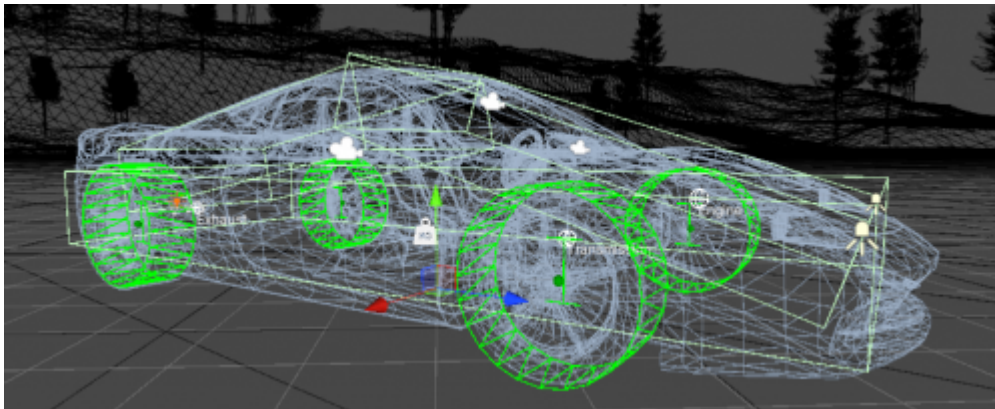


Vehicle Controller Manual

Before starting with detailed vehicle setup it is recommended to check out Quick Start guide as this page assumes a minimal working setup has been done.



One of the included, fully configured vehicle examples.

VehicleComponent

VehicleComponent is a building block of NWH Vehicle Physics 2.

VehicleController is a collection of VehicleComponents, which includes VehicleModules, Effects and SoundComponents - both of which inherit from VehicleComponent.

For class reference [click here](#).

States

Each VehicleComponent can:

- Be turned on or off.
- Be enabled or disabled.
- Have LOD set.

IsEnabled

- VehicleComponent that is enabled is updated.
- If the component will never be used isOn can be used instead.
- Disabled components are not initialized until they get enabled for the first time.



Enabled VehicleComponent.



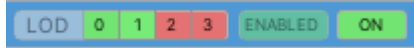
Disabled VehicleComponent.

LOD Index

- If the component's `lodIndex` value is lower than `vehicleController.activeLodIndex` it will be enabled, otherwise it will be disabled.
- Set `lodIndex` to a value less than 0 to ignore LODs.
- If adjusting LODs through inspector click on the currently active LOD to disable LOD system:



LOD system is inactive and the `VehicleComponent` will be active until manually disabled.



LOD system is active. `VehicleComponent` will be enabled while active LOD is 0 or 1 and disabled while active LOD is 2 or 3.

IsOn

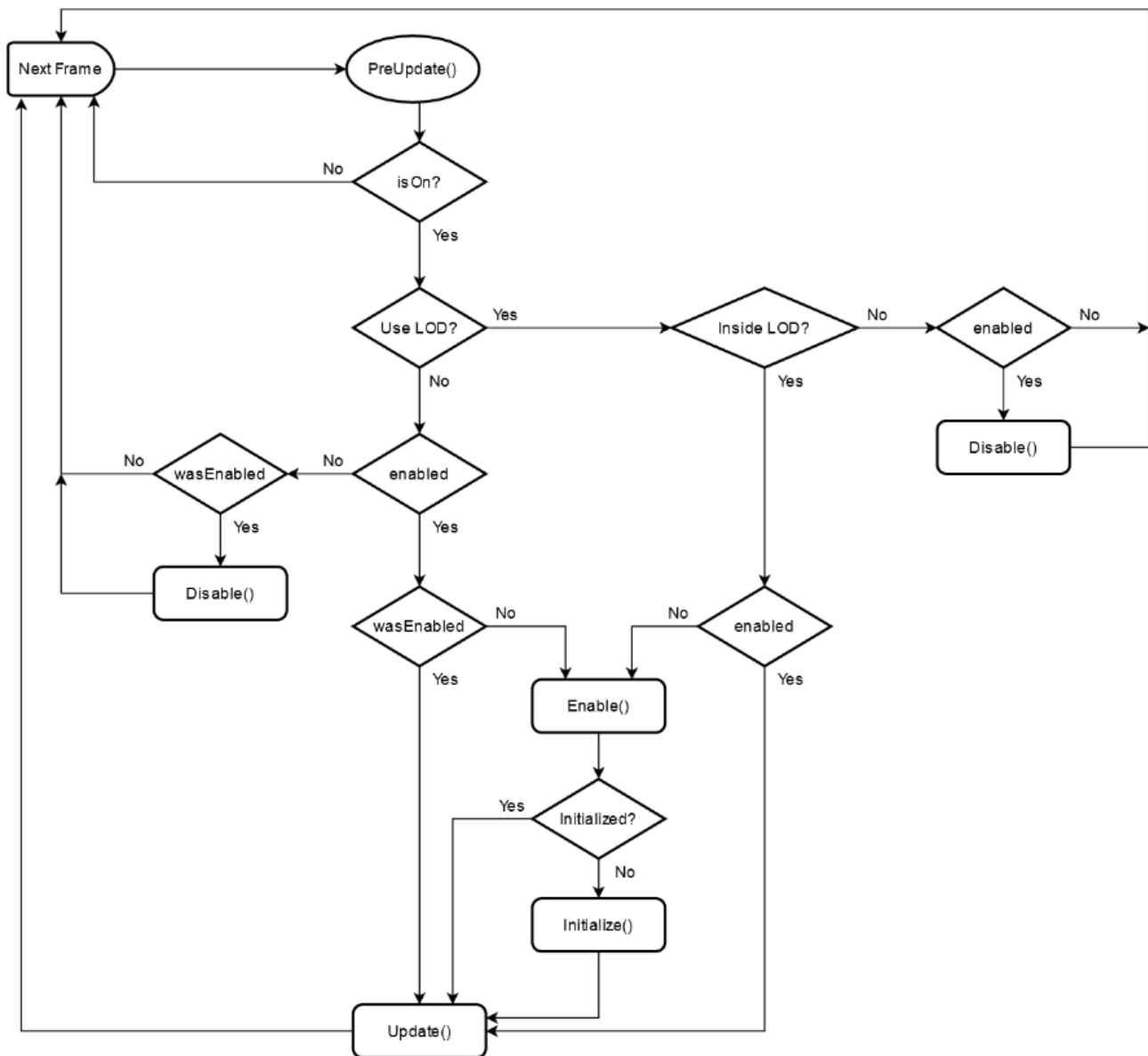
`VehicleComponents` that have `isOn` set to `false` are disabled and can not be enabled by LODs. Turning components off should be used for components that will never be used.



`VehicleComponent` turned off. Once it is off it can only be turned on manually. LODs are ignored when off.

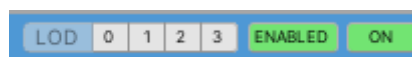
State Diagram

Diagram below demonstrates how a `VehicleComponent` determines if it should be updated:



State Bar

Each `VehicleComponent` has a state bar which is visible in play mode:

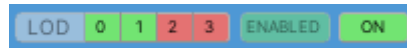


This is where the state of the `VehicleComponent` can be checked and changed. Note that changing the state through state bar only affects runtime values and will revert after exiting play mode. The state bar is intended for previewing the current state and testing different values during run-time. For persistent state check *StateSettings*.

Using State Bar

- Click on “On” button to turn the component on or off.
- Click on “Enabled” button to enable or disable the component.
- Click on any of the LOD numbers to set LOD. Click again on the same number to disable LODs. Component will be enabled if the current LOD number is green, and disabled if it is red. When LODs are active Enabled button is greyed out as the LODs determine if the component will be

enabled or disabled.



State Settings

- To prevent having to adjust the states of each `VehicleComponent` on each `VehicleController`, a `ScriptableObject` `StateSettings` was introduced.
- `StateSettings` for each vehicle can be assigned under `VehicleController` ⇒ `Settings` tab.
- More about `StateSettings` on [State Settings](#) page.

Scripting

To change `VehicleComponent` state through scripting following can be done:

```
myVehicleComponent.IsEnabled= myBoolValue;  
myVehicleComponent.IsOn = myBoolValue;  
myVehicleComponent.LodIndex = myIntValue;
```

2020/04/08 12:36

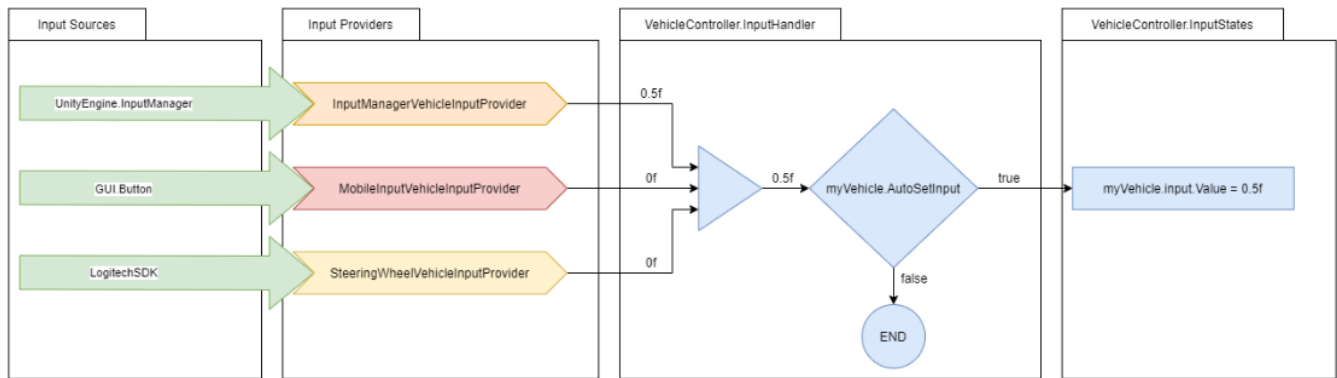
Input

Make sure to use *Project Settings* ⇒ *Player* ⇒ *Input Handling* ⇒ *Both* or multiple errors will pop up.

Input Concept

The input in NWH Vehicle Physics 2 centers around *InputProviders*. These are scripts that take user input (e.g. keypresses, mouse movement, gamepad input, etc.), process it and pass it on to the vehicles.

Input Retrieval



Process of retrieving input from different sources using InputProvider system.

The diagram above explains the path from input source to vehicle input state:

1. Input is retrieved from hardware through *Input Sources* (green). This can be input from any input method available for Unity. In this example InputManager (old/standard Unity input), touchscreen/sensors (mobile input) and LogitechSDK for steering wheel input.
2. *InputProviders* pass this input to the vehicle (VehicleController.InputHandler). Input from all input sources is combined and processed depending on settings for that particular vehicle.
3. If the Auto Set Input is set to false the state of the corresponding input for the vehicle in question will be set to the combined value of all inputs.
4. If the VehicleController has Input > AutoSetInput set to false the new input will be discarded. This happens in case the vehicle is inactive or the input is being set by some other script (e.g. AI).

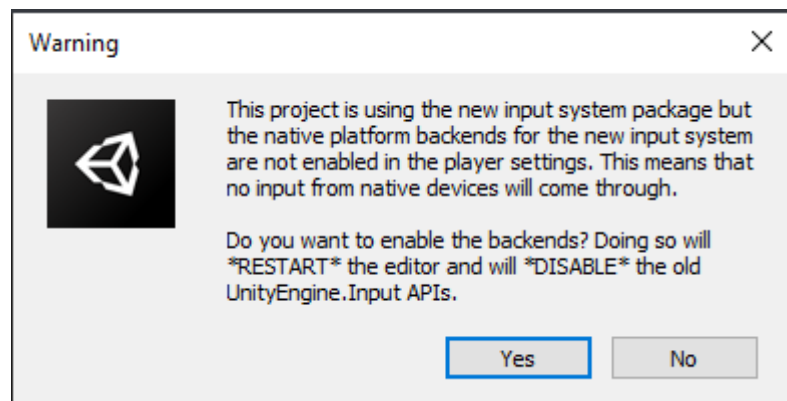
Notes

- A single *InputProvider* provides the input for all the vehicles so it is recommended to only have one *InputProvider* of particular type (e.g. one desktop and one mobile input provider - this will allow the vehicles to get both desktop and mobile input at the same time).
- Input providers just provide the input, they do not set it. Vehicles themselves get the input from input providers if AutoSetInput is enabled.
- All *InputProviders* inherit from either *VehicleInputProvider* (for vehicle-related input) or *SceneInputProvider* (for scene-related input such as cameras). Therefore it is best to think about *InputProviders* as a standardized interface between different input methods and a vehicle.
- *InputProviders* are split into *VehicleInputProviders* and *SceneInputProviders*. *VehicleInputProviders* relay vehicle input (throttle, brakes, etc.) while *SceneInputProviders* take care of scene input (vehicle changing, camera changing, camera movement and the rest of the inputs not directly related to vehicle. One of each needs to be present (e.g. *InputSystemVehicleProvider* and *InputSystemSceneInputProvider*).
- Multiple different *InputProviders* can be present in the scene (v1.0 or newer required). E.g. *InputSystemProviders* and *MobileInputProviders* can be used in the same scene. The resulting input will be a sum of inputs from all *InputProviders* in case of numeric inputs and logical OR operation of all inputs in case of boolean inputs.
- Input is stored inside *InputStates* struct and can be copied over from one vehicle to another. E.g. this is what is done when a trailer is connected to a towing vehicle.
- To manually set the *InputStates* make sure Auto Set Input is set to false.

All input providers inherit from either `VehicleInputProviderBase` or `SceneInputProviderBase`, but differ in their implementation. To create a new input provider simply inherit from one of those two classes and implement the members.

Input System Warning

When importing the asset for the first time this message will pop up:



Both Yes or No can be selected but it is important to set the *Project Settings* ⇒ *Player* ⇒ *Input Handling* to Both afterwards. This way both new `InputSystem` and the old `InputManager` will work. If this setting is set to `InputManager` only errors might appear as the demo scenes of the asset rely on `InputSystem`.

If a message *This Unity Package has Package Manager dependencies.* appears, click *Install/Upgrade*.

Available Bindings

Vehicle Input Provider Bindings

Out of the box gamepad bindings are only available for `InputSystem`.

Name	Type	Keyboard Defaults	Gamepad Defaults	Description
Steering	axis [-1,1]	A/D	Left Stick - Left/Right	Steering.
Throttle	axis [0,1]	W	Left Stick - Up, Right Trigger	Throttle.
Brakes	axis [0,1]	S	Left Stick - Down, Left Trigger	Brakes.
Clutch	axis [0,1]			Manual clutch. 0 for disengaged and 1 for engaged.
Handbrake	axis [0,1]	Space	B (Xbox) / Circle (PS)	
EngineStartStop	Button	E		

Name	Type	Keyboard Defaults	Gamepad Defaults	Description
ShiftUp	button	R	Right Shoulder	
ShiftDown	button	F	Left Shoulder	
ShiftIntoR1	button	`		Shift into 1st reverse gear.
ShiftIntoN	button	0		Shift into neutral.
ShiftInto1	button	1		Shift into 1st forward gear.
ShiftInto[n]	button	2,3,4,etc.		Shift into [n]th gear.
LowBeamLights	button	L	Y (Xbox) / Triangle (PS)	
HighBeamLights	button	K		
HazardLights	button	J		
ExtraLights	button	;		
LeftBlinker	button	Z		
RightBlinker	button	X		
Horn	button	H		
Module Bindings				
FlipOver	button	M		Used for FlipOverModule.
Boost	button	Left Shift	A (Xbox) / X (PS)	Used for NOSModule.
Cruise Control	button	N		Used for CruiseControlModule.
TrailerAttachDetach	button	T	X (Xbox) / Square (PS)	Used for Trailer and TrailerHitch modules.

Scene Input Provider Bindings

Name	Type	Keyboard Defaults	Gamepad Defaults	Description
ChangeCamera	button	C	Start	Changes camera.
CameraRotation	2D axis	Mouse Delta	Right Stick	Controls camera rotation.
CameraPanning	2D axis	Mouse Delta	Right Stick	Controls camera panning.
CameraRotationModifier	button	Mouse - LMB	Right Stick Press	Enables camera rotation.
CameraPanningModifier	button	Mouse - RMB	Left Stick Press	Enables camera panning.
CameraZoom	axis	Mouse - Scroll	D-Pad Up/Down	Camera zoom in/out.
ChangeVehicle	button	V	Select	Change vehicle or enter/exit vehicle.
FPSMovement	2D axis	WASD	Left Stick	Demo FPS controller movement.
ToggleGUI	button	Tab		Toggles demo scene GUI.

Input Manager (old/classic)

- Type of InputProvider for handling user input on desktop devices through keyboard and mouse or gamepad.
- Uses classic/old Unity Input Manager. It is recommended to use the Unity's new Input System instead for new projects.

Since v1.1 InputSystem package is required even if not used. If using the old/classic Unity input set

Project Settings ⇒ Player ⇒ Input Handling to Both and proceed as normal. InputSystem package being present installed will not interfere with old/classic Unity input / InputManager.

