an Automated Driving Toolbox® license.
- Have a RoadRunner® license and the product is installed.
- Have created a RoadRunner project folder.

Start RoadRunner Programmatically

To use MATLAB functions to control RoadRunner 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_*R20NNx*/bin/glnxa64

R20NNx is the MATLAB version for the release you are using.

If your RoadRunner installation is at a different location than the default location, use MATLAB `settings` API to customize the default value of the RoadRunner installation folder.

Export Scene from RoadRunner to ASAM OpenDRIVE

Export a scene from a RoadRunner project to the ASAM OpenDRIVE format using MATLAB.

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.

```
projectFolder = "C:\RR\MyProject";
rrApp = roadrunner(projectFolder,InstallationFolder="C:\Program Files\RoadRunner R2022b\bin\win64
```

Open a scene in the project by using the `openScene` function with the `roadrunner` object and the RoadRunner scene you wish to open as input arguments. This example uses the `FourWaySignal.rrscene` scene, which is one of the scenes included by default in RoadRunner projects, `and` is located in the `Scenes` folder of the project.

```
sceneName = "FourWaySignal.rrscene";
openScene(rrApp,sceneName);
```

Set export options by creating an `openDriveExportOptions` object to enable export of signals and objects from the file.

```
options = openDriveExportOptions(OpenDriveVersion=1.5,ExportSignals=true,ExportObjects=true);
```

Use the `exportScene` function to export the scene to ASAM OpenDRIVE. Specify your `roadrunner` object, the name of the file to which you want to export the scene, the export format, and the export options as input arguments to the `exportScene` function.

```
filename = "FourWaySignal.xodr";
formatname = "OpenDRIVE";
exportScene(rrApp,filename,formatname,options);
```

Navigate to the `Exports` folder and open the exported ASAM OpenDRIVE file. Verify that the <header> tag contains the specified file version. The Exports folder contains only the ASAM OpenDRIVE file and corresponding GeoJSON file.

Export RoadRunner Scene in Console mode using gRPC APIs

This section shows how you can export a scene in console mode using gRPC APIs. In this example, you export a RoadRunner scene to the ASAM OpenDRIVE® file format using command-line operations. For more details on gRPC APIs, consider previewing "Control RoadRunner Programmatically Using gRPC API" on page 6-2.

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. It is recommended to use MinGW® because some Windows shells do not output the standard output from `AppRoadRunner` to the command line.

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:

- (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`.
- Specify the `nodisplay` option to start RoadRunner without a GUI window.

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

Preview the programmatic operations by exporting one scene from the current project to ASAM OpenDRIVE. To perform these operations, you use the `CmdRoadRunnerApi` helper command. This helper command is located in the same folder as the `AppRoadRunner` executable file.

Load the `FourWaySignal` scene by using the `LoadScene` RPC method. This scene is included by default in the `Scenes` 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 this option.

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

Export the scene to the ASAM OpenDRIVE version 1.6 format by using the `Export` RPC method.

- For the exported file name, specify the same name as the scene. By default, RoadRunner exports the scene to the `Exports` folder of the opened project.
- In the `serverAddress` option, replace the port number with the `apiPort` value you specified when opening RoadRunner.

```
CmdRoadRunnerApi "Export(file_path='FourWaySignal.xodr' format_name='opendrive' ^
open_drive_settings.open_drive_version='1.6')" --serverAddress=localhost:54321
```

Navigate to the `Exports` folder and open the exported ASAM OpenDRIVE file. Verify that the `<header>` tag contains the specified file version.

```
cd "C:\RR\MyProject\Exports"
FourWaySignal.xodr
```

```
<OpenDRIVE>
    <header revMajor="1" revMinor="6" ...>
    ...
```

If your project included only the default scene, then your folder contains only the ASAM OpenDRIVE file and corresponding GeoJSON file.

```
FourWaySignal.xodr
FourWaySignal.geojson
```

See Also

roadrunner | openScene | exportScene | AppRoadRunner | LoadScene | Export

Related Examples

- "Control RoadRunner Programmatically Using gRPC API" on page 6-2
- "Export Multiple Scenes Using MATLAB" on page 6-38

Export Multiple Scenes Using MATLAB

This example shows how to bulk-export scenes from a RoadRunner project to one of the file formats supported by RoadRunner. In this example, you export scenes to the ASAM OpenDRIVE® file format using MATLAB® functions.

To run this example, you must:

- Have an Automated Driving Toolbox® license.
- Have a RoadRunner® license and the product is installed.
- Have created a RoadRunner project folder.

Start RoadRunner Programmatically

To use MATLAB functions to control RoadRunner 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_*R20NNx*/bin/glnxa64

R20NNx is the MATLAB version for the release you are using.

If your RoadRunner installation is at a different location than the default location, use MATLAB `settings` API to customize the default value of the RoadRunner installation folder.

Export Scene from RoadRunner to ASAM OpenDRIVE

Export a scene from a RoadRunner project to the ASAM OpenDRIVE format using MATLAB.

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.

```
projectFolder = "C:\RR\MyProject";
rrApp = roadrunner(projectFolder,InstallationFolder="C:\Program Files\RoadRunner R2022b\bin\win64
```

Open a scene in the project by using the `openScene` function with the `roadrunner` object and the RoadRunner scene you wish to open as input arguments. This example uses the `FourWaySignal.rrscene` scene, which is one of the scenes included by default in RoadRunner projects, and is located in the `Scenes` folder of the project.

```
sceneName = "FourWaySignal.rrscene";
openScene(rrApp,sceneName);
```

Set export options by creating an `openDriveExportOptions` object to enable export of signals and objects from the file.

```
options = openDriveExportOptions(OpenDriveVersion=1.5,ExportSignals=true,ExportObjects=true);
```

Use the `exportScene` function to export the scene to ASAM OpenDRIVE. Specify your `roadrunner` object, the name of the file to which you want to export the scene, the export format, and the export options as input arguments to the `exportScene` function.

```
filename = "FourWaySignal.xodr";
formatname = "OpenDRIVE";
exportScene(rrApp,filename,formatname,options);
```

Export Multiple Scenes from RoadRunner to ASAM OpenDRIVE Format

Export multiple scenes in a RoadRunner project to ASAM OpenDRIVE® format using MATLAB.