Installation

When first importing NWH Vehicle Physics 2 the project will be missing required bindings. There are two ways to add those:

1. Manually adding each entry to the *Project Settings ⇒ Input* following the input bindings table available [here](#).
2. Copying the contents of *InputBindings.txt* and appending them to the contents of the `[UnityProjectPath]/ProjectSettings/InputManager.asset` file. To do so:
 - Close Unity.
 - Open *InputManager.asset* in Notepad/Notepad++/Visual Studio or any other text editor of your choice.
 - Copy the contents of the provided *InputBindings.txt* file (*Scripts ⇒ Vehicle ⇒ Control ⇒ Input ⇒ InputProviders ⇒ InputManagerProvider ⇒ InputBindings.txt*) and paste them at the end of the *InputManager.asset*. Make sure there are no empty lines between the existing content and the pasted content. Save the file.
 - Open Unity. Check *Project Settings ⇒ Input*. The input bindings for NWH Vehicle Physics will appear towards the bottom of the list.

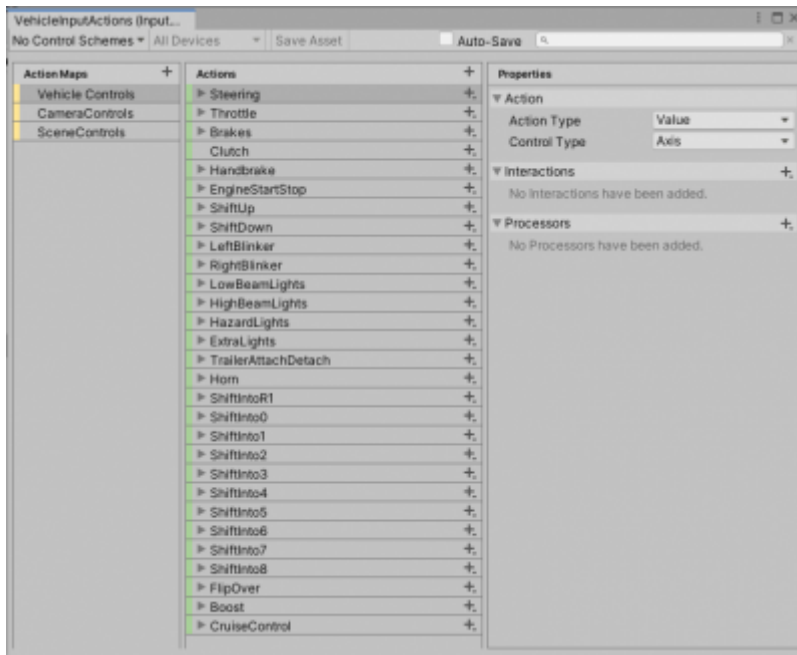
Scene Setup

To set up InputManager-based input in the scene add the following components to the scene:

1. 'InputManagerVehicleInputProvider'
2. 'InputManagerSceneInputProvider'

Any vehicle that is present in the scene will now receive input from these providers.

Input System (new)



Default InputActions.

- Since v1.1 NWH Vehicle Physics 2 has moved to InputSystem as a default input method.
- InputSystem v1.0 or higher is required. This is available in Unity 2019.3 or newer.

When using DS4Windows, InputSystem will detect button presses twice.

Installation

- Install 'Input System' package through Window ⇒ Package Manager
- Under Edit ⇒ Project Settings ⇒ Player ⇒ Other Settings ⇒ Active Input Handling select Input System Package (New) or Both - the latter in case your project still uses UnityEngine.Input somewhere.

Scene Setup

- Add InputSystemVehicleInputProvider and InputSystemSceneInputProvider to any object in your scene.
- Default bindings can be modified by double clicking on .inputactions files. Save Asset must be clicked for the changes to take effect.

Mobile Input Provider

- Add MobileVehicleInputProvider and MobileSceneInputProvider to the scene.
- Create a few UI ⇒ Button objects inside your canvas. Make sure that they are clickable.
- Remove the UnityEngine.UI.Button component and replace it with MobileInputButton. MobileInputButton inherits from UnityEngine.UI.Button and adds hasBeenClicked and isPressed fields which are required for Mobile Input Provider
- Drag the buttons to the corresponding fields in the MobileVehicleInputProvider and MobileSceneInputProvider inspectors. Empty fields will be ignored.

Steering Wheel Input Provider

For more info visit [SteeringWheelInputProvider](#) page.

Scripting

Retrieving Input

Since v1.0 multiple InputProviders can be present in the scene, meaning that their input has to be combined to get the final input result. To get the combined input use:

```
float throttle = InputProvider.CombinedInput(i => i.Throttle());  
bool engineStartStop = InputProvider.CombinedInput(i =>  
i.EngineStartStop());
```

Or to get the input from individual InputProviders (say to find out if a button was pressed on a keyboard): `float throttle = InputProvider.Instances[0].Throttle;` When using input generated by code (i.e. AI) it is usually handy to have access to a single axis throttle/brake. This can be done like so:

```
vehicleController.input.Vertical = 0.5f; //Sets throttle to 0.5f, resets  
brakes.  
vehicleController.input.Vertical = -0.5f; //Sets brakes to 0.5f, resets  
throttle.
```

vehicleController.input.states.throttle is equal to vehicleController.input.Throttle. The latter is just a getter/setter for convenience.

Manually Setting Input

Input in each vehicle is stored in InputStates struct:

```
myVehicleController.input.states
```

In case input should not be retrieved from user but from another script - as is the case when AI is used - AutoSettable should be set to false. This will disable automatic input fetching from the active InputProvider.

Input now can be set from any script:

```
myVehicleController.input.Horizontal = myFloatValue; // Using getter/setter.
```

```
myVehicleController.input.states.horizontal = myFloatValue; // Directly  
accessing states.
```

Custom InputProvider

If a custom `InputProvider` is needed it can easily be written. Custom `InputProviders` allow for new input methods or for modifying the existing ones. E.g. if the `MobileInputProvider` does not fit the needs of the project a copy of it can be made and modifications done on that copy. That way it will not get overwritten when the asset is updated.

Steps to create a new `InputProvider`:

- Create a new class, e.g. `ExampleInputProvider` and make it inherit from `InputProvider` class:

```
public class ExampleInputProvider : InputProvider {}
```

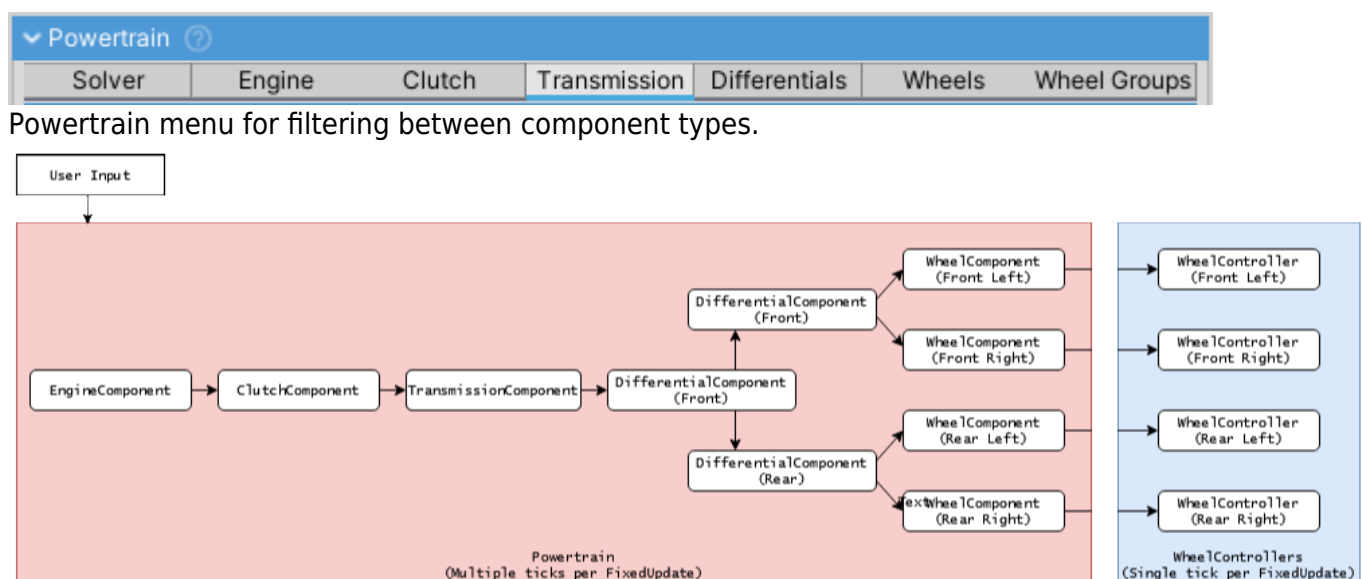
- Implement missing methods. Most IDEs can do this automatically.
- The required methods are *abstract* and will need to be implemented. There are also *virtual* methods such as `ToggleGUI()` which are optional and will be ignored if not implemented.
- Methods that are not used should return `false`, `0` or `-999` in case of `ShiftInto()` method.

2020/03/14 11:38 · Aron Rescec

Powertrain

NWH Vehicle Physics 2 uses different solver from NWH Vehicle Physics 1. Solver in NWH Vehicle Physics uses multiple ticks per one `FixedUpdate` and is physically accurate. The only component that is approximated is the clutch and that is due to performance reasons.

Powertrain in NWH Vehicle Physics 2 is a collection of *Powertrain Components* such as `EngineComponent`, `ClutchComponent`, `DifferentialComponent`, etc.



Example of typical Powertrain setup for 4-wheel, 4WD vehicle.

- Each component outputs to one or more `PowertrainComponents`, except for `WheelComponent` which always outputs to `WheelController`.
- `EngineComponent` which acts as a power source and `WheelComponent` acts as a power sink.

The components in-between determine how the power/torque will be transmitted.

Solver

The only setting available for solver is **Physics Quality**. **Physics Quality** changes the number of ticks per one FixedUpdate: * **Low** - 8 ticks * **Medium** - 12 ticks * **High** - 16 ticks * **Very High** - 24 ticks * **Ultra** - 32 ticks * **Overkill** - 48 ticks * Performance impact is directly proportional to the number of ticks. Powertrain code is heavily optimized. * Using very low values (<8) can result in low simulation fidelity and/or instability, especially if used with high power vehicles, high gear ratios or complex differential setups.

Since v1.4 physics quality can be adjusted under *Settings* tab by changing the number of physics substeps.

2020/04/08 12:36

PowertrainComponent

PowertrainComponent is a base class for all powertrain components: EngineComponent, ClutchComponent, TransmissionComponent, etc.

- All the PowertrainComponents have the following common fields in the inspector:
 - **Name** - name of the component. Changing the Name of a component will reset Output on any components that are using that component.
 - **Inertia** - inertia of the component. Inertia of each component contributes to the total system inertia. How much depends on the clutch engagement and current gear ratio.
 - **Output** - Powertrain Component to which the torque is forwarded. In cases such as Differential Component there can be multiple outputs (e.g. left and right wheel).



Common PowertrainComponent fields.

- Changing the name field on component will reset all the Outputs on other PowertrainComponents that use that component as an output.
- Increasing Inertia will make the component spin up slower if the same torque is applied.
- Inertia of WheelComponent is calculated from WheelControllers mass and radius settings.

Inertia must always be larger than 0!

2020/04/08 12:36

Engine

Engine

Common Properties

Name

Engine

Inertia

0.12

Output To:

[Clutch] Clutch

General

Engine Type

ICE

Stalling Enabled

☒

Stall RPM

300

rpm

Ignition

☒

Auto Start On Throttle

☒

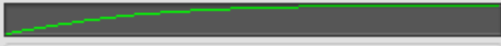
Power & Torque

Max Power

260

kW

Power Curve



Power Modifier Sum

0

x100 %

Starter

Starter Active

☐

Starter Run Time

1

s

Starter RPM Limit

600

rpm

Starter Torque

60

Nm

Idler Circuit

Idler Circuit Enabled

☒

Idle RPM

1200

Idler Circuit Sensitivity

0.2

Throttle

Throttle Smoothing

0

Rev Limiter

Rev Limiter Enabled

☒

Rev Limiter Active

☐

Rev Limiter RPM

8700

Rev Limiter Cutoff Duration

0.01

- Power Gain Multiplier adds power on top of the existing Max Power so the vehicle with 100kW and Power Gain Multiplier of 1.5 will actually produce 150kW.
- Boost value affects sound components TurboWhistleComponent and TurboFlutterComponent. If forced induction is to be used just for the sound effects Power Gain Multiplier should be set to 1.

2020/04/08 12:36

Power Modifiers

Power modifiers can be used through scripting to modify the power of the engine. These are functions that return a float which denotes an engine power coefficient. Example:

```
public float AddBoost()
{
    if(boostIsActive)
    {
        return 1.5f; // Increases power for 50%.
    }
}
...
myVehicleController.powertrain.engine.powerModifiers.Add(AddBoost);
```

This is a fictional example. A concrete example can be found inside TCS module which uses this mechanic to limit power when there is wheel spin.

2020/04/08 12:36

Clutch

The image shows a software inspector for a 'Clutch' component. It is organized into several sections: 'Common Properties' with fields for Name, Inertia, and Output To; 'General' with sliders for Clutch Engagement and Slip Torque; 'Automatic Clutch' with checkboxes and RPM values; 'PID Controller' with sliders for Kp, Ki, and Kd; and 'Torque Converter' with a checkbox and a slip torque value.

Section	Property	Value
Common Properties	Name	Clutch
	Inertia	0.02
	Output To:	[Transmission] Transmission
General	Clutch Engagement	0
	Slip Torque	3000 Nm
Automatic Clutch	Is Automatic	<input checked="" type="checkbox"/>
	Base Engagement RPM	2300
	Variable Engagement RPM Range	1000
	Final Engagement RPM	2000
PID Controller	PID_Kp	1.18
	PID_Ki	2.32
	PID_Kd	2.25
Torque Converter	Has Torque Converter	<input checked="" type="checkbox"/>
	Torque Converter Slip Torque	120

ClutchComponent inspector.

ClutchComponent is a mandatory Powertrain component. It is always second in the Components list.

- ClutchComponent can be bypassed by setting the output of EngineComponent directly to the desired PowertrainComponent but this is not recommended as it will cause stalling in most cases.

Manual Control

- Untick Is Automatic to use manual clutch.
- Clutch can be controlled through Clutch axis - check [Input](#) section for more info on setting up axes.

PID Controller

PID controller is used to control Clutch Engagement when Is Automatic is true.

- More about PID controllers [here](#).
- In general default clutch settings should be adequate for most setups.
- PID_Coefficient can be adjusted to slow down or speed up clutch engagement.

Slip Torque

- When Has Torque Converter is false Slip Torque is used. Otherwise, clutch will use Torque Converter Slip Torque. \\]
- Slip torque for a normal clutch should be a bit higher than the maximum engine torque. Usually a few hundred to a few thousand Nm.
- Too high Slip Torque will result in grabby clutch.
- Using too high Slip Torque values can result in torque spikes when clutch is suddenly released which can impact solver stability in extreme cases.
- Values near zero will result in engine spinning up as if the clutch is not engaged due to clutch slip.

2020/04/08 12:36

Transmission

Transmission ?

Common Properties

Name

Transmission

Inertia

0.02

Output To:

[Differential] CenterDifferential

General

Transmission Type

Automatic Sequential

Reverse Type

Auto

Gearing

Gearing Profile

SportsCarGearingProfile (TransmissionGearingProfile ?)

Transmission Gearing Profile ?

! This is a ScriptableObject. All changes are global.

Gear Ratios

Final Gear Ratio

5.2

Forward Gears

Element 0

3.25

Element 1

2.23

Element 2

1.66

Element 3

1.29

Element 4

1.03

Element 5

0.84

Element 6

0.69

+

-

Reverse Gears

Element 0

-3.79

+

-

Shifting

Upshift RPM

4900

rpm

Downshift RPM

2600

rpm

Shift Duration

0.05

s

Post Shift Ban

0.1

s

Current Gear Index

0

Variable Shift Point

☒

Variable Shift Intensity

0.4

Incline Effect Coeff

0.5

Target Upshift RPM

0

rpm

Target Downshift RPM

0

rpm

Shift Conditions

Shift Check Cooldown

0.1

No Wheel Spin

☐

No Wheel Skid

☐

No Wheel Air

☐

Clutch Engaged

☐

External Shift Checks Valid

☒

Events

On Shift (Int32)

List is Empty

+

-

On Upshift (Int32)

List is Empty

+

-

On Downshift (Int32)

List is Empty

+

-

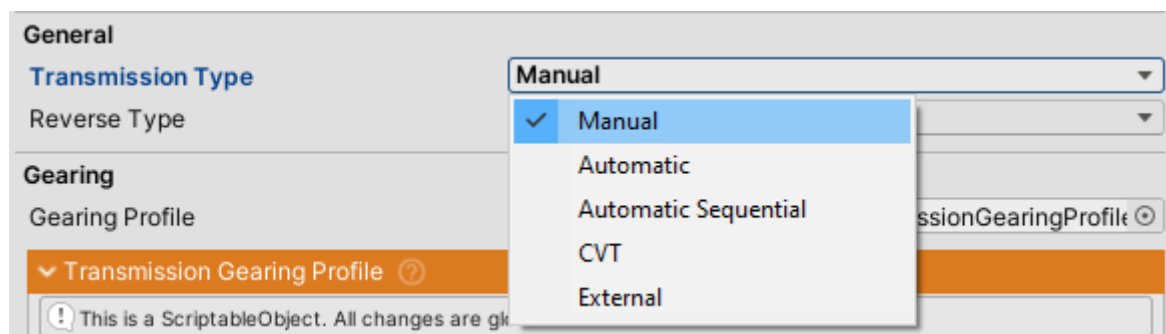
TransmissionComponent inspector.