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.

```
demoProj = fullfile('C:','DemoProject');
rrApp = roadrunner(demoProj,InstallationFolder="C:\Program Files\RoadRunner R2022b\bin\win64");
```

Specify the path to the scene files you wish to export. You must specify the path to the `Scenes` folder in your RoadRunner project, which contains all the scenes in that project.

```
sceneFiles = dir(fullfile(demoProj,'Scenes','*.rrscene'));
scenes = {sceneFiles.name};
```

Specify the path to your export folder. this is the folder into which RoadRunner exports all your scene files. Iterate through all the scene files, opening each scene using the `openScene` function and then calling the `exportScene` function to export the open scene to the ASAM OpenDRIVE format.

```
exportFolder = fullfile('C:','OpenDRIVE');
for sndx = 1:numel(scenes)
    openScene(rrApp,scenes{sndx});
    [~,fileName] = fileparts(scenes{sndx});
    exportFilePath = [fullfile(exportFolder,fileName) '.xodr'];
    exportScene(rrApp,exportFilePath,'OpenDRIVE');
end
```

Once all the scene files have been exported, close the RoadRunner application by using the `close` function.

```
close(rrApp);
```

Extend RoadRunner Export Options

To customize the script further, you can specify non-default export settings or specify other file formats. For more details on supported formats, see the `exportScene` function. For additional flexibility in exporting scenes, consider exporting the scene using custom export options. For more details, see the `exportCustomFormat` function.

See Also

roadrunner | openScene | exportScene | close

Related Examples

- "Export to ASAM OpenDRIVE" on page 5-24
- "Convert Asset Data Between RoadRunner and ASAM OpenDRIVE" on page 5-16

Convert Scenes Between Formats Using MATLAB Functions

This example shows how to import RoadRunner scenes from one file format and export those scenes to a different format. In this example, you import ASAM OpenDRIVE® files into scenes, save them to a project, and export the scenes to export them to Filmbox® FBX® files using MATLAB® functions.

To run this example, you must:

- Have an Automated Driving Toolbox® license.
- Have a RoadRunner® license and the product is installed.
- Have created a RoadRunner project folder.

Start RoadRunner Programmatically

To use MATLAB functions to control RoadRunner 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_*R20NNx*/bin/glnxa64

R20NNx is the MATLAB release you are using.

If your RoadRunner installation is at a different location than the default location, use MATLAB `settings` API to customize the default value of the RoadRunner installation folder.

Import and Export of Multiple Scenes

Import multiple scenes from the ASAM OpenDRIVE format and export them to the FBX format.

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.

```
demoProj = fullfile('C:','DemoProject');
rrApp = roadrunner(demoProj,InstallationFolder="C:\Program Files\RoadRunner R2022b\bin\win64 ");
```

Specify the paths to the ASAM OpenDRIVE files you want to import and to the folder into which you want to export the Filmbox files.

```
odrFolder = fullfile('C:','OpenDRIVE');
odrFiles = dir(fullfile(odrFolder,'*.xodr'));
exportFolder = fullfile('C:','Filmbox');
```

Import the ASAM OpenDRIVE files and export them to the FBX format. Import each ASAM OpenDRIVE file into a new scene by using the `newScene` function to create a new scene in your project, then specifying the file path for each scene to import to the `importScene` function. Then, export the imported scene to a file by using the `exportScene` function.

```
for fndx = 1:length(odrFiles)
    newScene(rrApp);
```

```
        importFilePath = fullfile(odrFolder,odrFiles(fndx).name);
        importScene(rrApp,importFilePath,"OpenDRIVE");
        [~,fileName] = fileparts(odrFiles(fndx).name);
        exportFilePath = [fullfile(exportFolder,fileName) '.fbx'];
        exportScene(rrApp,exportFilePath,"Filmbox");
end
```

Once all the scenes have been exported, close the RoadRunner application by using the `close` function.

```
close(rrApp);
```

Extend RoadRunner Export Options

To customize the script further, you can specify non-default import and export settings or specify other file formats. For more details on supported formats, see the `importScene` and `exportScene` functions. For additional flexibility in exporting scenes, consider exporting the scene using custom export options. For more details, see the `exportCustomFormat` function.

See Also

roadrunner | newScene | exportScene | exportCustomFormat | importScene | close

Related Examples

- "Importing ASAM OpenDRIVE Files" on page 3-2
- "Export to FBX" on page 5-3

Build Simple Roads Programatically Using RoadRunner HD Map

RoadRunner HD Map is a road data model for representing high-definition (HD) map data in a RoadRunner scene. The model defines a simple structure to represent road layouts using lanes, lane boundaries, lane markings, and junctions. This example shows how to build simple roads using RoadRunner HD Map MATLAB® objects and functions. You can then import the roads into RoadRunner. Use these steps to construct a road and import it into a RoadRunner scene:

- Build an HD map in MATLAB.
- Verify the representation of lanes and lane boundaries by plotting the map in MATLAB.
- Write the map to a RoadRunner HD Map (`.rrhd`) file.
- Import the file into RoadRunner and preview the RoadRunner HD Map data.
- Build a RoadRunner scene from the imported file.

Create Straight, Unmarked Road

To define a fixed-width road, you must specify a series of *xy*-coordinates for the center of the road at various points along its path. First, create a straight, unmarked road, as shown in this figure. Then, plot the road in MATLAB and save it to a binary file.
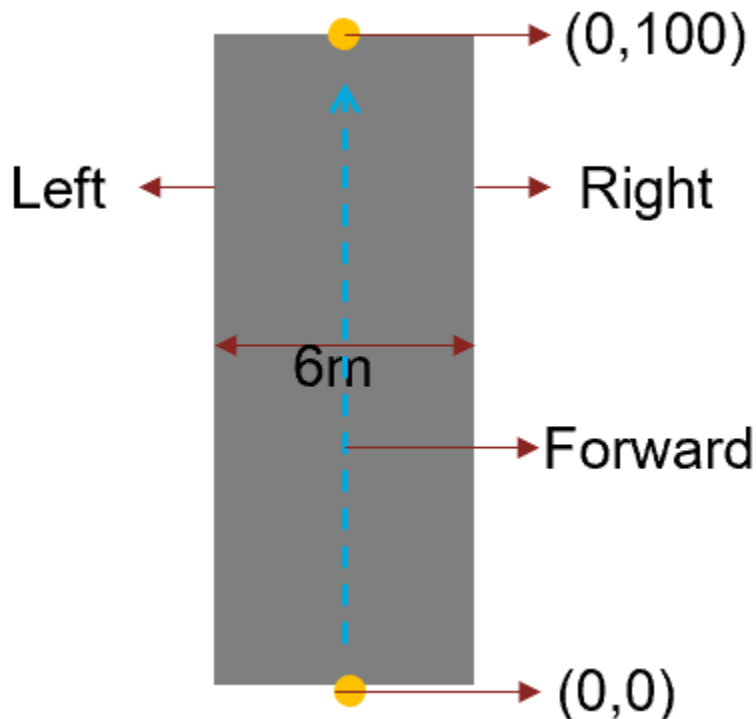
Create an empty RoadRunner HD Map as a `roadrunnerHDMap` object.

```
rrMap = roadrunnerHDMap;
```

Define the center of the straight road as a matrix that contains three sets of *xy*-coordinates specifying the center of the road. Also, define the width of the road.

```
roadCenters = [0 0; 0 50; 0 100];
roadWidth = 6;
```

Create the left and right boundaries of the road using the `roadrunner.hdmap.LaneBoundary` object. Specify the lane boundary information for the lane ID and the coordinates defining the lane geometry.

```
laneGeometryLeft = roadCenters-[roadWidth/2 0];
laneGeometryRight = roadCenters+[roadWidth/2 0];
rrMap.LaneBoundaries(1) = roadrunner.hdmap.LaneBoundary(ID="Left",Geometry=laneGeometryLeft);
rrMap.LaneBoundaries(2) = roadrunner.hdmap.LaneBoundary(ID="Right",Geometry=laneGeometryRight);
```

Create the road lane using the `roadrunner.hdmap.Lane` object. Specify the lane information for the lane ID, the coordinates defining the lane geometry, the driving direction, and the lane type.

```
rLane = roadrunner.hdmap.Lane(ID="Lane",Geometry=roadCenters,TravelDirection="Forward",LaneType="
```

Link the lane boundaries to the lane. Define the left and the right lane boundaries for the lane, and specify the alignments between the lane and lane boundaries.

```
leftBoundary(rLane,"Left",Alignment="Forward");
rightBoundary(rLane,"Right",Alignment="Forward");
rrMap.Lanes = rLane;
```

Plot the lane centers and lane boundaries to preview the lanes and lane boundaries before importing them into RoadRunner.

```
plot(rrMap)
```
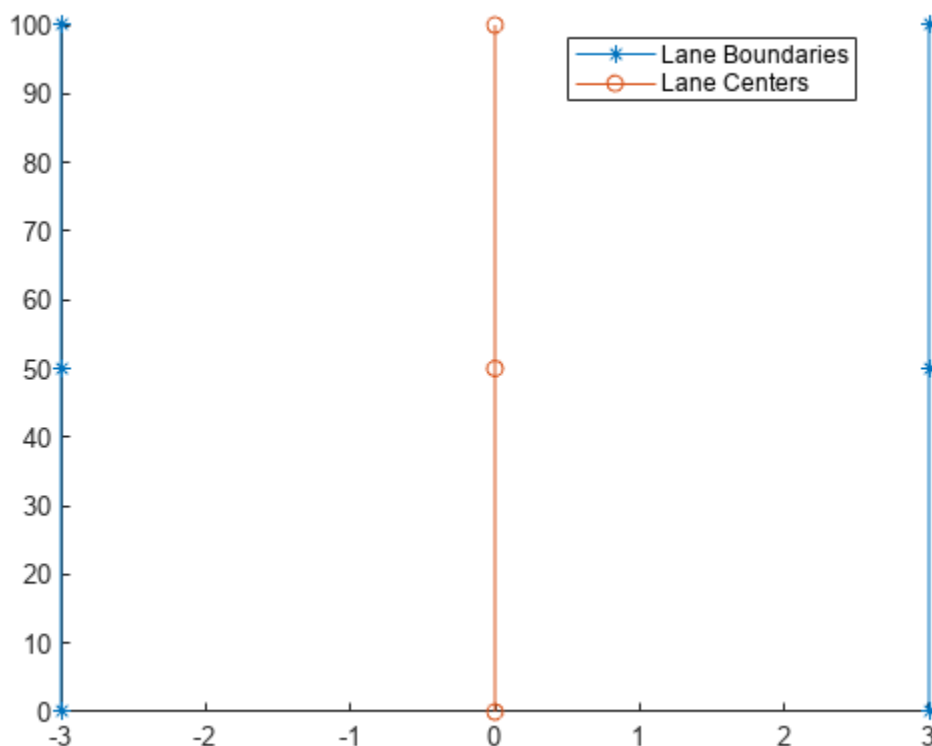
Write the HD map to a file using the `write` function.

```
write(rrMap,"straightRoad.rrhd")
```

Import HD Map File into RoadRunner

For detailed instructions on importing a RoadRunner HD Map file with the `.rrhd` extension into RoadRunner, previewing the map, and building the scene, see "Import Custom Data Using RoadRunner HD Map" on page 3-21.