- `TransmissionComponent` is a mandatory `Powertrain` component. It is always third in the `Powertrain.Components` list.
- NWH Vehicle Physics uses gear ratios – just like the real transmission does.
- If gears are not set up properly the vehicle will not move.
- There are a few included `TransmissionGearingProfiles` which can be used as an example.

Gearing

- Gearing is assigned through `TransmissionGearingProfile` `ScriptableObject`.
- Gear ratios can be adjusted during run-time.
- For more info check [Transmission Gearing Profile page](#).

Transmission Types



Manual

- Changing gears can only be done through user input. Check [Input Setup](#) for more info on input bindings.

Automatic

- Vehicle shifts gears based on the gear ratios and `Upshift RPM`, `Downshift RPM`, `Variable Shift Intensity` and `Incline Effect Coeff` variables. Gear skipping is enabled (e.g. it is possible that the vehicle will shift from 1st to 3rd if conditions are right).
- You can check the current `Target Upshift RPM` and `Target Downshift RPM` under the *Shifting* section of the `TransmissionComponent` inspector. These values vary depending on the variables mentioned above.

Vehicle under full throttle:

Target Upshift RPM	8280.042	rpm
Target Downshift RPM	4968.025	rpm

Same vehicle with 0 throttle:

Target Upshift RPM	4917.088	rpm
Target Downshift RPM	2617.089	rpm

Automatic Sequential

Same as Automatic but without gear skipping.

CVT

- CVT (and eCVT) transmissions have variable gearing ratio dependent on load.
- Gear ratio will be interpolated between first and second element in *Forward Gears* list based on input torque

and Cvt Max Input Torque field value.

Example transmission gearing profile for CVT transmission:

Transmission Gearing Profile ?

! This is a ScriptableObject. All changes are global.

Gear Ratios

Final Gear Ratio

Forward Gears

— Element 0

— Element 1

+ -

Reverse Gears

— Element 0

+ -

External

- Shift delegate is used for changing gears.
- This allows for external shifting logic.
- If the delegate is not assigned this option will result in no gear shifts.

Timing

To make shifting more realistic two timers have been added:

- Shift Duration - time Transmission takes to change from one gear to another. During this time EngineComponent's throttle is cut off. Works for all transmission types that shift gears.
- Post Shift Ban timer. This field determines minimum time between two shifts. Used to prevent transmission for shifting too often. Only affects automatic transmission types.

Shift Conditions

Transmission will only shift in automatic mode if ALL of the following conditions are met and after they have been met for Shift Check Cooldown seconds:

- No Wheel Spin - longitudinal slip on all wheels is less than Longitudinal Slip Threshold(*Settings tab*)
- No Wheel Skid- lateral slip on all wheels is less than Lateral Slip Threshold(*Settings tab*)
- No Wheel Air- none of the wheels are in the air.
- Clutch Engaged- clutch is fully engaged.
- External Shifts Checks Valid- list of ShiftCheckdelegates. All external shift checks must be valid for transmission to be able to shift.

You can check the state of shift conditions in the inspector:

Shift Conditions	
Shift Check Cooldown	0.1
No Wheel Spin	<input type="checkbox"/>
No Wheel Skid	<input type="checkbox"/>
No Wheel Air	<input type="checkbox"/>
Clutch Engaged	<input type="checkbox"/>
External Shift Checks Valid	<input checked="" type="checkbox"/>

Transmission Gearing Profile

Transmission Gearing Profile ?

This is a ScriptableObject. All changes are global.

Gear Ratios

Final Gear Ratio: 4.38

Forward Gears

= Element 0	3.08
= Element 1	2.18
= Element 2	1.63
= Element 3	1.29
= Element 4	1.03
= Element 5	0.84
= Element 6	0.69

+ -

Reverse Gears

= Element 0	-3.79
-------------	-------

+ -

TransmissionGearingProfile is a ScriptableObject that determines the gear ratios for the [TransmissionComponent](#).

- Forward Gears list contains all forward gear ratios in order from 1st gear up. In the example

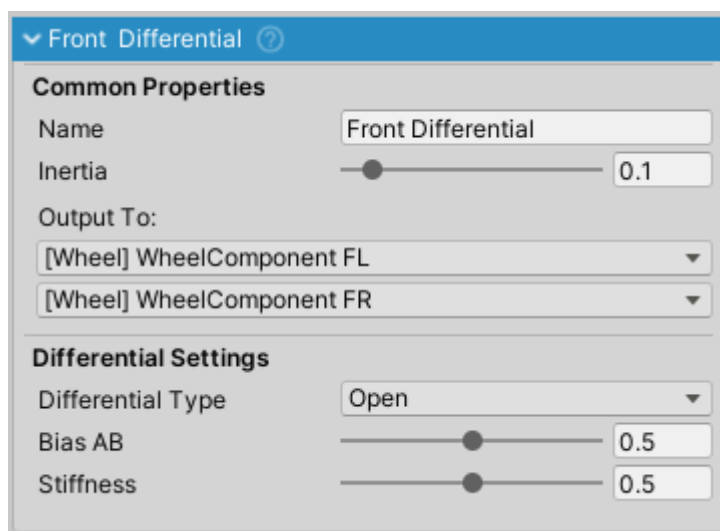
above:

- 1st gear = 3.25 ratio
- 2nd gear = 2.23 ratio
- 3rd gear = 1.66 ratio
- ...
- **Reverse Gears** list contains all reverse gear ratios. Multiple reverse gears can be added, e.g.
 - 1st reverse gear = -3.79 ratio
 - 2nd reverse gear = -2.61 ratio
 - ...
- **Final Gear Ratio** is the coefficient by which all the gears are multiplied. This is very similar to differential gear ratio and can be used to tune the gearing without having to adjust the gearing between individual ratios.

2020/04/08 12:36

2020/04/08 12:36

Differentials



DifferentialComponent inspector.

DifferentialComponent is a type of PowertrainComponent that splits input torque between two or more outputs.

There can be multiple DifferentialComponents present on one vehicle and one differential can output to other differentials which is useful for 4WD setup with center differential.

Differential Types

Open

Torque in open differential is equally split between the left output and right output.

Locked

Locked differential keeps both outputs rotating at same angular velocity.

Viscous LSD

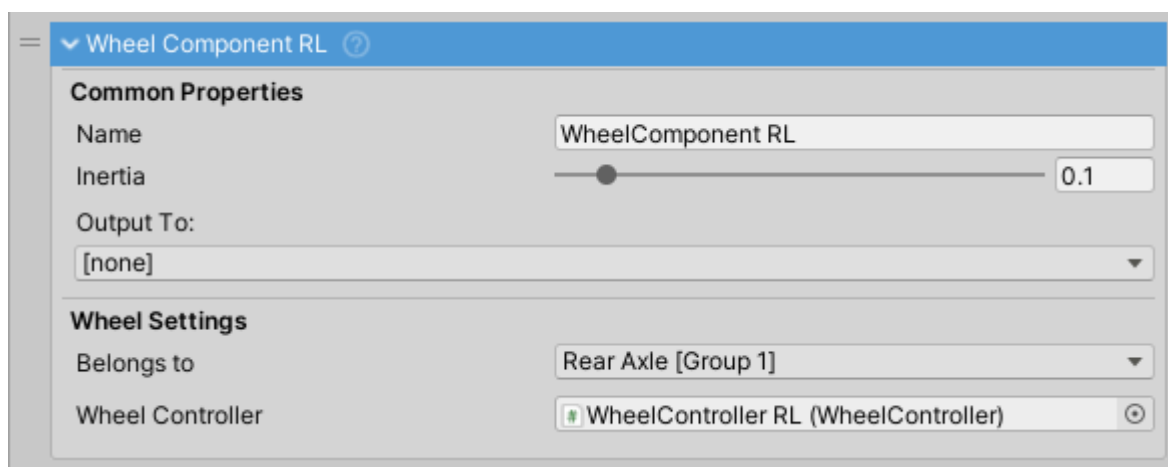
A type of limited slip differential (LSD). Torque is split based on difference in speed between the two outputs.

Clutch LSD

Also known as cone-type or plate limited slip differential (LSD). Torque is split based on speed difference between the two inputs and the input torque.

2020/04/08 12:36

WheelComponents



WheelComponent inspector.

- WheelComponent is a PowertrainComponent. It acts as a torque sink and can not output to another PowertrainComponent
- WheelComponent should not be mixed up with WheelController which is a replacement for Unity's WheelCollider.
- Belongs To field determines to which [WheelGroup](#) the WheelComponent belongs to. This will determine values such as braking and steering coefficients and geometry.
- Inertia field gets auto-calculated from assigned WheelController's mass and radius.

2020/04/08 12:36

Wheel Groups / Axles

Front Axle

General

Name

Front Axle

Steering

Steer Coefficient0

Ackerman Percent0.15

Brakes

Brake Coefficient1

Handbrake Coefficient1

Geometry

Toe Angle0 deg

Caster Angle0 deg

Camber At Top-5 deg

Camber At Bottom1 deg

Axle

Anti Roll Bar Force

0

Nm

Is Solid

!

Fields 'Anti Roll Bar Force' and 'Axle Is Solid' will only work if wheel group has two wheels - a left and a right one.

WheelGroup inspector.

Steering

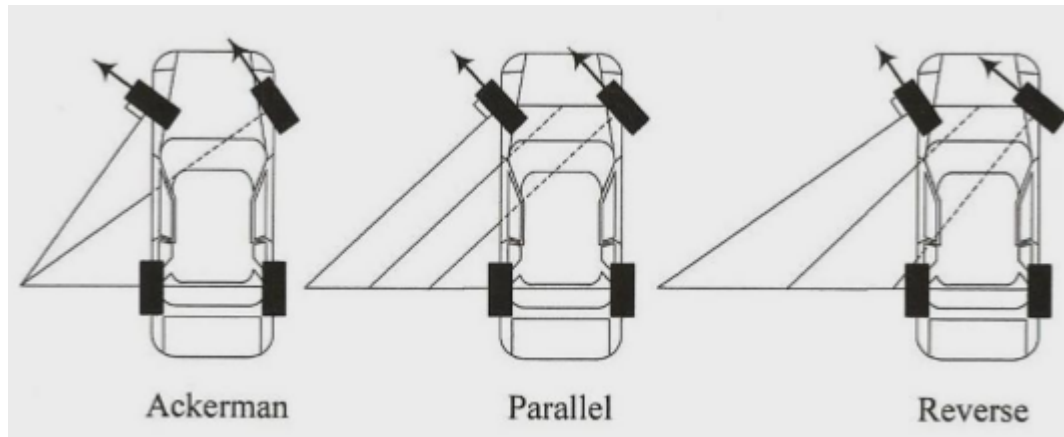
- Steer Coefficient determines how much the wheel will steer depending on input. In general cars would have Steer Coefficient of 1 in front and 0 in the back, except for four wheel steering cars where rear axle usually steers opposite of the front so the value would be negative. Examples:
 - 1 - 100% steering.
 - 0 - no steering.
 - 0.5 - 50% steering in the opposite direction.



Vehicle with Steer Coefficient of 1 on the front axle and -0.5 on the rear axle. Steering wheel

turned fully to the right.

- **Ackerman Percent** - check this [Wikipedia link](#) for more info about Ackerman Steering setup. Set to <0 for Reverse or Anti Ackermann or >0 for Ackermann steering. Field represents percent where 0.12 equals 12% of the steer angle. Following image describes the effect:



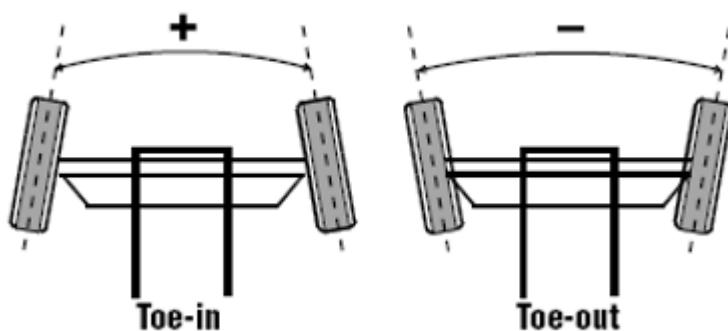
Effect of Ackerman setting on steering.

Brakes

- **Brake Coefficient** - amount of brake torque used as a percentage of Brakes \Rightarrow Max Torque.
- **Handbrake Coefficient** - amount of brake torque applied when handbrake is activated.

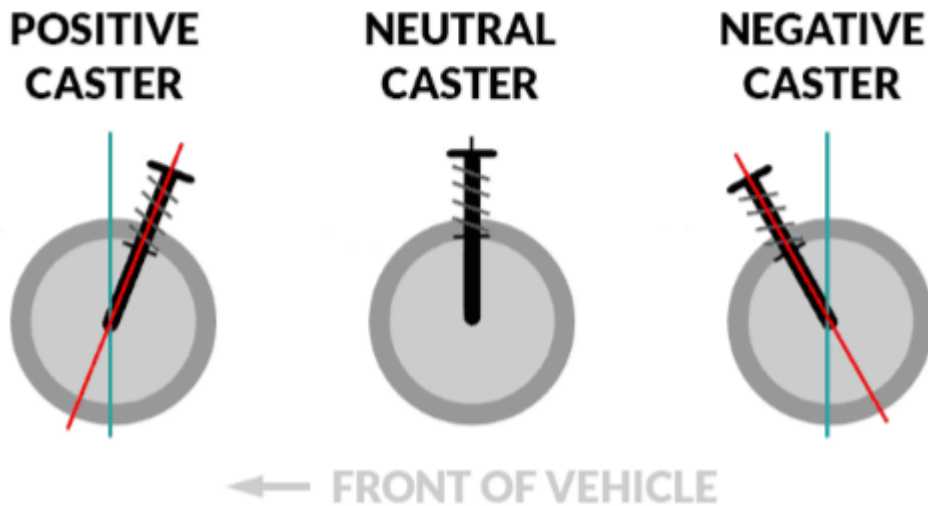
Geometry

- **Toe Angle** - toe angle in degrees.



Toe angle. Positive on the left (toe-in) and negative on the right (toe-out).

- **Caster Angle** - caster angle in degrees.



ImageCaption

- **Camber At Top and Camber At Bottom** - camber angle in degrees at the top (fully compressed) and the bottom (fully relaxed) of suspension travel. Current value will be an interpolated result between the two.

Axle

Axle settings are used only if there are exactly two wheels in the `WheelGroup`.

- **Anti Roll Bar Force** - force that imitates anti-roll bar in a vehicle. Before increasing value it is best to make sure that the center of mass of the vehicle is correct as too high center of mass can result in unstable vehicle. Too high values can introduce jitter to the vehicle as the ARB will try to equalize the suspension travel of both wheels
- **Is Solid** - Imitates solid axle and auto-adjusts camber to make sure that both wheels always stay parallel to each other.

2020/04/08 12:37

2020/04/08 12:36

Sound

Sound

Sound system in NVP2 consists of `SoundManager` and multiple `SoundComponents`. Disabling `SoundManager` also disables all the `SoundComponents`.

- `SoundComponent` and `SoundManager` are `VehicleComponents`. Check [VehicleComponent](#) page for more info.
- Each `SoundComponent` (type of `Vehicle Component`) is responsible for one sound, e.g. `EngineRunningComponent` or `EngineStartingComponent`.
- `AudioSources` are not added manually but are instead generated by the script when scene is started. Some `SoundComponents` can have more than one `AudioSource` - e.g. wheel related `SoundComponents` have one `AudioSource` for each wheel.

- Each field affects only the vehicle to which the script is attached. To modify audio output for all the vehicles `VehicleAudioMixer` (`VehicleAudioMixer.mixer`) can be used.

Requirements

- Mixer field must have `AudioMixer` assigned. By default `VehicleAudioMixer` will be used.

SoundComponent

`SoundComponent` inherits from `VehicleComponent`.

Check [VehicleComponent](#) page for more info.

2020/04/08 12:36

SoundManager

▼ Sound Manager ?
LOD 0 1 2 3 ENABLED

Master Settings

Master Volume 0.42

Spatial Blend 1

Mixer 🔊 VehicleAudioMixer ⌵

Interior Settings

Inside Vehicle ☐

Interior Attenuation dB

Low Pass Frequency Hz

Low Pass Q 1

Positioning

Engine Source Position X Y Z

Transmission Source Pos X Y Z

Exhaust Source Position X Y Z

Other Sources Position X Y Z

Enable gizmos and make sure the vehicle is selected in editor to be able to see the positions.

Components

Engine	Forced Ind.	Transmission	Suspension
Ground	Collision	Brakes	Blinkers
Horn		Reverse Beep	

▼ Crash Component ?
LOD 0 1 2 3 ENABLED

Base Volume 1

Base Pitch 1

Clips

- = Element 0 🎵 Crash ⌵
- = Element 1 🎵 CrashWallHit ⌵

+ -

If more than clip is supplied for sound components that require only one, a random clip will be selected each time sound is played.

SoundManager inspector.

SoundManager is the main class for handling sound. It contains all the global sound settings and also manages individual **SoundComponents**.

Also check [Sound](#) page.