This figure shows a scene of a straight, unmarked road built using RoadRunner Scene Builder.
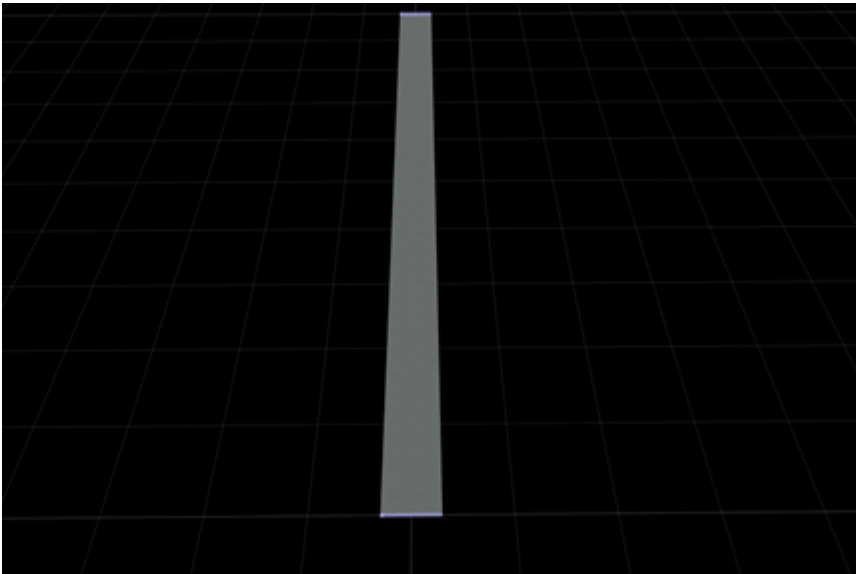
Add Markings to Straight Road

In this section, you add solid, white lane markings to the left and right lane boundaries of the straight road you created in the previous section. To specify a lane marking, you must use an asset in RoadRunner. In this example, you use assets that are part of the "RoadRunner Asset Types" on page 2-45. Specify these assets in the map by using a relative path to the RoadRunner project folder.

Define the path to the solid, white lane marking asset using the `roadrunner.hdmap.RelativeAssetPath` function.

```
solidWhiteAsset = roadrunner.hdmap.RelativeAssetPath(AssetPath="Assets/Markings/SolidSingleWhite
```

Create a solid, white lane marking on the straight road using the `roadrunner.hdmap.LaneMarking` object. Specify the lane marking information for the lane marking ID and the path to the asset.

```
rrMap.LaneMarkings = roadrunner.hdmap.LaneMarking(ID="SolidWhite",AssetPath=solidWhiteAsset);
```

Create a reference for the solid, white marking using the `roadrunner.hdmap.MarkingReference` object. You can use this reference to apply the marking to the lane boundaries.

```
markingRefSW = roadrunner.hdmap.MarkingReference(MarkingID=roadrunner.hdmap.Reference(ID="SolidWl
```

Use parametric attributes to apply this lane marking to the full length of the left and right lane boundaries.

```
markingSpan = [0 1];
markingAttribSW = roadrunner.hdmap.ParametricAttribution(MarkingReference=markingRefSW,Span=mark
rrMap.LaneBoundaries(1).ParametricAttributes = markingAttribSW;
rrMap.LaneBoundaries(2).ParametricAttributes = markingAttribSW;
```

Write the modified HD map to a file.

```
write(rrMap,"straightRoadWithMarkings.rrhd");
```

This figure shows a scene of a straight road with solid, white lane markings for the left and right boundaries built using RoadRunner Scene Builder.
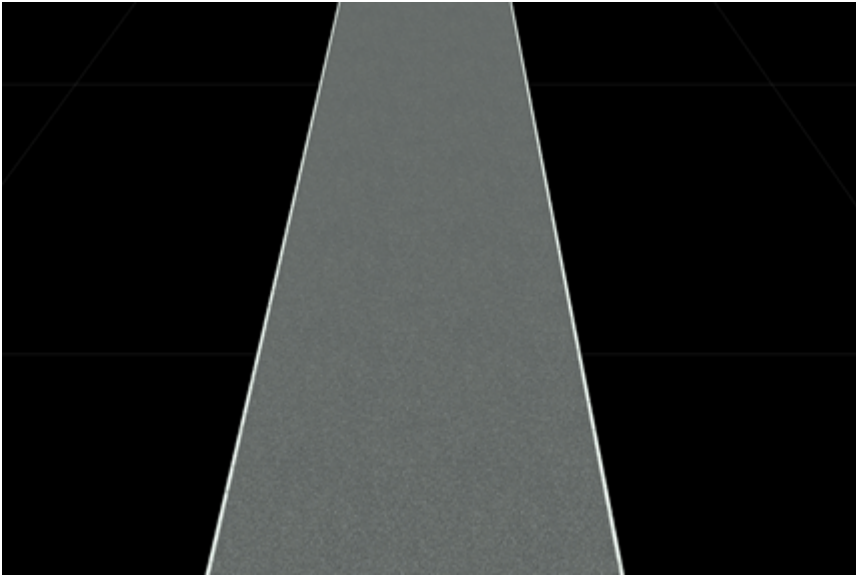
Create Two-Way Road

A two-way road has two lanes with opposite driving directions. A solid yellow lane marking separates the lanes. In this section, you create a straight, two-way road, as shown in this figure, using the same road centers and road width as for the straight, unmarked road.
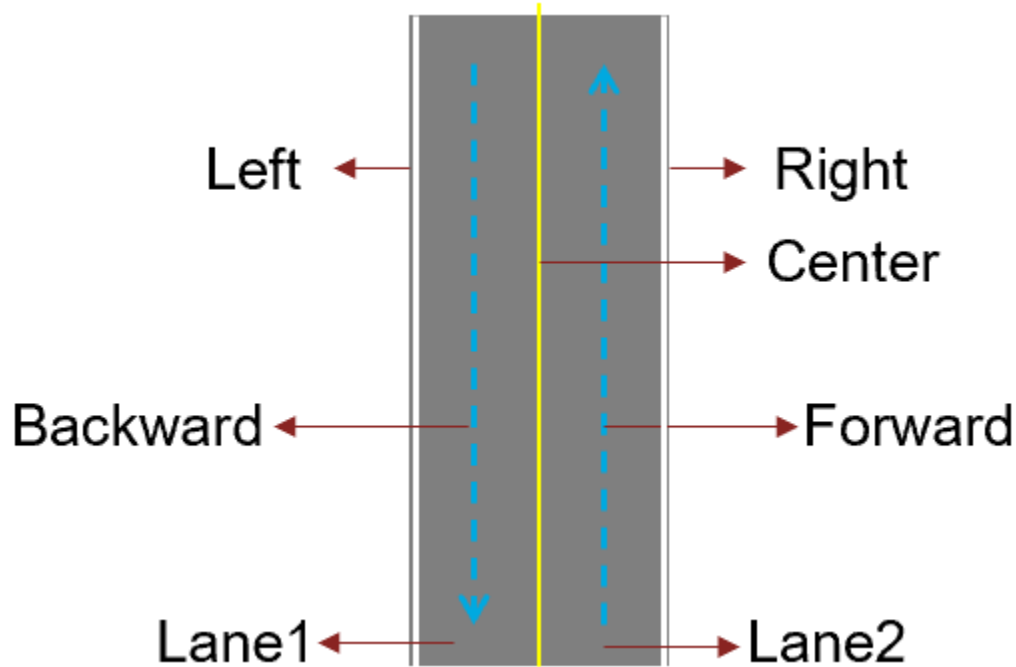
Create an empty RoadRunner HD Map as a `roadrunnerHDMap` object.

```
rrMap = roadrunnerHDMap;
```

Specify the lane and the lane boundaries. In this example, preinitialization of these values results in improved performace as the number of objects in the map increases.

```
rrMap.Lanes(2,1) = roadrunner.hdmap.Lane();
rrMap.LaneBoundaries(3,1) = roadrunner.hdmap.LaneBoundary();
```

Assign the `Lane` property values. Use the `deal` function to match up the input and the output lists.

```
roadCenterLeft = roadCenters-[roadWidth/4 0];
roadCenterRight = roadCenters+[roadWidth/4 0];
[rrMap.Lanes.ID] = deal("Lane1","Lane2");
[rrMap.Lanes.Geometry] = deal(roadCenterLeft,roadCenterRight);
[rrMap.Lanes.TravelDirection] = deal("Backward","Forward");
[rrMap.Lanes.LaneType] = deal("Driving");
```

Assign the `LaneBoundaries` property values. In this example, the center lane is shared between `Lane1` and `Lane2`.

```
[rrMap.LaneBoundaries.ID] = deal("Left","Center","Right");
[rrMap.LaneBoundaries.Geometry] = deal(roadCenters-[roadWidth/2 0], ...
  roadCenters,roadCenters+[roadWidth/2 0]);
```

Link the lane boundaries to the lanes. Define the left and the right lane boundaries for each lane, and specify the alignments between the lanes and lane boundaries.

```
leftBoundary(rrMap.Lanes(1),"Left",Alignment="Forward");
rightBoundary(rrMap.Lanes(1),"Center",Alignment="Forward");
leftBoundary(rrMap.Lanes(2),"Center",Alignment="Forward");
rightBoundary(rrMap.Lanes(2),"Right",Alignment="Forward");
```

Add a solid, yellow marking, in addition to the solid, white marking to the HD map. Define the path to the solid, yellow lane marking asset using the `roadrunner.hdmap.RelativeAssetPath` function.

```
solidYellowAsset = roadrunner.hdmap.RelativeAssetPath(AssetPath="Assets/Markings/SolidSingleYell
```

Create a solid, yellow lane marking and solid, white lane markings on the straight road using the `roadrunner.hdmap.LaneMarking` object. Specify the lane marking information for the lane marking IDs and the paths to the assets.

```
rrMap.LaneMarkings(2,1) = roadrunner.hdmap.LaneMarking();
[rrMap.LaneMarkings.ID] = deal("SolidWhite","SolidYellow");
[rrMap.LaneMarkings.AssetPath] = deal(solidWhiteAsset,solidYellowAsset);
```