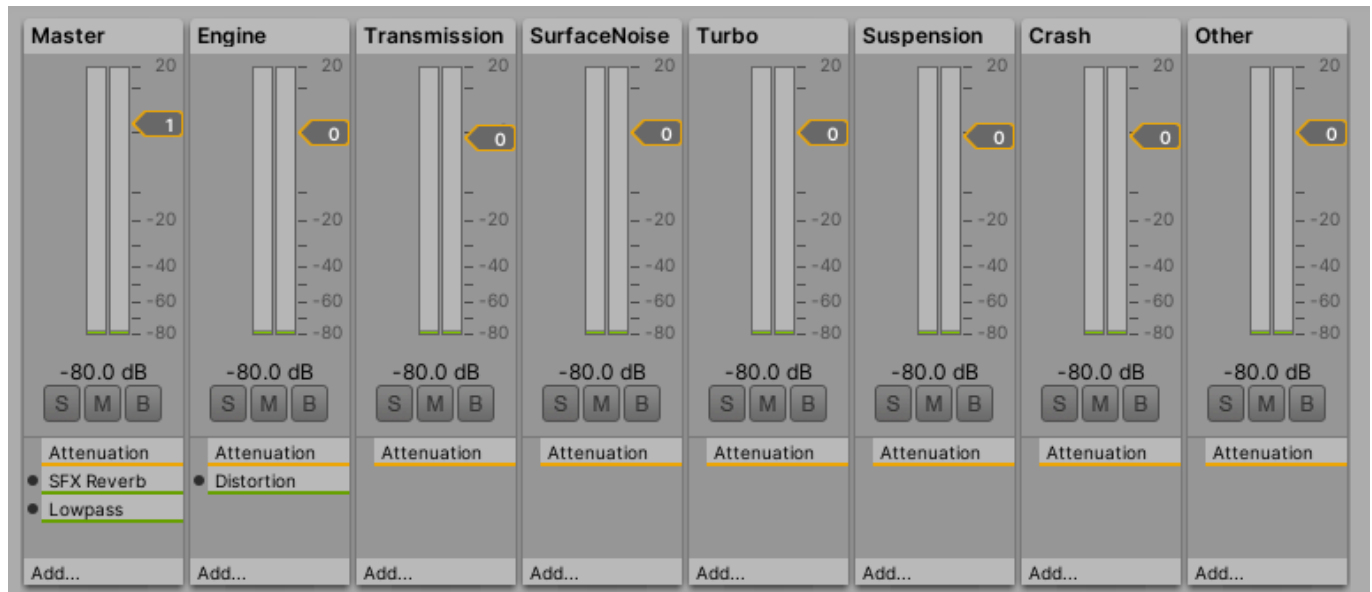
Settings

Master Settings

Master settings affect all the SoundComponents

Equalizer

Each sound component belongs to one of the Audio Mixer groups: Engine, Transmission, SurfaceNoise, Turbo, Suspension, Crash or Other. Here you can add additional effects and modify the sound. One example of this is distortion which is added to the engine sound based on load through the Audio Mixer.

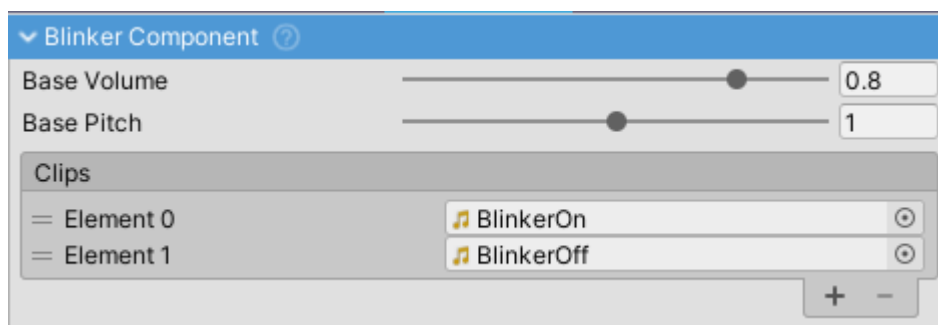


VehicleAudioMixer

2020/04/08 12:36

Available Sound Components

Blinker Component

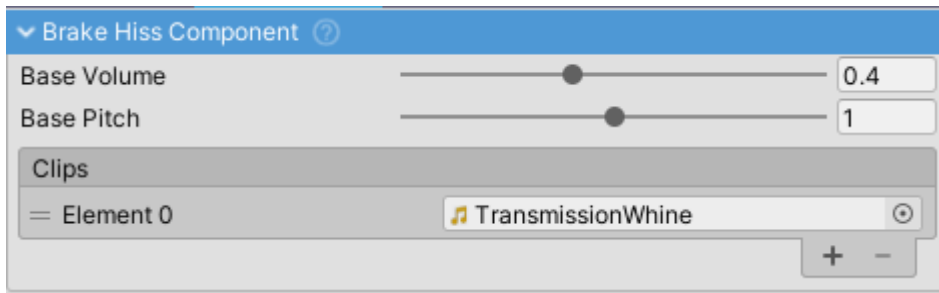


BlinkerComponent inspector.

- Click-clack of the working blinker.
- Accepts two clips, first is for the blinker turning on and the second is for blinker turning off.

2020/04/08 12:37

Brake Hiss

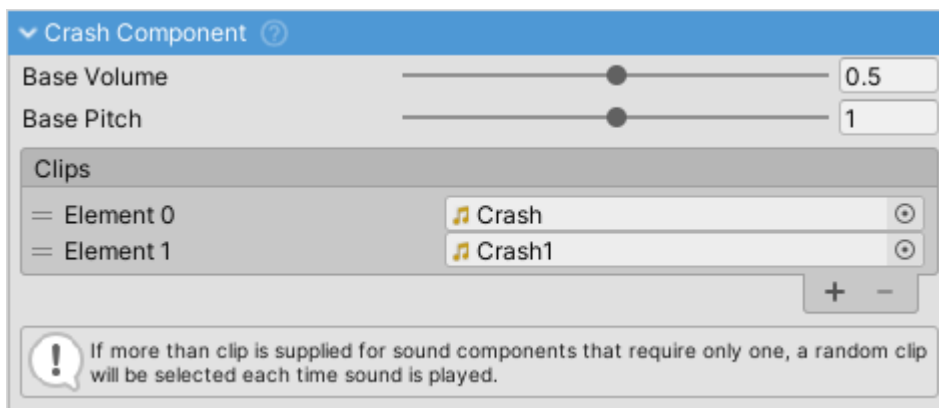


BrakeHissComponent inspector.

- Imitates brake hiss on vehicles with pneumatic brake systems such as trucks and buses.

2020/04/08 12:37

Crash

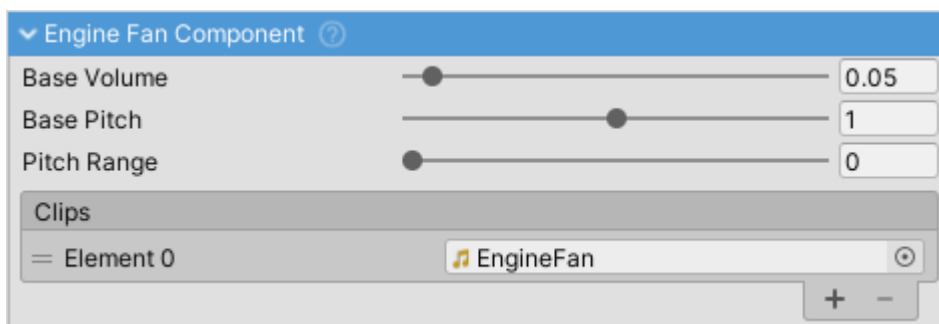


CrashComponent inspector.

- Sound of vehicle crashing.
- Supports multiple audio clips of which one will be chosen at random each time this effect is played.
- Volume is dependent on collision intensity. This can be adjusted through Velocity Magnitude Effect.
- Pitch is random. Randomness can be adjusted through Pitch Randomness field.

2020/04/08 12:37

Engine Fan



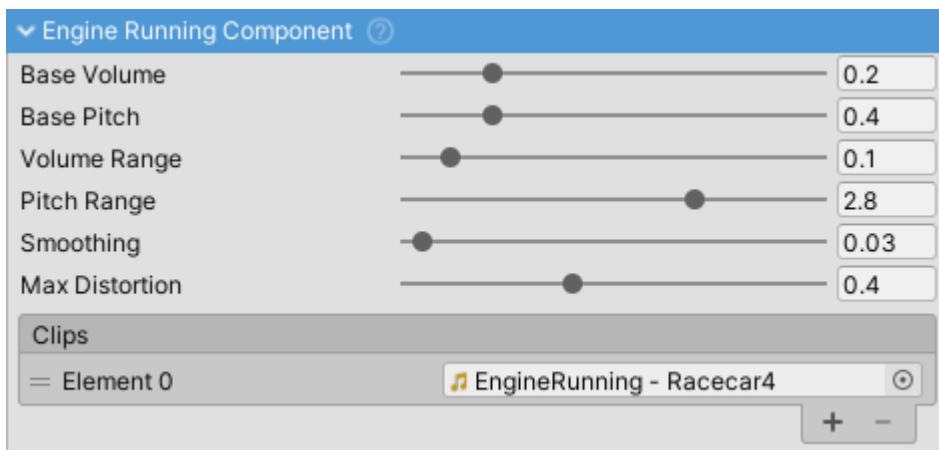
EngineFanComponent inspector.

EngineFanComponent is used to imitate engine fan running, the sound especially prominent in commercial vehicles and off-road vehicles with clutch driven fan.

- AudioSource of EngineFanComponent is positioned at Engine Position (*Settings* tab).

2020/04/08 12:37

Engine Running



EngineRunningComponent inspector.

EngineRunningComponent is a SoundComponent responsible for the engine sound. A pitch based approach using a single pre-recorded engine sound clip is used along with filters to achieve relatively realistic sound.

Big upside of this approach is that only a single clip is required which is ideal for small game studios, as opposed to layering approach where usually tens of clips of engine at different loads, positions and RPMs are used to achieve the engine sound effect. Of course, the downside is that realism suffers.

If you need the layering feature please make a feature request through support.

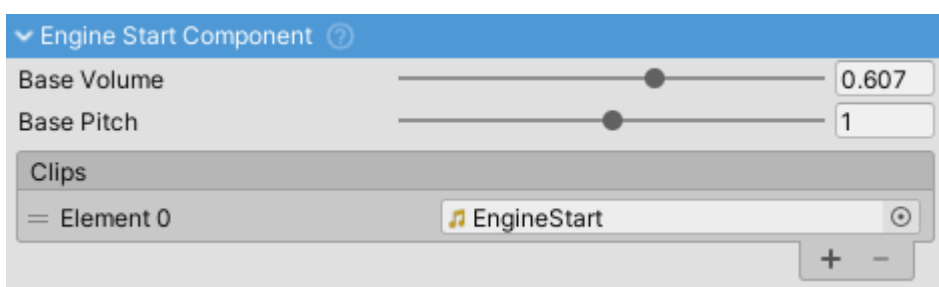
- AudioSource of EngineRunningComponent is positioned at Exhaust Position (*Settings* tab).

Notes

Distortion affects volume so when high levels of distortion are used it is usually a good idea to reduce volume range proportionately.

2020/04/08 12:37

Engine Start



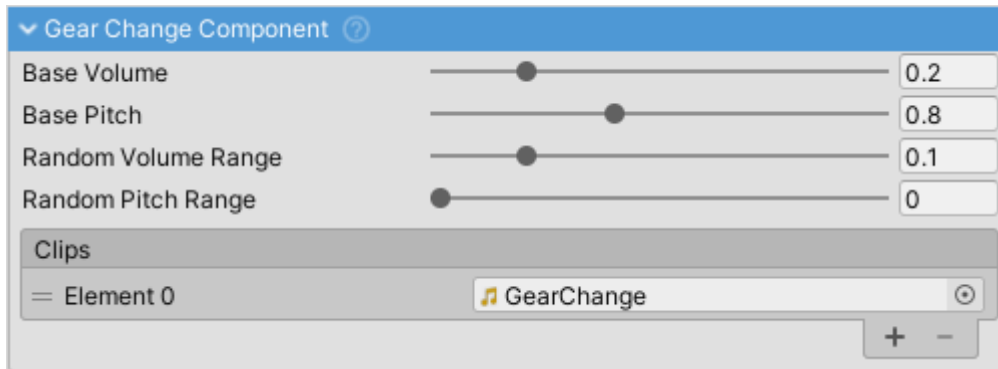
EngineStartComponent inspector.

- EngineStartComponent plays when starter is active.
- Starter settings can be adjusted under Powertrain ⇒ Engine tab.
- AudioSource of EngineStartComponent will be positioned at Settings ⇒ Engine

Position.

2020/04/08 12:37

Gear Change



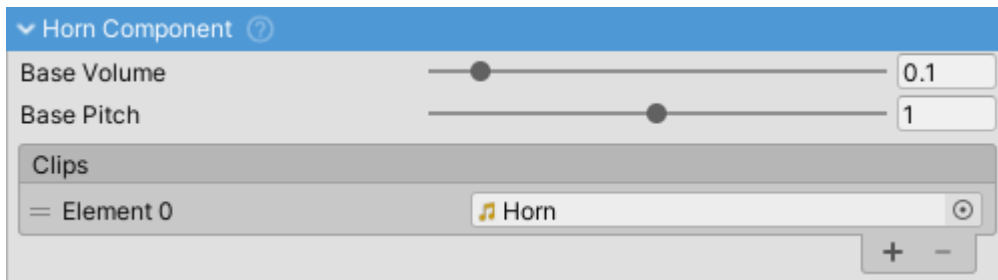
GearChangeComponent inspector.

Sound of changing gears.

- Supports multiple audio clips of which one is chosen at random each time the effect is played.
- AudioSource of GearChangeComponent is positioned at Settings ⇒ Transmission Position.

2020/04/08 12:37

Horn

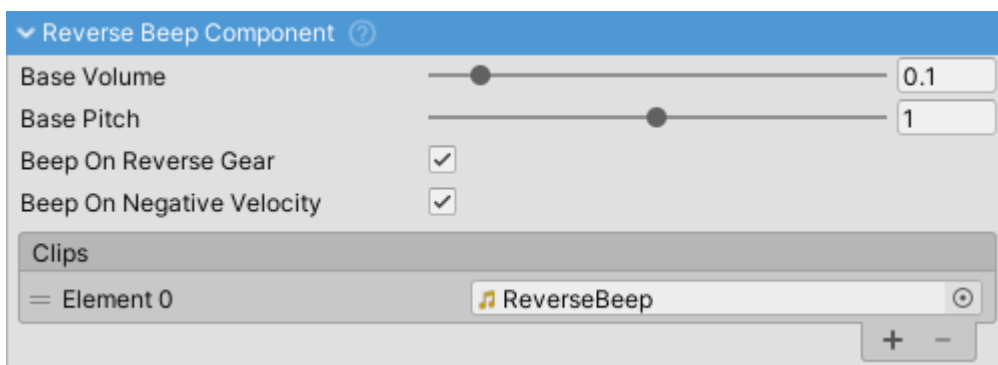


HornComponent inspector.

- Vehicle horn sound.

2020/04/08 12:37

Reverse Beep

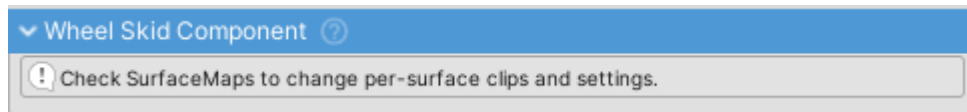


ReverseBeepComponent inspector.

- Beeping sound commercial vehicles make when driving in reverse.

2020/04/08 12:37

Skid

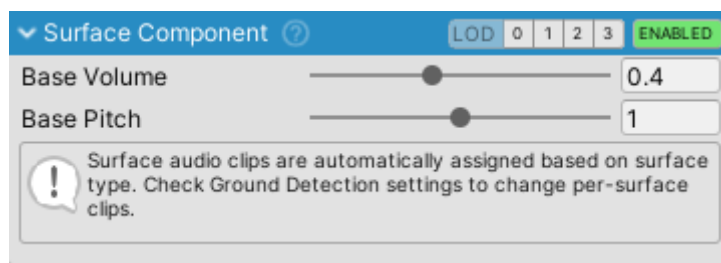


SkidComponent inspector.

- Sound produced by tire skidding/slipping over a surface.
- SkidComponent gets clips, volume and pitch settings from active SurfaceMap. TODO

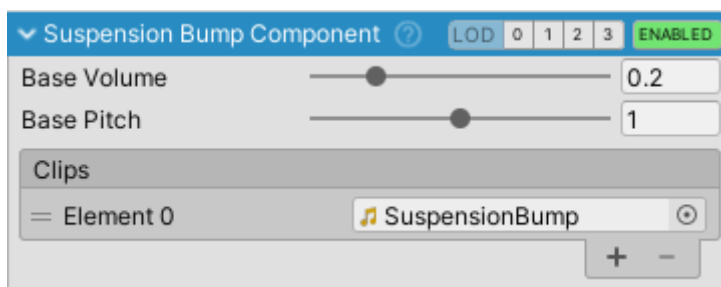
2020/04/08 12:37

Surface



2020/04/08 12:37

Suspension Bump

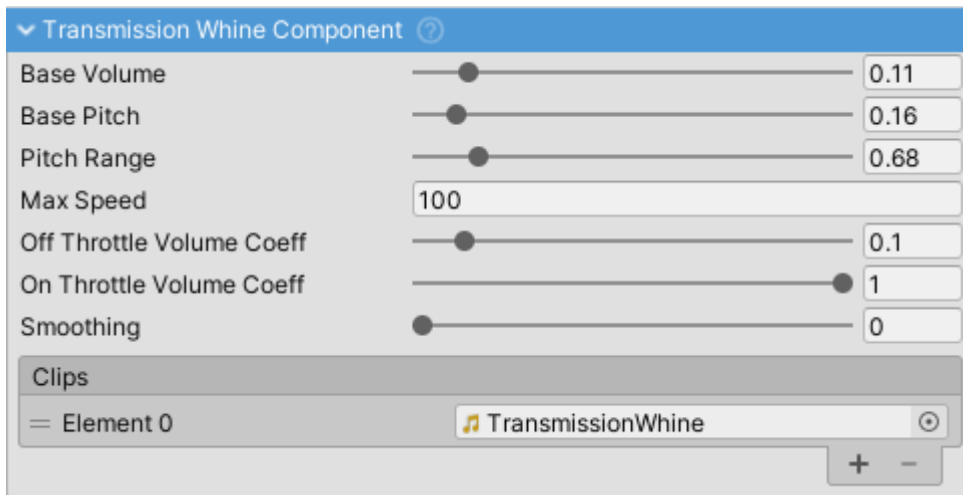


Sound of tire going over obstacle.