Assign the white marking to the lane boundaries at the lane edges and the yellow marking to the center lane boundary. These markings span the entire length of the boundaries.

```
markingRefSY = roadrunner.hdmap.MarkingReference(MarkingID=roadrunner.hdmap.Reference(ID="SolidYe
markingAttribSY = roadrunner.hdmap.ParametricAttribution(MarkingReference=markingRefSY,Span=mark:
[rrMap.LaneBoundaries.ParametricAttributes] = deal(markingAttribSW,markingAttribSY,markingAttribS
```
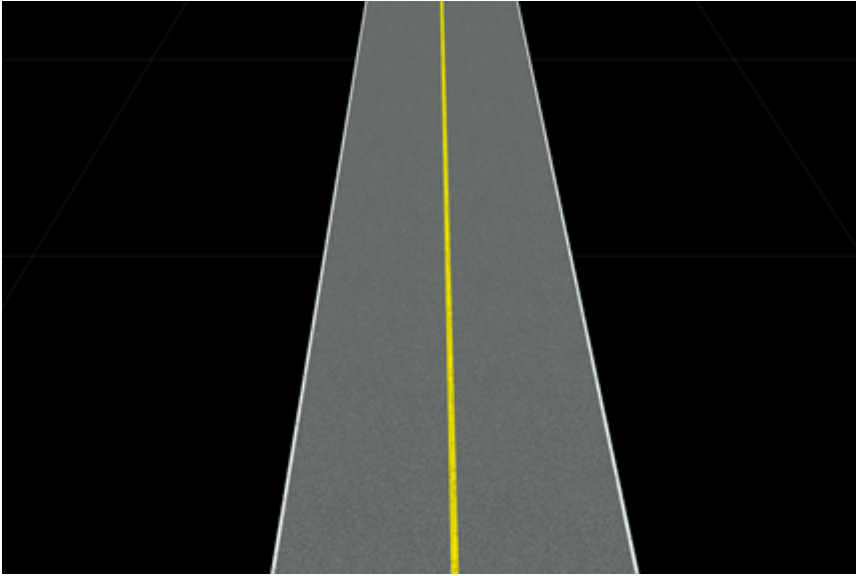
Plot the lane centers and lane boundaries.

```
plot(rrMap)
```
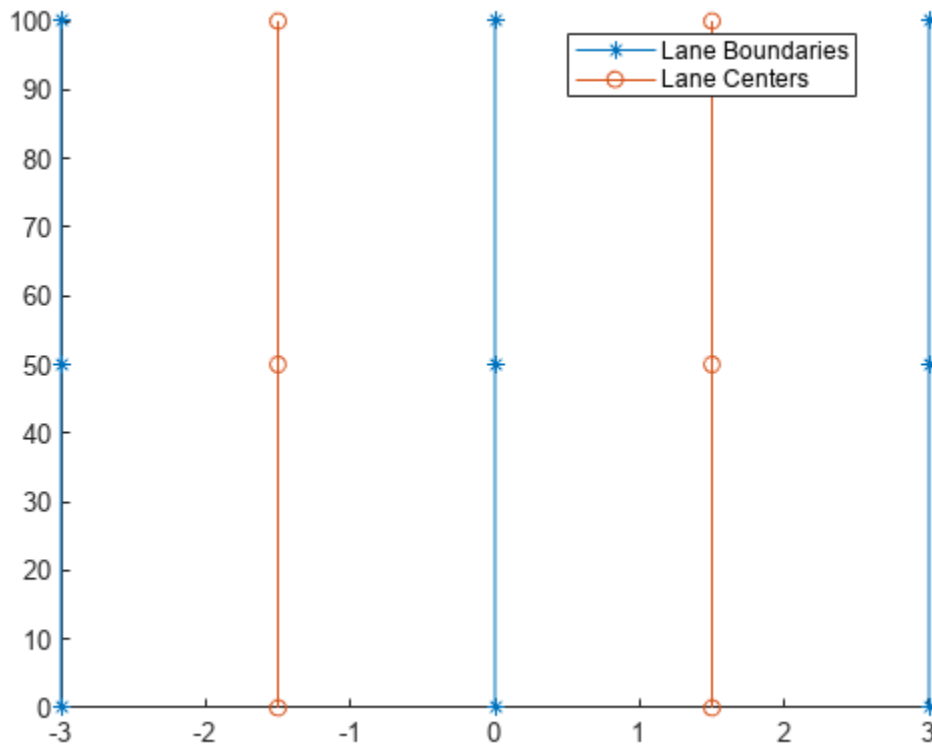

Write the HD map to a file.

```
write(rrMap,"twoWayRoad.rrhd")
```

This figure shows a scene of the two-way road defined in this section, built using RoadRunner Scene Builder.

Add Lane to One-Way Road

In this section, you add a lane to a one-way road. Use a dashed, white marking to separate two lanes with the same travel direction. To add a lane to a one-way road, you must split one lane into two from the left edge of the lane. You must create additional lanes and lane boundaries in the RoadRunner HD map where the lane splits. This figure shows a one-way road with an added lane on the left side:
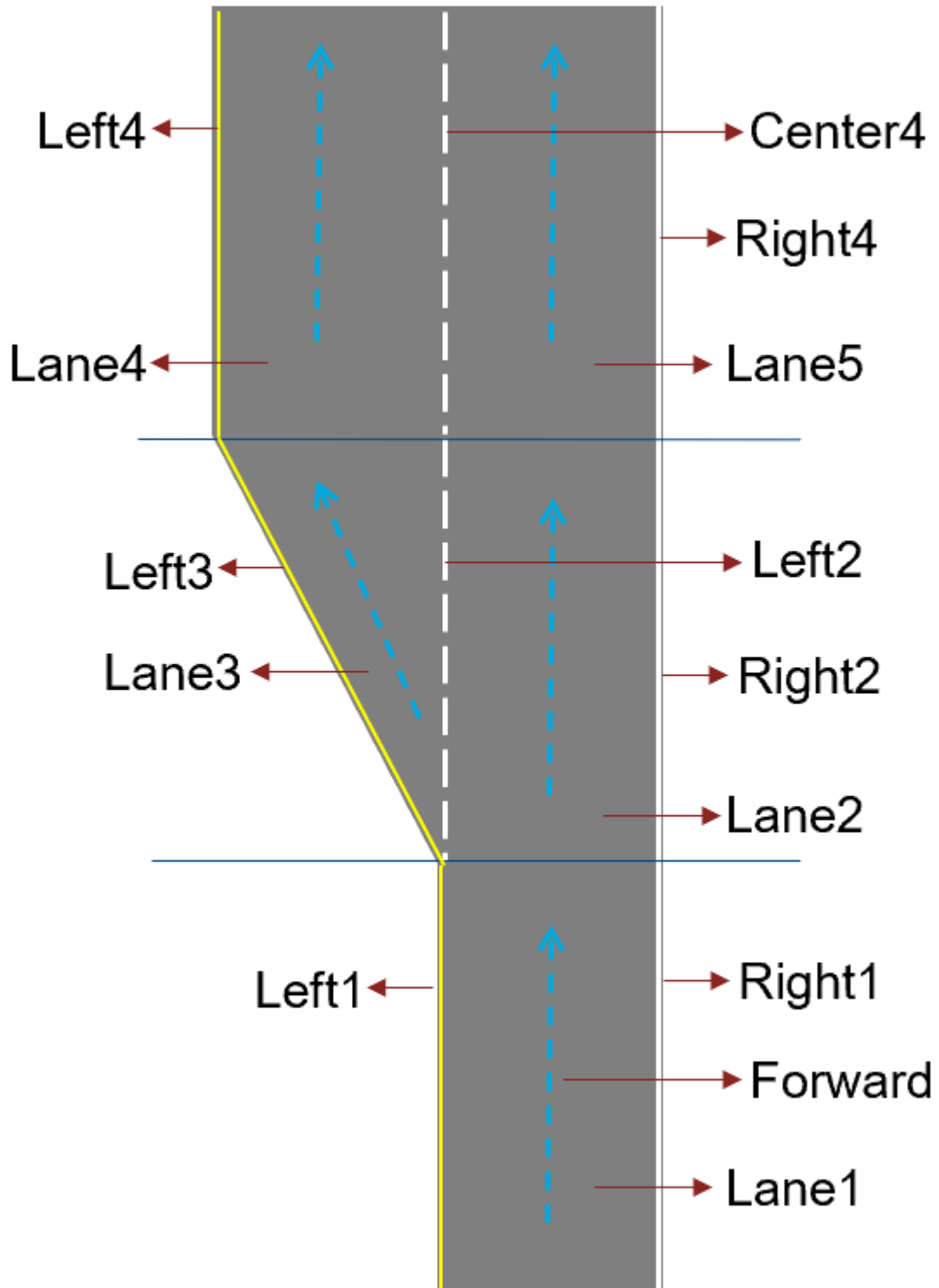
Create an empty RoadRunner HD Map as a `roadrunnerHDMap` object.

```
rrMap = roadrunnerHDMap;
```

Specify the lane and the lane boundaries.

```
rrMap.Lanes(5,1) = roadrunner.hdmap.Lane();
rrMap.LaneBoundaries(8,1) = roadrunner.hdmap.LaneBoundary();
```

Specify the lane groups and the lane markings.

```
rrMap.LaneGroups(3,1) = roadrunner.hdmap.LaneGroup();
rrMap.LaneMarkings(3,1) = roadrunner.hdmap.LaneMarking();
```

Assign the `Lane` property values. Split `Lane1` into `Lane4` and `Lane5,` and use `Lane2` and `Lane3` for the transition.

```
[rrMap.Lanes.ID] = deal("Lane1","Lane2","Lane3","Lane4","Lane5");
[rrMap.Lanes.Geometry] = deal([0 -20;0 0;0 20;], [0 20;0 40;0 60;], [0 20;-3 40;-6 60], [-6 60;-6
[rrMap.Lanes.TravelDirection] = deal("Forward");
[rrMap.Lanes.LaneType] = deal("Driving");
```

Assign the `LaneBoundaries` property values. `Lane3` shares its right boundary, `Left2,` with `Lane2`.

```
[rrMap.LaneBoundaries.ID] = deal("Left1","Right1","Left2","Right2","Left3","Left4","Center4","Rig
[rrMap.LaneBoundaries.Geometry] = deal([-3 -20;-3 0;-3 20],[3 -20;3 0;3 20],[-3 20;-3 40;-3 60;]
        [3 20;3 40;3 60],[-3 20;-6 40;-9 60],[-9 60;-9 80;-9 100],[-3 60;-3 80;-3 100],[3 60;3 80
```

Link the lane boundaries to the lanes. Define the left and the right lane boundaries for each lane, and specify alignment between the lanes and lane boundaries.

```
leftBoundary(rrMap.Lanes(1),"Left1",Alignment="Forward");
rightBoundary(rrMap.Lanes(1),"Right1",Alignment="Forward");
leftBoundary(rrMap.Lanes(2),"Left2",Alignment="Forward");
rightBoundary(rrMap.Lanes(2),"Right2",Alignment="Forward");
leftBoundary(rrMap.Lanes(3),"Left3",Alignment="Forward");
rightBoundary(rrMap.Lanes(3),"Left2",Alignment="Forward");
leftBoundary(rrMap.Lanes(4),"Left4",Alignment="Forward");
rightBoundary(rrMap.Lanes(4),"Center4",Alignment="Forward");
leftBoundary(rrMap.Lanes(5),"Center4",Alignment="Forward");
rightBoundary(rrMap.Lanes(5),"Right4",Alignment="Forward");
```

Specify the alignments between the lanes by defining their predecessor and successor relationships.

```
rrMap.Lanes(3).Successors = roadrunner.hdmap.AlignedReference(Reference=roadrunner.hdmap.Referenc
rrMap.Lanes(3).Predecessors = roadrunner.hdmap.AlignedReference(Reference=roadrunner.hdmap.Refere
rrMap.Lanes(2).Successors = roadrunner.hdmap.AlignedReference(Reference=roadrunner.hdmap.Referenc
rrMap.Lanes(2).Predecessors = roadrunner.hdmap.AlignedReference(Reference=roadrunner.hdmap.Refere
rrMap.Lanes(1).Successors = roadrunner.hdmap.AlignedReference(Reference=roadrunner.hdmap.Referenc
rrMap.Lanes(4).Predecessors = roadrunner.hdmap.AlignedReference(Reference=roadrunner.hdmap.Refere
rrMap.Lanes(5).Predecessors = roadrunner.hdmap.AlignedReference(Reference=roadrunner.hdmap.Refere
```