Each time the clip is played volume and pitch are changed to reduce repetitiveness. Multiple clips can also be used for the same reason.

2020/04/08 12:37

Transmission Whine



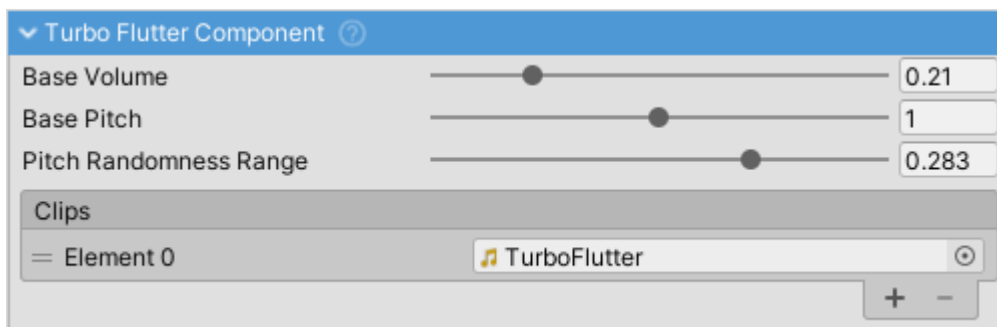
TransmissionWhineComponent inspector.

Sound of vehicle transmission. Most prominent on rally and racing cars with straight cut gears in the gearbox.

- AudioSource of TransmissionWhineComponent is positioned at Settings ⇒ Transmission Position.

2020/04/08 12:37

Turbo Flutter



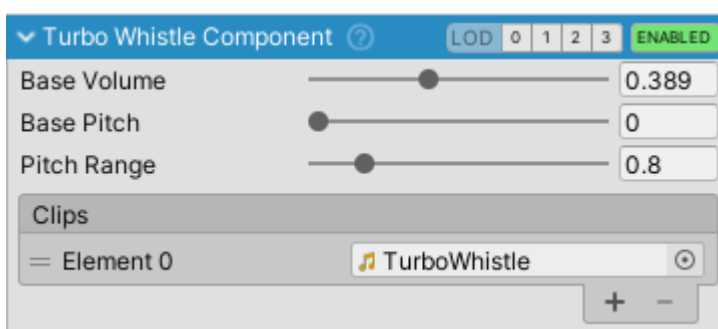
TurboFlutterComponent inspector.

Sound of a wastegate releasing air on turbocharged vehicles.

- Gets triggered after releasing throttle if there is adequate boost build up in ForcedInduction.
- AudioSource of TurboFlutterComponent is positioned at Settings ⇒ Engine Position.

2020/04/08 12:37

Turbo Whistle



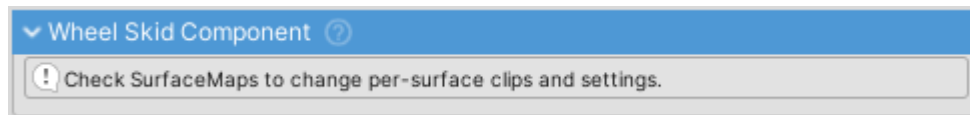
TurboWhistleComponent inspector.

Imitates high-pitched sound of forced induction.

- Can be used for both turbocharger and supercharger sound.
- Sound depends on [EngineComponent.ForcedInduction](#).
- AudioSource of TurboWhistleComponent is positioned at Settings ⇒ Engine Position.

2020/04/08 12:37

Wheel Skid

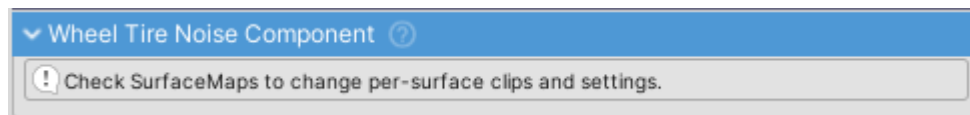


WheelSkidComponentInspector

- Sound produced by slipping/skidding over a surface.
- WheelSkidComponent gets clips, volume and pitch settings from active SurfaceMap. TODO
- An AudioSource gets generated for each WheelComponent.

2020/04/08 12:37

Wheel Tire Noise



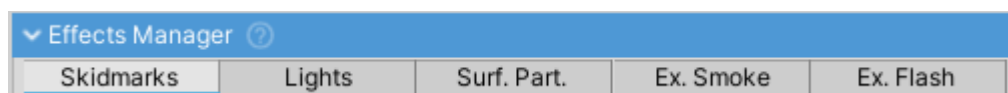
WheelTireNoiseComponent

- Sound produced by tire rolling over a surface.
- WheelTireNoiseComponent gets clips, volume and pitch settings from active SurfaceMap. TODO
- An AudioSource gets generated for each WheelComponent.

2020/04/08 12:37

2020/04/08 12:36

Effects

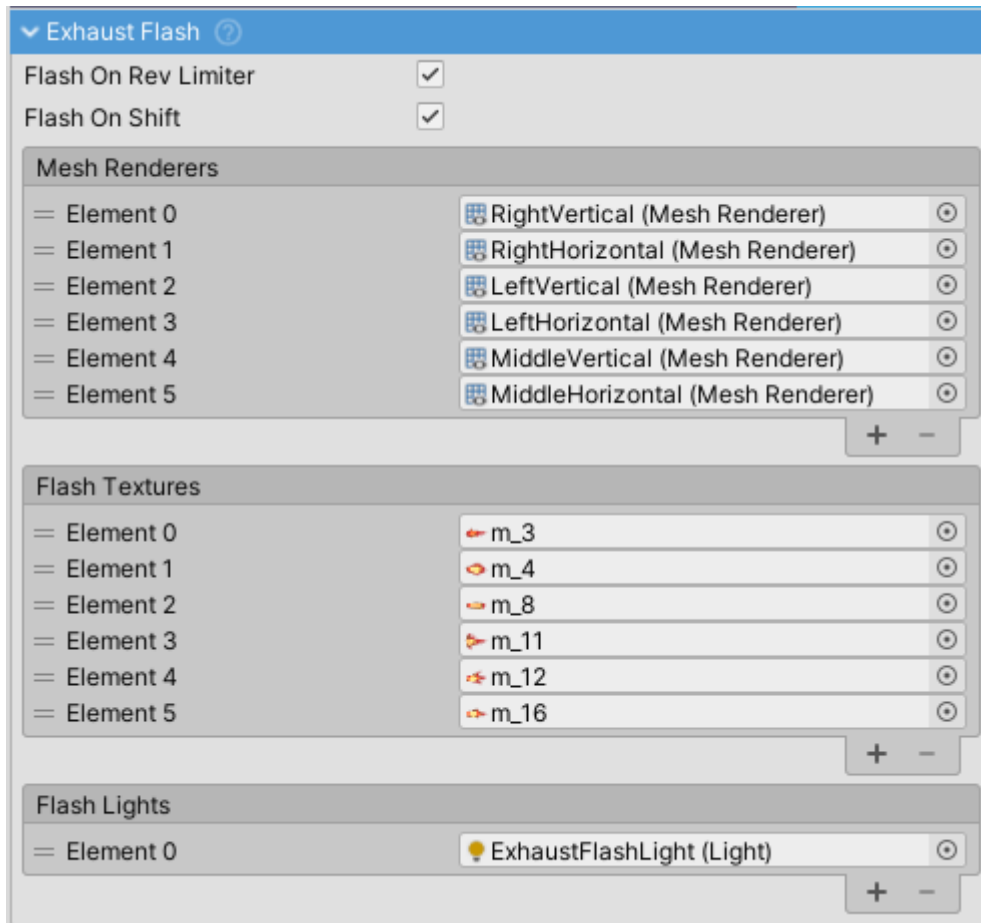


EffectsManager menu.

Effect system makes use of [VehicleComponents](#) and therefore each Effect can be turned on or off, be enabled or disabled or have LOD set.

- All Effects with *manager* in the name manage multiple instances of the effect, usually one for each wheel - e.g. skidmarks or surface particles.

Exhaust Flash



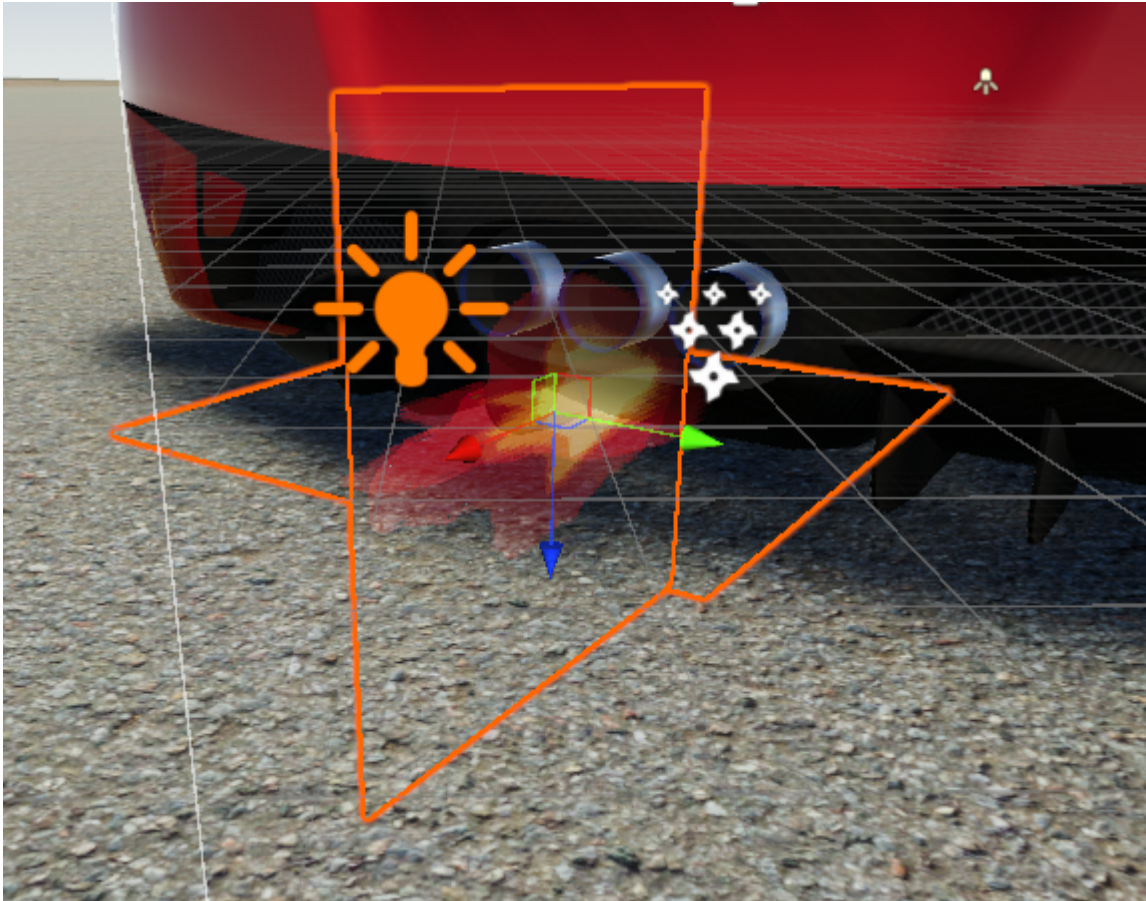
ExhaustFlash inspector.

ExhaustFlash Effect is used to imitate flames shooting out of exhaust. The method to achieve this is identical to the one used for most muzzle flash in FPS games; that is a set of images gets enabled and disabled at rapid rate with different sprite and scale each time. This is a performant way to achieve the effect while avoiding particle effects.

Setup

Adding Quads

- Create and place two Quads perpendicular to each other. Move them to the location of the exhaust.
- Remove any colliders from Quads.
- Create a new material that uses Particles/Standard Unlit (Unity 2019) or equivalent shader with Transparent rendering and Multiply color mode. The included example on *Sports Car* prefab can be copied. Assigning one of the included flame textures will show if the rotation of the Quad is correct. Rotate Quad if needed.



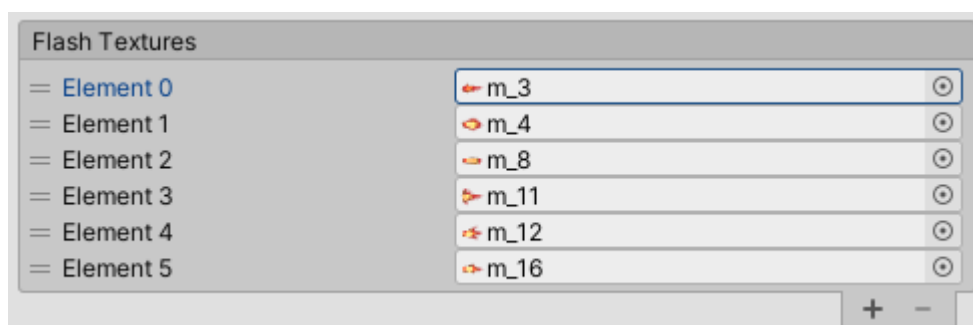
Two perpendicular quads with flame sprites.

- Assign the MeshRenderers from the newly created Quads to the ExhaustFlash ⇒ Mesh Renderers list.

Assigning Textures

To make flames look more convincing a random texture is assigned to each Quad on each flash. A number of default textures is included.

- Assign textures to Flash Textures list.

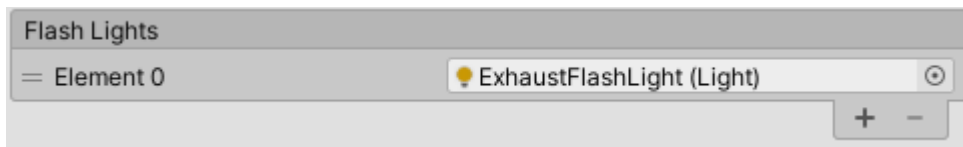


Flash Textures list.

Adding Point Lights

If the material used for exhaust flash is not emissive additional PointLight(s) can be added to light the surrounding area on flash.

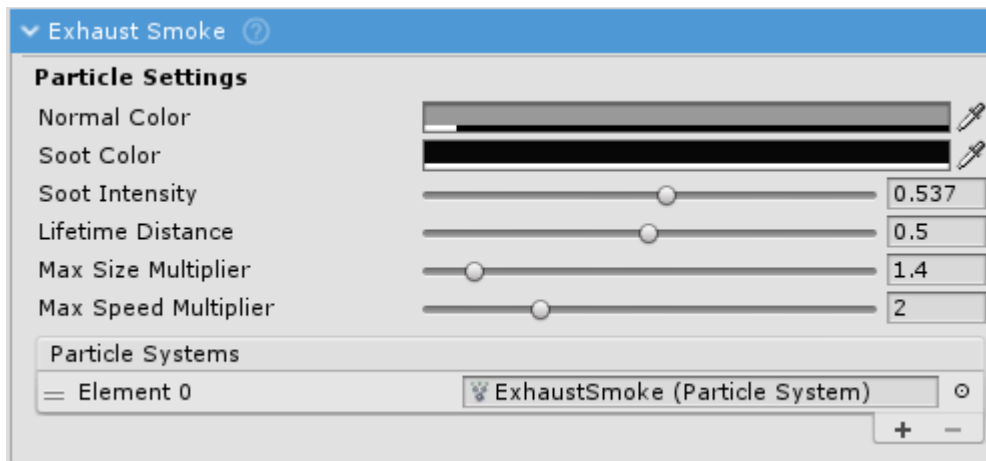
- Set up a `PointLight` and place it at the exhaust location.
- After configuring, disable the light.
- Add the light to the `Flash Lights` list.



Flash Lights list.

2020/04/08 12:36

Exhaust Smoke



ExhaustSmoke inspector.

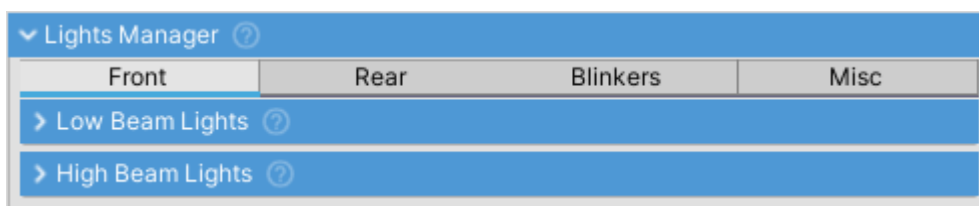
ExhaustSmoke controls exhaust `ParticleSystem` color, size and emission speed. It interpolates between `NormalColor` and `SootColor` based on engine state.