Add a dashed, white marking in addition to the solid white and yellow markings, to the HD map. Define the path to the dashed, white lane marking asset using the `roadrunner.hdmap.RelativeAssetPath` function.

```
dashedWhiteAsset = roadrunner.hdmap.RelativeAssetPath(AssetPath="Assets/Markings/DashedSingleWhi
```

Create a dashed, white lane marking on the road using the `roadrunner.hdmap.LaneMarking` object.

```
rrMap.LaneMarkings(3,1) = roadrunner.hdmap.LaneMarking();
[rrMap.LaneMarkings.ID] = deal("SolidWhite","SolidYellow","DashedWhite");
[rrMap.LaneMarkings.AssetPath] = deal(solidWhiteAsset,solidYellowAsset,dashedWhiteAsset);
```

Assign lane markings using the parametric attributes.

```
markingRefDW = roadrunner.hdmap.MarkingReference(MarkingID=roadrunner.hdmap.Reference(ID="DashedW
markingAttribDW = roadrunner.hdmap.ParametricAttribution(MarkingReference=markingRefDW,Span=marki
[rrMap.LaneBoundaries.ParametricAttributes] = deal(markingAttribSY,markingAttribSW,markingAttribD
    markingAttribSY,markingAttribSY,markingAttribDW,markingAttribSW);
```
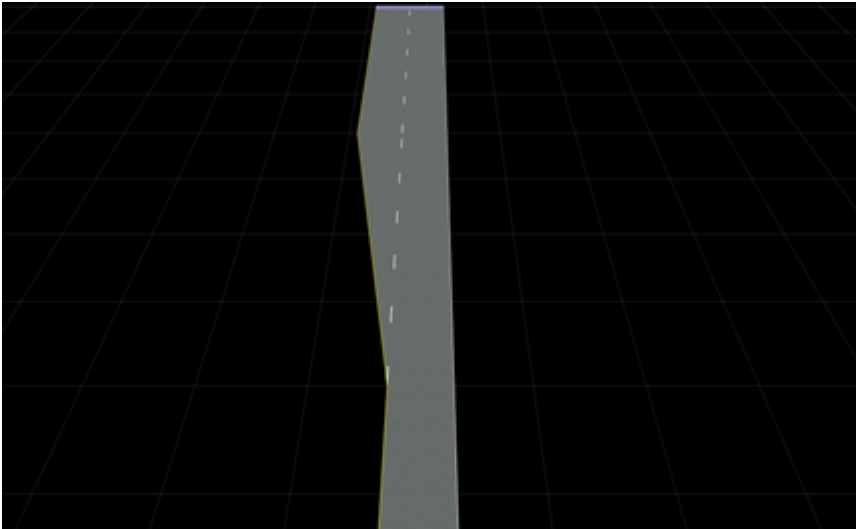
Plot the lane centers and lane boundaries.

```
plot(rrMap)
```
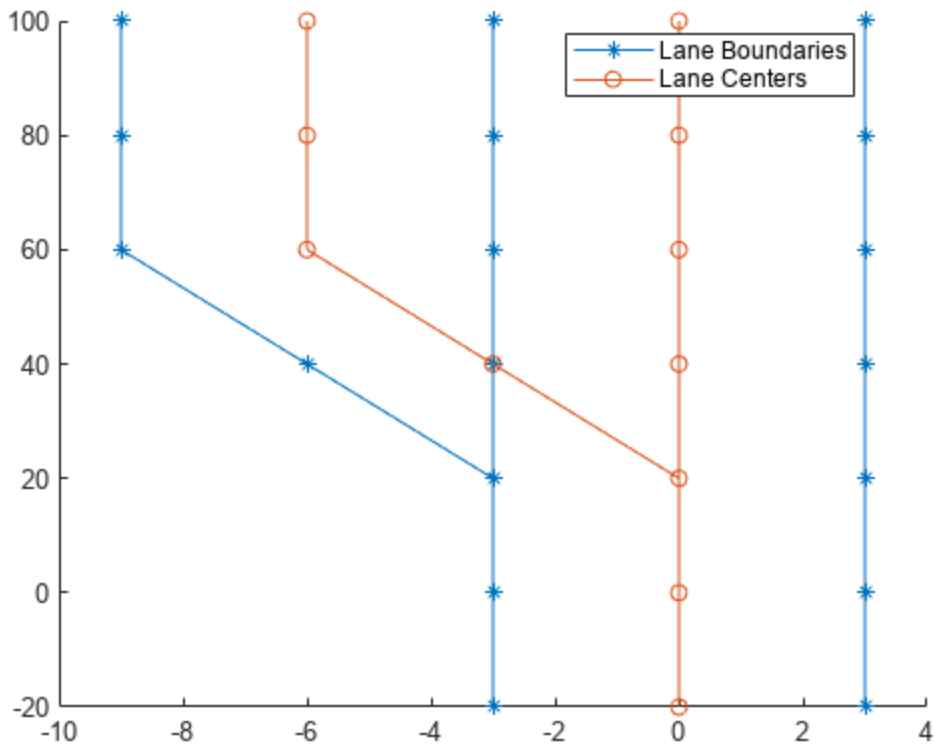

Write the HD map to a file.

```
write(rrMap,"laneAdd.rrhd")
```

This figure shows a scene of the one-way road with two lanes defined in this section, built using RoadRunner Scene Builder.

See Also
roadrunner | roadrunnerHDMap

Related Examples
- "Build Scenes from Custom Data Using RoadRunner HD Map"

RoadRunner Asset Library Product Overview

RoadRunner Asset Library Product Description

Populate RoadRunner scenes with a library of 3D models

RoadRunner Asset Library is a set of 3D models and assets for 3D scenes created with RoadRunner. The library provides hundreds of models, including road and highway signs, traffic signals, road surface markings, trees, barriers, and road damage textures, such as cracks and oil spills. All models are professionally designed and visually consistent.