Setup

- Position `ParticleSystems` at vehicle exhaust position. Prefab of pre-configured `ParticleSystem` comes with the asset (*Effects > Particles > Prefabs*).
- Assign the `ParticleSystems` to the `Particle Systems` list inside ExhaustSmoke inspector.

2020/04/08 12:36

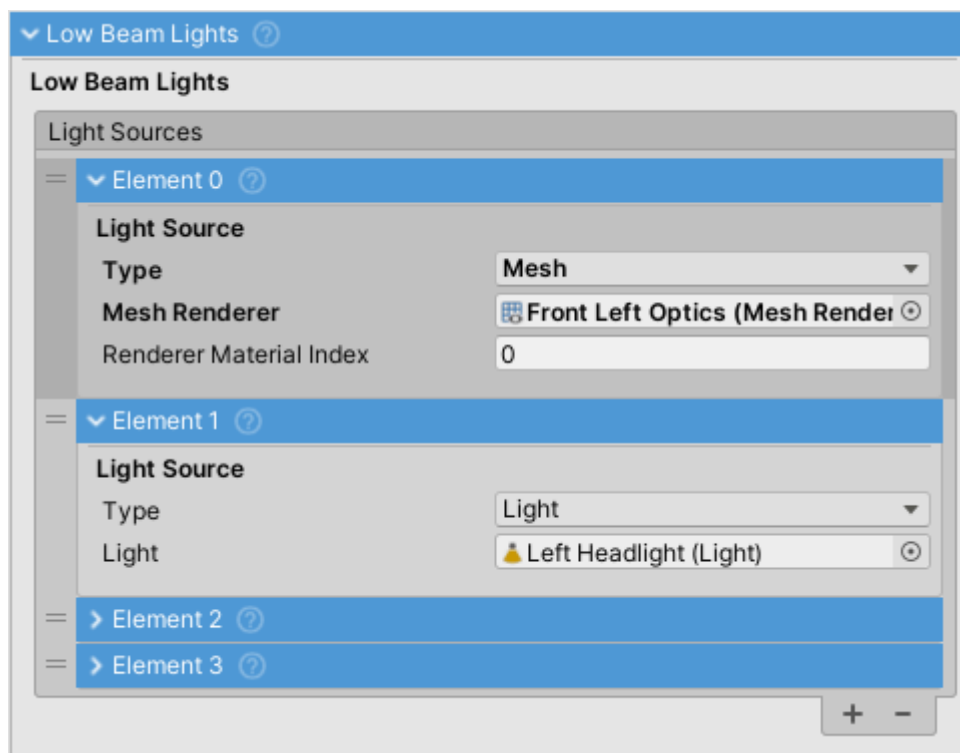
Lights Manager



LightsManager inspector.

LightsManager is tasked with turning `VehicleLights` on or off depending on user input.

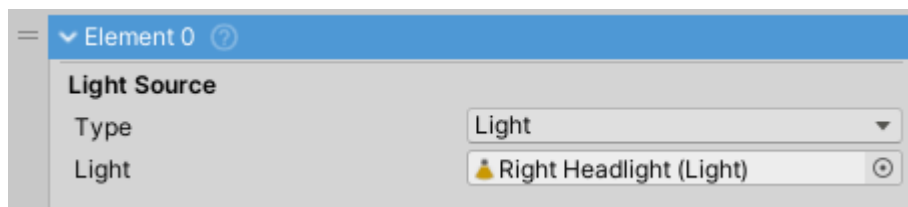
Vehicle Light



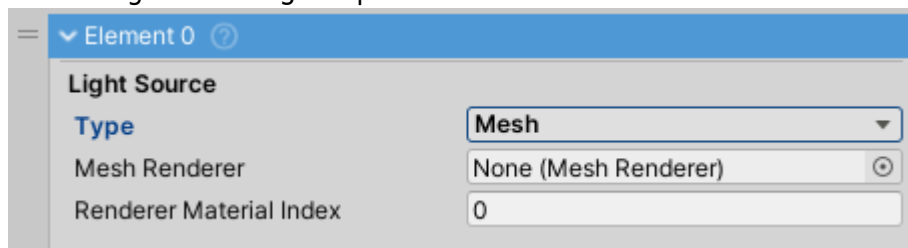
VehicleLight inspector.

VehicleLight is a collection of LightSources. When the light is turned on all LightSources are turned on and vice versa.

Light Source



VehicleLight with Light option selected.



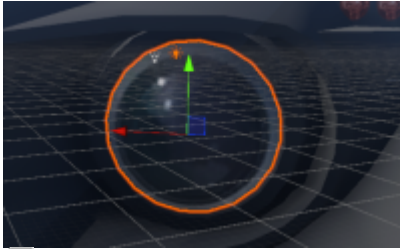
VehicleLight with Mesh option selected.

One vehicle light source. Can be a Light or emissive Mesh.

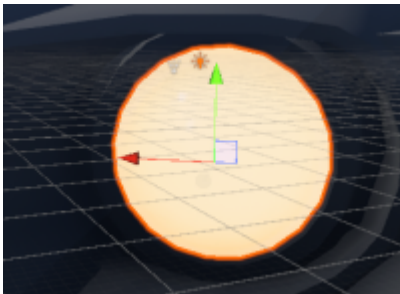
- If Light is selected as Type any Unity Light can be assigned. It will be turned on and off according to user input.
- If Mesh is selected as Type any Mesh with Standard shader can be used. It's Emission field will be toggled to imitate a working light. Since this does not emit enough light to be a headlight usually another light source is used in tandem, this one with SpotLight assigned.

For an emissive mesh light to work it needs to be a separate object. If the model does not come with lights and blinkers as separate objects these will need to be separated in 3D modelling software such as Blender (free, open source).

When using Mesh light source make sure to tick the Emission checkbox on the material. This lets Unity know that this variant of the material is in use and should be included in the build.



Mesh with Emission turned on.



Mesh with Emission turned off.

2020/04/08 12:36

2020/04/08 12:36

2020/04/08 12:36

Skidmark Manager

Skidmark Manager ?

Geometry

Max Marks Per Section

220

Min Distance

0.12

m

Ground Offset

0.03

m

Appearance

Skidmark Material

DefaultSkidmarkMaterial

Smoothing

0.0719

Global Skidmark Intensity

0.6

Max Skidmark Alpha

0.6

Fade Over Distance

☒

Lower Intensity Threshold

0.05

!

To change appearance of skidmarks on different surfaces, check GroundDetection preset.

Persistent Skidmarks

Persistent Skidmarks

☐

Persistent Skidmark Destroy Distar

100

m

SkidmarkManager inspector.

Skidmarks are generated when wheel skids / slips over a surface.

- Skidmarks are achieved by procedurally generating a mesh. One mark consists of two triangles and the number of triangles per one section can be calculated as $\text{Max Marks Per Section} * 2$.
- Min Distance is the distance a wheel needs to travel before a new mark is created.
- Default behavior is to delete the oldest triangles as soon as number of marks reaches Max Marks Per Section - similar to the old snake game. To make this transition smooth Fade Over Distance can be enabled or Persistent Skidmarks can be used. Check the section below for more info.
- If skidmarks are not visible or clip into the terrain GrounOffset needs to be increased.
- To define when the wheel is slipping Longitudinal Slip Threshold and Lateral Slip Threshold from vehicle settings tab are used.

Surface-based Skidmarks

- Skidmarks use settings of currently active SurfacePreset to get the settings for the current surface type.

Skidmarks	
Thread Albedo Visibility	<input type="range"/> 0
Thread Normal Visibility	<input type="range"/> 0
Base Intensity	<input type="range"/> 0
Slip Factor	<input type="range"/> 1

Skidmark section of AsphaltTireFrictionPreset.

- Also check [GroundDetection](#) page for more info.

Persistent Skidmarks



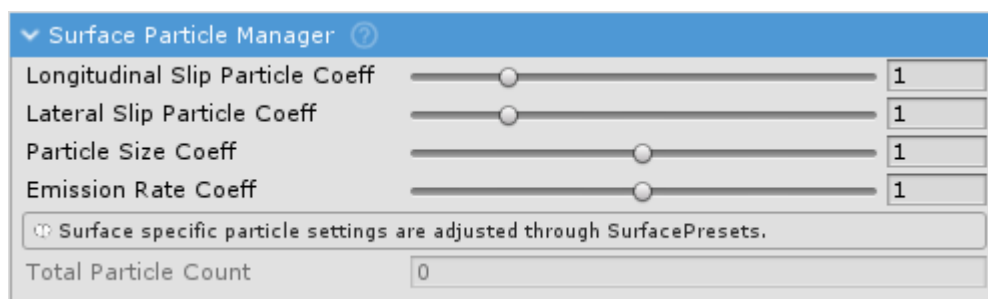
Persistent skidmarks enabled.

- Persistent skidmarks get stored into sections of Max Marks Per Section size.
- Sections do not get destroyed as long as the the player is under Persistent Skidmark Destroy Distance.
- Downside to this approach is that over time this will cause the number of triangles in the view-

port to increase.

2020/04/08 12:36

Surface Particle Manager



SurfaceParticleManager inspector.

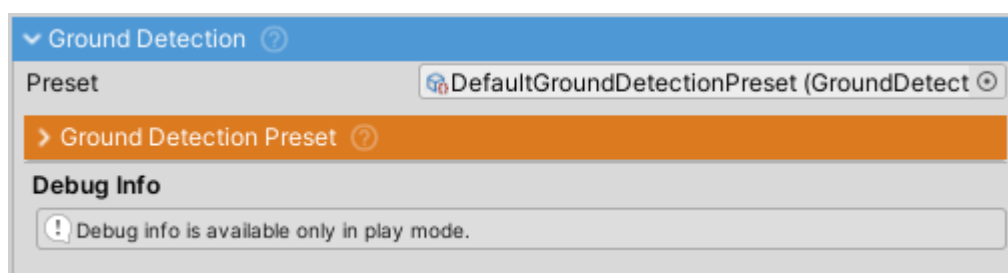
SurfaceParticleManager creates and manages particles based on current SurfacePreset settings.

- Particles are emitted on per-wheel basis. Total surface particle count is a sum of particle counts from all the wheels' ParticleSystems.
- Lateral Slip Threshold and Longitudinal Slip Threshold under *Settings* tab determine the lowest wheel slip threshold needed for wheel to be considered to be slipping. This affects particle effects.

2020/03/18 12:26

2020/04/08 12:36

Ground Detection



GroundDetection inspector.

GroundDetection is one of the most important aspects of NWH Vehicle Physics. It determines which WheelFrictionPreset will be used for calculating friction, which effects will be active and which sounds will play. In short, it determines which wheel is on which surface.

Ground detection works based on Terrain texture indices and object tags so it is very important how many textures there are assigned to the Terrain and in which order.

To prevent having to change the settings across all the vehicles GroundDetectionPreset ScriptableObject was introduced in NWH Vehicle Physics 2.

- NWH Vehicle Physics 2 supports ground detection with multiple Terrains in one scene.
- GroundDetection runs checks for a maximum of one wheel at a time. This is to improve on performance but it can introduce a small delay from the time the surface has been changed to the time GroundDetection registers it.

Ground Detection Preset

Ground Detection Preset ?

! This is a ScriptableObject. All changes are global.

Particles

Smoke Prefab SkidSmoke

Dust Prefab SurfaceDust

Surface Maps

Fallback Surface Preset FallbackSurfacePreset (SurfacePreset)

Surface Maps

Asphalt ?

Name Asphalt

Surface Preset AsphaltSurfacePreset (SurfacePreset)

Terrain Texture Indices

Element 0 5

+ -

Tags

Element 0 Road

+ -

Sand ?

Name Sand

Surface Preset SandSurfacePreset (SurfacePreset)

Terrain Texture Indices

Element 0 0

+ -

Tags

List is Empty

+ -

+ -

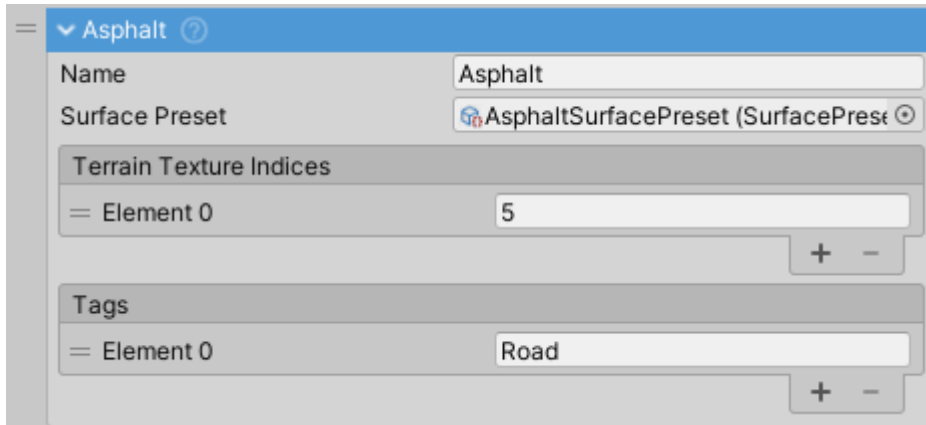
GroundDetectionPreset inspector.

GroundDetectionPreset is a ScriptableObject that determines which SurfacePreset will be used on which terrain texture and object. This is done through SurfaceMaps.

Surface Map

<http://nwhvehiclephysics.com/>

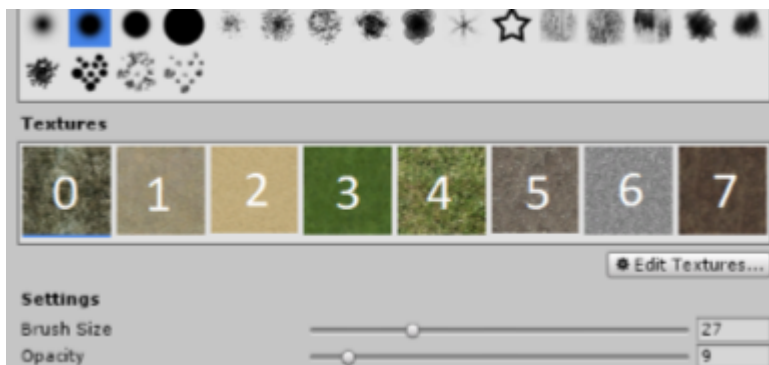
Printed on 2020/12/16 21:01



SurfaceMap inspector.

SurfaceMap tells GroundDetection which SurfacePreset to use for which terrain texture and/or object tag.

GroundDetection runs a check for each WheelComponent to determine which surface that WheelComponent is on. This is done by checking all the assigned SurfaceMaps in order and if any SurfaceMap has a matching terrain texture index in Terrain Texture Indices list, or the object the wheel is on has the tag from Tags list, Surface Preset will be assigned to that WheelComponent.



Terrain texture indices. Note that counting starts from 0.

Adding a New Surface Map

Adding a gravel SurfaceMap will be used as an example.

- Add a new SurfaceMap by clicking on + button on the bottom of the list.
- Check on which texture positions are the gravel textures. In the image above that would be 5 and 7. Add those numbers to the Terrain Texture Indices list.
- If there are objects that should represent gravel in the scene, assign a tag (e.g. GravelRoad) to those objects and add it to Tags list.

2020/04/08 12:36

Surface Preset

Surface Preset

This is a ScriptableObject. All changes are global.

General

Name

Gravel

Friction Settings

Friction Preset

GravelTireFrictionPreset (FrictionPreset)

Skidmark Settings

Draw Skidmarks

☒

Skidmark Material

SkidmarkGravel

Slip Factor

0.807

Dust/Smoke Particle Settings

Emit Particles

☒

Particle Type

Dust

Particle Size

1.63

Particle Color

Particle Max Alpha

1

Max Particle Emission Rate Over Distance

0.35

Particle Life Distance

15

Max Particle Lifetime

3

Chunk Particle Settings

Emit Chunks

☒

Max Chunk Emission Rate Over Distance

12

Chunk Life Distance

2

Max Chunk Lifetime

0.7

Sound Settings

Skid Sounds

Play Skid Sounds

☒

Skid Sound Volume

0.16

Skid Sound Pitch

0.4

Skid Sound Clip

GravelSkidNoise

Surface Sounds

Play Surface Sounds

☒

Slip Sensitive Surface Sound

☐

Surface Sound Volume

0.09

Surface Sound Pitch

1

Surface Sound Clip

GravelWheelNoise

SurfacePreset inspector.

SurfacePreset tells the VehicleController which settings to use for which surface. This determines tire friction, the look of effects such as skidmarks and particle effects. It also changes sounds to match the surface. Once set up, SurfacePreset can be used for different terrains, scenes or even across games since SurfacePresets do not carry any scene-specific fields.

Setup

- Right click on empty space in Project window and select Create ⇒ NWH Vehicle Physics ⇒ Surface Preset. This will create a new SurfacePreset ScriptableObject in the current directory.
- Assign a name to it for easier debugging later.

Friction

- Assign a Friction Preset. This is a ScriptableObject with settings for tire friction so that WheelController can adjust its behavior according to the surface type. With asset come multiple Friction Presets so just pick one of those for now.

Skidmarks

- Adjust skidmark settings according to the surface type. More about skidmark settings on [Skidmarks](#) page.
- Skidmark Material will be used on generated skidmarks while the wheel is on this surface type.
- Slip Factor determines if the skidmarks are slip dependent. On soft surfaces such as sand this value should be 0 as the skidmarks/thread-marks should always be visible, no matter the slip. On hard surfaces such as asphalt skidmarks are only visible when there is wheel slip.

Dust/Smoke Particles

- ParticleType - Smoke should be used for hard surfaces (asphalt, concrete), Dust for dusty surfaces (gravel, sand). The difference between the two is in the way the emission rates are calculated. If Smoke is selected this will be related to the wheel slip while it will depend on vehicle speed for Dust.
- Particle Life Distance is used to calculate ParticleSystem's Start Lifetime. The faster the vehicle is going the shorter the lifetime of the particles will be.
- Max Particle Emission Rate Over Distance determines amount of particles emitted over distance of one meter.
- Max Particle Lifetime determines the absolute maximum time a particle can be alive. When still this value determines how long the particle lives. When moving Particle Life Distance is used instead.

Chunk Particles

Chunk particles represent pieces of dirt, debris, sand or pebbles thrown behind the tire when there is wheel spin.

Depending on speed the lifetime of a a particle is calculated from Max Chunk Lifetime (when near still) or Chunk Life Distance (when moving).

Sounds

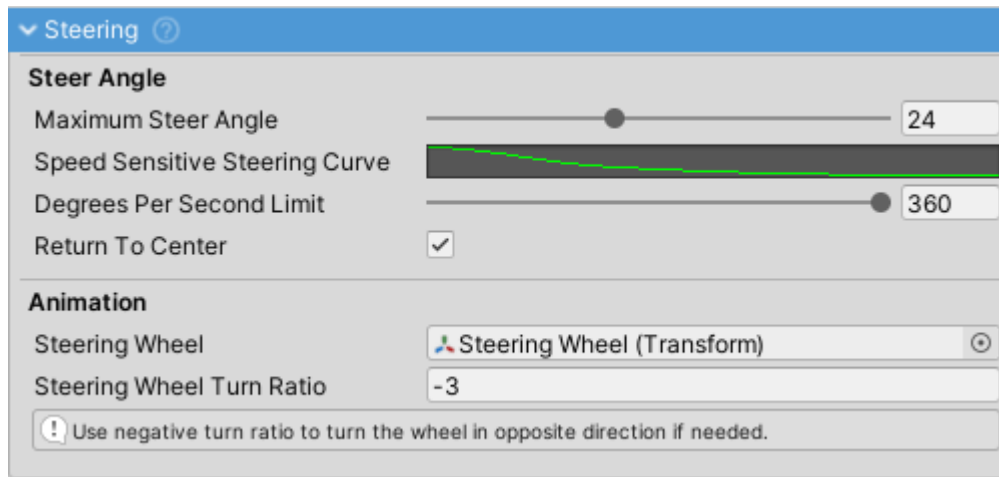
- SkidSound and SurfaceSound settings are used to set WheelSkidComponent and WheelTireNoiseComponent [SoundComponents](#) values for current surface.

2020/04/08 12:36

2020/04/08 12:36

2020/04/08 12:36

Steering



The image shows the 'Steering' inspector panel. It has a blue header with a dropdown arrow and a help icon. Below the header, there are two main sections: 'Steer Angle' and 'Animation'. The 'Steer Angle' section includes a slider for 'Maximum Steer Angle' set to 24, a 'Speed Sensitive Steering Curve' graph showing a green line on a black background, a slider for 'Degrees Per Second Limit' set to 360, and a checked checkbox for 'Return To Center'. The 'Animation' section includes a dropdown for 'Steering Wheel' set to 'Steering Wheel (Transform)' and a text input for 'Steering Wheel Turn Ratio' set to -3. A tooltip at the bottom says 'Use negative turn ratio to turn the wheel in opposite direction if needed.'

Steering inspector.

Controls vehicle's steering. Speed sensitive.

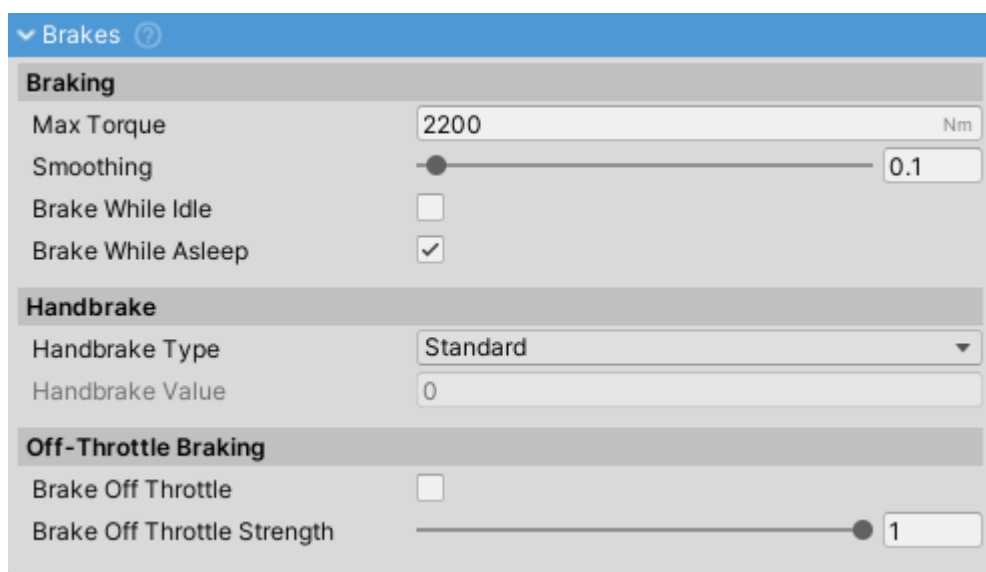
- If the vehicle is not steering also check [WheelGroup](#) settings. It is most likely that all WheelGroups have SteerCoefficient set to 0 which means no axles will steer.

Related:

- [Input Setup](#)
- [WheelGroup - Steering](#)

2020/04/08 12:36

Brakes



The image shows the 'Brakes' inspector panel. It has a blue header with a dropdown arrow and a help icon. Below the header, there are three main sections: 'Braking', 'Handbrake', and 'Off-Throttle Braking'. The 'Braking' section includes a text input for 'Max Torque' set to 2200 Nm, a slider for 'Smoothing' set to 0.1, and checkboxes for 'Brake While Idle' (unchecked) and 'Brake While Asleep' (checked). The 'Handbrake' section includes a dropdown for 'Handbrake Type' set to 'Standard' and a text input for 'Handbrake Value' set to 0. The 'Off-Throttle Braking' section includes a checkbox for 'Brake Off Throttle' (unchecked) and a slider for 'Brake Off Throttle Strength' set to 1.

Brakes inspector.

Controls brakes.

- Avoid using too high Max Torque as it might introduce jitter. Usually the best value is the one just above the point where the wheels lock up under heavy braking.
- Smoothing can be used to make braking more progressive when using binary input.

Related:

- [Input Setup](#)
- [BrakeHissComponent \(SoundComponent\)](#)
- [WheelGroup - Brakes](#)

2020/04/08 12:27

Damage

The image shows a 'Damage Handler' inspector panel. It is divided into several sections: 'Collision', 'Drivetrain Damage', 'Mesh Deformation', 'Actions', and 'Events'. The 'Collision' section has fields for 'Deceleration Threshold' (500 m/s²) and 'Collision Timeout' (0.8 s), and a list of 'Collision Ignore Tags' with 'Wheel' selected. The 'Drivetrain Damage' section has a 'Damage Intensity' slider set to 0.5. The 'Mesh Deformation' section has fields for 'Deformation Vertices Per Frame' (8000), 'Deformation Radius' (0.4 m), 'Deformation Strength' (1), and 'Deformation Randomness' (0.01), along with a 'Deformation Ignore Tags' list containing 'Wheel'. The 'Actions' section has a 'Repair' button. The 'Events' section shows 'On Collision (Collision)' and a message 'List is Empty'.

Damage Handler

Collision

Deceleration Threshold: 500 m/s²

Collision Timeout: 0.8 s

Collision Ignore Tags

Element 0: Wheel

Drivetrain Damage

Damage Intensity: 0.5

Damage debug info available in play mode.

Mesh Deformation

Deformation Vertices Per Frame: 8000

Deformation Radius: 0.4 m

Deformation Strength: 1

Deformation Randomness: 0.01

Deformation Ignore Tags

Element 0: Wheel

Actions

Repair

Events

On Collision (Collision)

List is Empty

DamageHandler inspector.

DamageHandler deforms the meshes on the vehicle (skinned meshes are not supported due to performance reasons).

- Damage depends on deceleration at the moment of impact meaning that the faster the vehicle goes and the faster it stops on collision the more damage vehicle will take.
- As the vehicle gets damaged more and more engine power will reduce, some randomness will be added to the engine sound and if the crash happened near the wheel (all the wheel meshes need to be tagged as "Wheel") wheel will start steering in a random direction degrading the handling of the vehicle. There is also a wheel wobble effect added.
- Mesh deformation that is the product of crashing is queue based. This means that only limited number of meshes will be deformed per frame. This value can be adjusted by changing the Deformation Vertices Per Frame field. Higher value will result in deformation being completed in less frames while lower value will stretch the deformation over the higher number of frames – and will also cause less frame dropping. It is best to adjust this value to as high as it is possible without causing frame drops. This will depend on the device the final game will run on.
- Deformation Radius, Deformation Strength and Deformation Randomness all affect how the mesh will be deformed – how far the vertex can be moved from the original position, how much it will be moved for the given deceleration and how random the final result will be. Setting randomness to 0 will cause final mesh to be smooth and setting randomness to a high value will result in noisy looking mesh.
- Deceleration Threshold is the minimum deceleration value needed for collision to be registered by DamageHandler.

2020/04/08 12:36

Settings

Settings

Actions

Validate Setup

Physics Sub-stepping

Low Speed Substeps

20

High Speed Substeps

20

Asleep Substeps

2

Physical Properties

!
Physical Properties should only be changed out of play mode or through scripting before vehicle initialization.

Vehicle Dimensions

X
1.83

Y
1.11

Z
4.6

!
Make sure to use 'Calculate Inertia Tensor' after adjusting vehicle dimensions.

Center Of Mass

X
0

Y
0.475002

Z
-0.1656258

Calculate Center Of Mass

Inertia Tensor

X
766.6821

Y
1394.951

Z
231.6374

Calculate Inertia Tensor

Mass

1385
kg

Drag

0

Angular Drag

0

Max Angular Velocity

10

Physics Material

VehicleMaterial

State Settings

State Settings

DefaultStateSettings (StateSettings)

> State Settings

LODs

!
Individual LOD settings can be changed through StateSettings above.

Update LODs

☒

Use Camera.main For LOD

☒

!
Enter play mode to view LOD debug data.

Positions

Engine Position

X
0

Y
0.56

Z
1.41

Transmission Position

X
0

Y
0.34

Z
0.79

Exhaust Position

X
0

Y
0.34

Z
-2.16

General

Longitudinal Slip Threshold

0.5

Lateral Slip Threshold

0.32

!
Slip threshold values are used only for effects and sound and do not affect handling.

Constrain When Asleep

☐

Settings tab.

Settings can be accessed through *Settings* tab in the VehicleController inspector.

NWH Vehicle Physics 2 Documentation - <http://nwhvehiclephysics.com/>

Actions

- **Validate Setup** - when clicked `Validate()` function gets called on `VehicleController` and all of its `VehicleComponents`. If there are any issues with the vehicle setup a message will pop up in the console.

[21:16:27] Sports Car: Validating VehicleController setup. If no other messages show up after this one, vehicle is good to go.

[21:16:27] Audio mixer of 'SoundManager' is not assigned.

Example `Validate Setup` output with missing audio mixer.

Physical Properties

Physical properties section mostly relates to the vehicle `Rigidbody`. `Rigidbody` properties are set by `VehicleController` at start. Changing them after `VehicleController` has been initialized will have no effect. If you need to do that change the values of the `Rigidbody` directly from your script.

- **Vehicle Dimensions** - dimensions of the vehicle in meters. Width x Height x Length. Has impact on aerodynamic drag calculation if `Aerodynamics` module is active.
- **Inertia Tensor** - inertia tensor of the vehicle. Unity calculates inertia of `Rigidbody`s as if they are solid (i.e. have uniform density). Cars have a lot of quite heavy parts and a lot of empty space and therefore custom inertia calculation is required. Click *Calculate Inertia Tensor* button to automatically calculate inertia. If you want the vehicle to have more rotational inertia around any of the three rotation axes increase the value for that axis. I.e. increasing Y will make the vehicle resist steering more.
- **Center Of Mass** is a point relative to the vehicle's coordinate system at which the center of mass of the `Rigidbody` will be. There is a gizmo of a weight at the center of mass position. Longitudinal positioning of center of mass heavily affects vehicle handling. Avoid using high centers of mass.
- **Calculate Center Of Mass** will auto-center center of mass to be exactly between the wheels. This is a good starting point but does not guarantee the best weight distribution and should be manually tweaked for best results.

Positions

- **Positions** determine the location of different vehicle parts. These positions are used for positioning sound sources, effects, etc.
- **Coordinates** are in local space.

Multiplayer

- **Multiplayer Instance Type** determines if the current vehicle instance is `Local` (i.e. controlled by a local player) or `Remote` (i.e. controlled by another player over network).

General

- Longitudinal Slip Threshold and Lateral Slip Threshold determine the amount of slip a wheel can have before it is considered to be slipping/skidding. Most sounds and effects will not trigger while the slip value is below the threshold value.
- Use Low Speed Fix - if true drag will be added to Rigidbody when vehicle velocity is ~ 0 to prevent vehicle from creeping on slopes due to the nature of slip calculation.

2020/03/18 20:12 · Aron Rescec

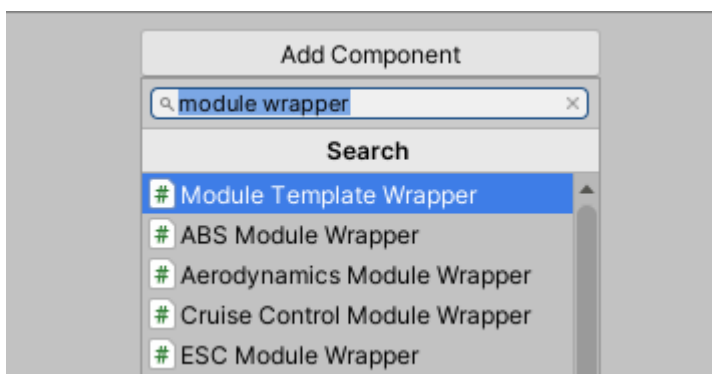
Modules

Modules

NWH Vehicle Physics 2 is a collection of `VehicleComponents`. All aspects of it are a component - sound components, effect components, etc. - they all inherit from `VehicleComponent`. The only way that a module is different from an inbuilt component is that it can be added or removed as needed. Modules carry over state system from the `VehicleComponent` which means that each module can be turned on or off, enabled or disabled or have LOD set.

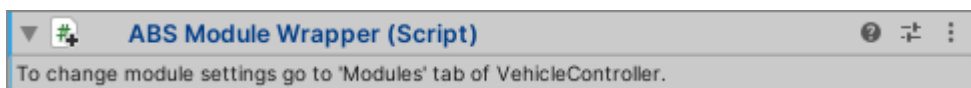
Usage

- To add a module click on *Add Component* button at the bottom of the Inspector. Modules must be added to an object that already contains `VehicleController`.
- Each module is wrapped in a `MonoBehaviour` wrapper `ModuleWrapper`. This is a way to work around lack of multiple inheritance in C#. So, to add a module just add its wrapper to the vehicle:

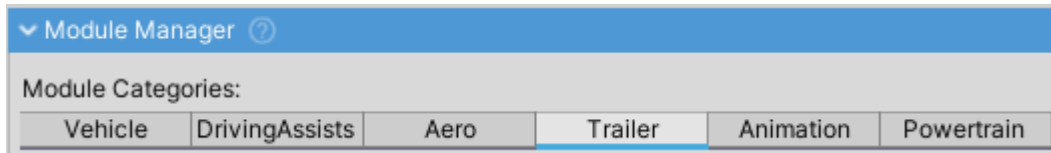


Adding a new module.

- Modules get drawn inside `ModuleManager` drawer. Go to `Modules` tab of `VehicleController` to see all of the available modules. This is to avoid cluttering the inspector with a multitude of different editors. Modules can also be filtered by category to make navigation easier.



Example module wrapper editor.



Module category selector inside ModuleManager.

Module Wrapper

Each module is wrapped in a MonoBehaviour wrapper called ModuleWrapper. This is a way to get around lack of multiple inheritance in C#.

VehicleModule inherits from VehicleComponent, but it also needs to be serialized and for that it needs to be a MonoBehaviour. The new attribute [SerializeReference] has been present in Unity since 2019.3 and original implementation used that but the amount of bugs and lack of backwards-compatibility with older versions of Unity resulted in the wrappers being used instead.

Scripting

Adding a Module

To add a module use:

```
myVehicleController.moduleManager.AddModule<MyModuleWrapperType,  
MyModuleType>();
```

Example (adding an ABSModule):

```
ABSModule absModule =  
myVehicleController.moduleManager.AddModule<ABSModuleWrapper, ABSModule>();
```

Getting a Module

To get a module use:

```
myVehicleController.moduleManager.GetModule<ModuleWrapperType,  
ModuleType>();
```

Example (getting ABSModule):

```
ABSModule absModule =  
myVehicleController.moduleManager.GetModule<ABSModuleWrapper, ABSModule>();
```

Alternatively, module can be retrieved through MonoBehaviour. The next snippet does exactly the same as the snippet above but might be more practical in some cases:

```
MyModule module =  
myVehicleController.GetComponent<MyModuleWrapper>().module as MyModule;
```

Removing a Module

To remove a module use:

```
myVehicleController.moduleManager.RemoveModule<ModuleWrapperType>();
```

Example (removing ABSModule):

```
myVehicleController.moduleManager.RemoveModule<ABSModule>();
```

Enabling/Disabling a Module

Since modules inherits from [VehicleComponent](#) they also work on the same principle:

```
myModule.Enable();
```

```
myModule.Disable();
```

Note that disabling or enabling a module will have no effect while LODs are active for that module as the LOD system will override the manual settings while active.

To disable LODs for a module use:

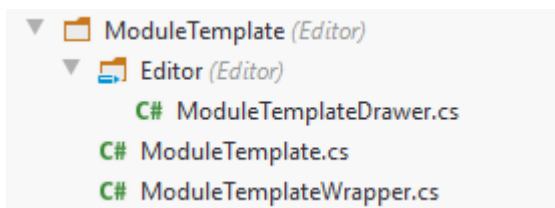
```
myModule.LodIndex = -1;
```

Class Reference

For class reference [click here](#).

Creating a New Module

- In the Scripts ⇒ Vehicle ⇒ Modules ⇒ ModuleTemplate folder there is an empty example module. Copy ModuleTemplate to create a starting point for a new module.
- Modules can be placed anywhere in the project and do not have to be in the same namespace as included modules.
- Directory structure of a module should be as following:



Example module directory structure.

- Each module must have at least three files:
 - ModuleDrawer (a type of CustomPropertyDrawer) placed in Editor folder.
 - Module file(s).

- ModuleWrapper
- **Each module must have a property drawer.** All module drawers inherit from ModuleDrawer which uses *NUI* editor GUI framework (developed by NWH coding) to render custom property drawers and editors through simplified syntax. This also makes the created module compatible with ModuleManager drawer.

Scripting Reference

Check out this link for NWH Vehicle Physics scripting reference. TODO

Module Manager

The screenshot shows the 'Module Manager' interface. Under the 'Aero' category, the 'Aerodynamics Module' is expanded. It contains two sections: 'Drag' and 'Downforce'. The 'Drag' section has a 'Simulate Drag' checkbox (checked), 'Frontal Cd' and 'Side Cd' sliders (set to 0.28 and 0.7 respectively), and 'Longitudinal Drag Force' and 'Lateral Drag Force' input fields (both set to 0). The 'Downforce' section has a 'Simulate Downforce' checkbox (checked) and a 'Max Downforce Speed' input field (set to 80 m/s). Below these is a 'Downforce Points' section with two elements. 'Element 0' has a position of (0, 0.3, 1.66) and a max force of 5000. 'Element 1' has a position of (0, 0.3, -1.58) and a max force of 5000. A warning message at the bottom of the Drag section states: 'To change vehicle dimensions go to 'Other > Settings' tab.'

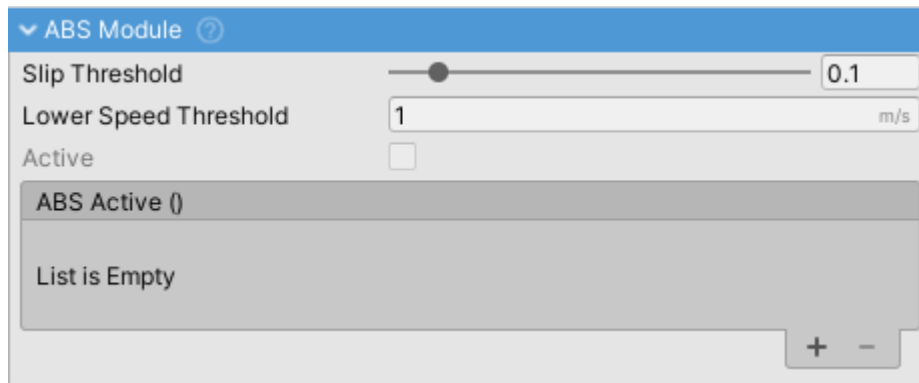
ModuleManager inspector.

ModuleManager is a VehicleComponent that manages VehicleModules.

2020/04/08 12:36

Included Modules

ABS



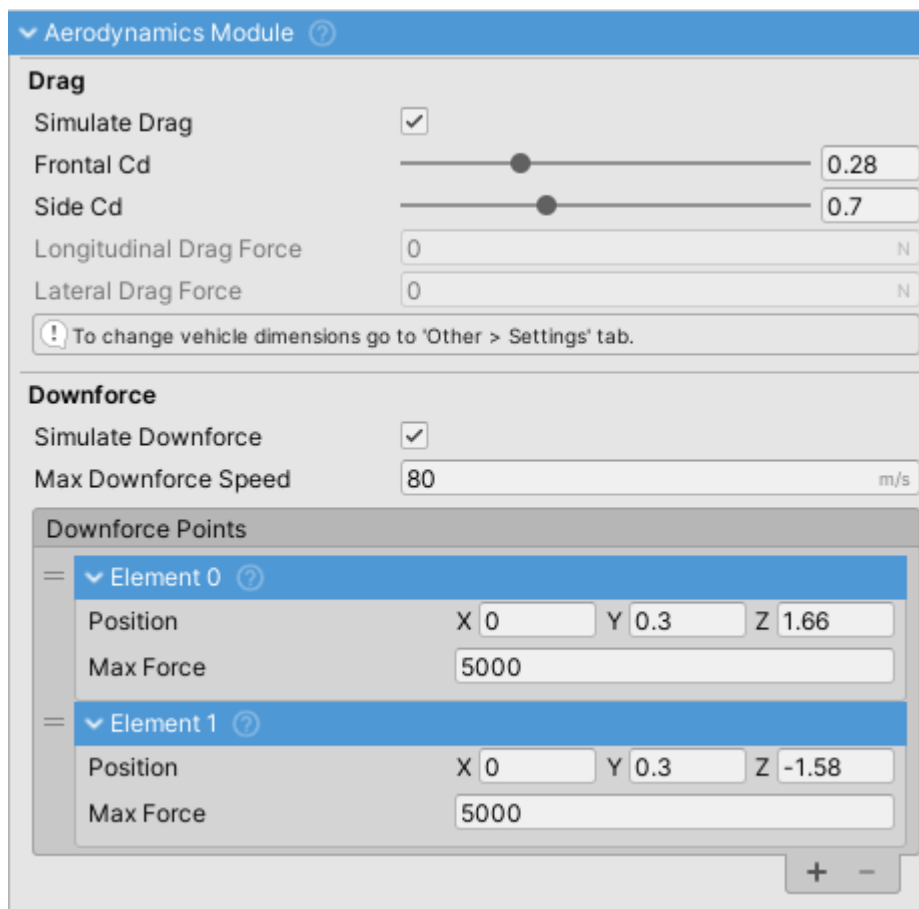
ABSModule inspector.

Anti-lock Braking System module.

Prevents wheels from locking up by reducing brake torque when slip reaches too high value. Slip value above which ABS is activated can be adjusted through Slip Threshold field.

2020/04/08 12:37

Aerodynamics



AerodynamicsModule inspector.

Calculates and applies aerodynamic drag and downforce.

Drag

- Drag depends on vehicle dimensions. Those can be adjusted under vehicle's *Settings* tab.
- Drag is calculated both in longitudinal and lateral directions. Intensity of drag can be adjusted through Frontal Cd and Side Cd (Cd = coefficient of drag) fields. Data for different vehicles is available [here](#).

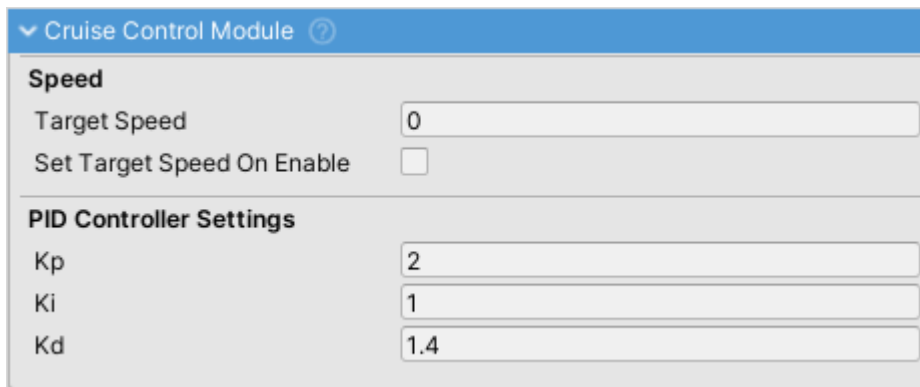
Downforce

Downforce is calculated in a simplified fashion by applying downforce to a number of points over the vehicle. In the simplest form a single downforce point at the center of the vehicle can be used, or one point at the front and one point at the end of the vehicle.

- Vertical position of Downforce Points should be below the WheelController position, or even as low as the floor of the vehicle. This is because all the force is applied in a single point which, if applied too high, can cause the vehicle to snap oversteer when changing direction.
- Downforce is not dependent on vehicle shape or dimensions. It is calculated through Downforce Points and Max Downforce Speed.
- Downforce increases exponentially from 0 to Max Downforce Speed at which it reaches Max Force value.
- Enable Gizmos to be able to see downforce points (red sphere).

2020/04/08 12:37

Cruise Control



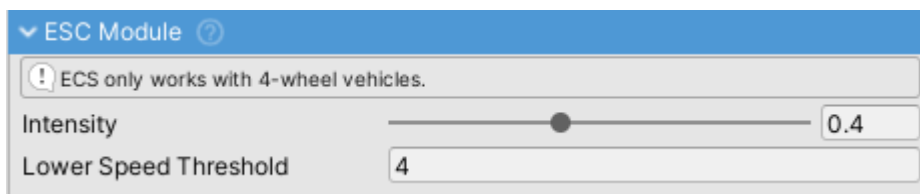
The screenshot shows the 'Cruise Control Module' settings. It has a blue header with a dropdown arrow and a help icon. Below the header, there are two sections. The first section is 'Speed' and contains 'Target Speed' with a text input field set to '0', and 'Set Target Speed On Enable' with an unchecked checkbox. The second section is 'PID Controller Settings' and contains three text input fields: 'Kp' set to '2', 'Ki' set to '1', and 'Kd' set to '1.4'.

CruiseControl module.

- Cruise Control implemented through a PID controller.
- Target Speed value sets the speed that cruise control will try to achieve. Ignored in reverse.
- Can apply throttle and braking.

2020/04/08 12:37

ESC



The screenshot shows the 'ESC Module' settings. It has a blue header with a dropdown arrow and a help icon. Below the header, there is a warning box with an exclamation mark icon and the text 'ECS only works with 4-wheel vehicles.' Below the warning box, there are two settings: 'Intensity' with a slider control set to '0.4', and 'Lower Speed Threshold' with a text input field set to '4'.

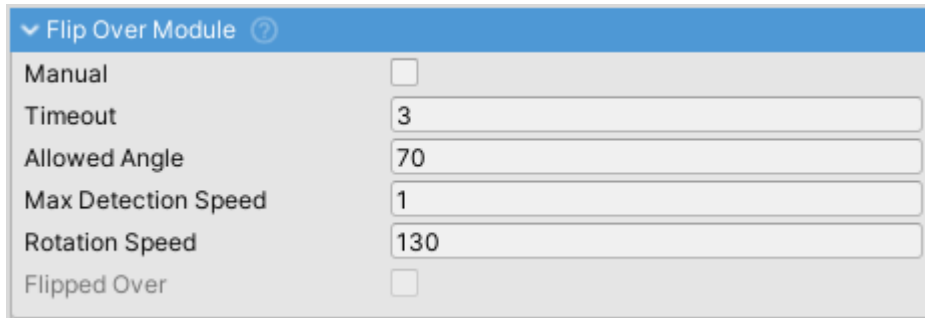
ESC module inspector.

Electronic Stability Control (ESC) module.

Applies braking on individual wheels to try and stabilize the vehicle when the vehicle velocity and vehicle direction do not match.

2020/04/08 12:37

Flip Over



The screenshot shows the 'Flip Over Module' settings. It has a blue header with a dropdown arrow and a help icon. Below the header, there are six settings: 'Manual' with a checkbox, 'Timeout' with a text input field containing '3', 'Allowed Angle' with a text input field containing '70', 'Max Detection Speed' with a text input field containing '1', 'Rotation Speed' with a text input field containing '130', and 'Flipped Over' with a checkbox.

Flip Over Module ?	
Manual	<input type="checkbox"/>
Timeout	3
Allowed Angle	70
Max Detection Speed	1
Rotation Speed	130
Flipped Over	<input type="checkbox"/>

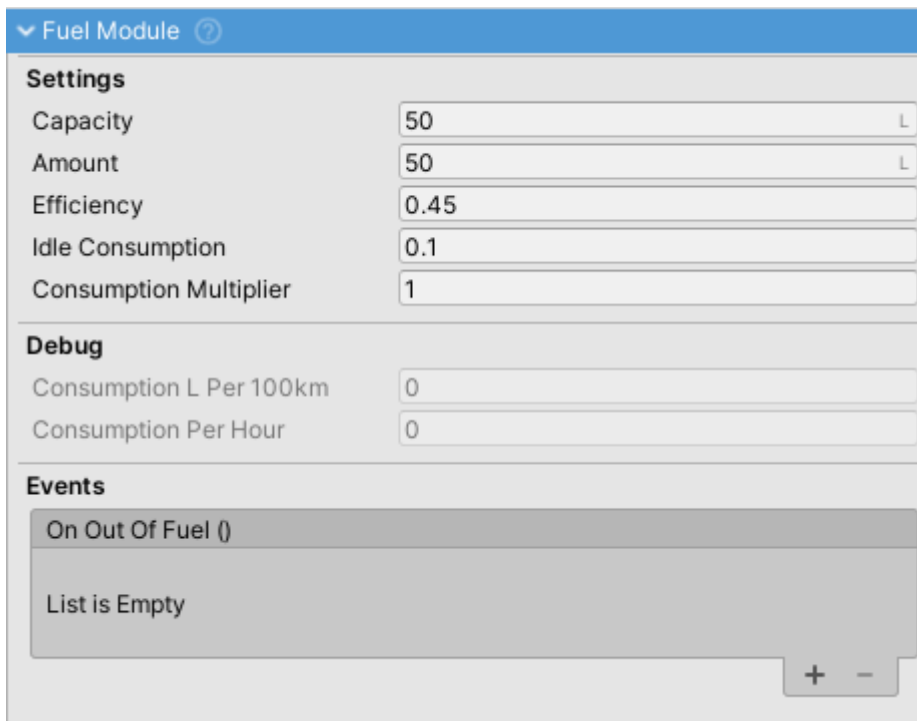
FlipOver module.

If the vehicle gets flipped over FlipOverModule will flip it to be right side up again.

- If Manual is set user will have to press a button to flip the vehicle over.

2020/04/08 12:37

Fuel



The screenshot shows the 'Fuel Module' settings. It has a blue header with a dropdown arrow and a help icon. Below the header, there are three sections: 'Settings' with five text input fields (Capacity: 50 L, Amount: 50 L, Efficiency: 0.45, Idle Consumption: 0.1, Consumption Multiplier: 1), 'Debug' with two text input fields (Consumption L Per 100km: 0, Consumption Per Hour: 0), and 'Events' with a list box containing 'On Out Of Fuel ()' and 'List is Empty'. There are '+' and '-' buttons at the bottom right of the list box.

Fuel Module ?	
Settings	
Capacity	50 L
Amount	50 L
Efficiency	0.45
Idle Consumption	0.1
Consumption Multiplier	1
Debug	
Consumption L Per 100km	0
Consumption Per Hour	0
Events	
On Out Of Fuel ()	
List is Empty	

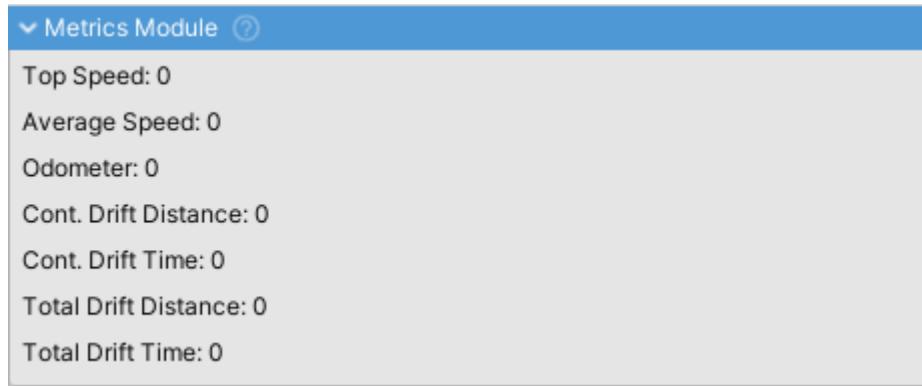
FuelModule inspector.

Module for simulating the fuel system in a vehicle. Fuel consumption gets automatically generated from engine efficiency (average ICE efficiency is used) and fuel energy content. Consumption can be adjusted through Consumption Multiplier.

- Prevents engine from running when out of fuel.
- Amount indicates the amount of fuel currently in the tank while Capacity indicates maximum tank capacity.

2020/04/08 12:37

Metrics



▼ Metrics Module ?

Top Speed: 0

Average Speed: 0

Odometer: 0

Cont. Drift Distance: 0

Cont. Drift Time: 0

Total Drift Distance: 0

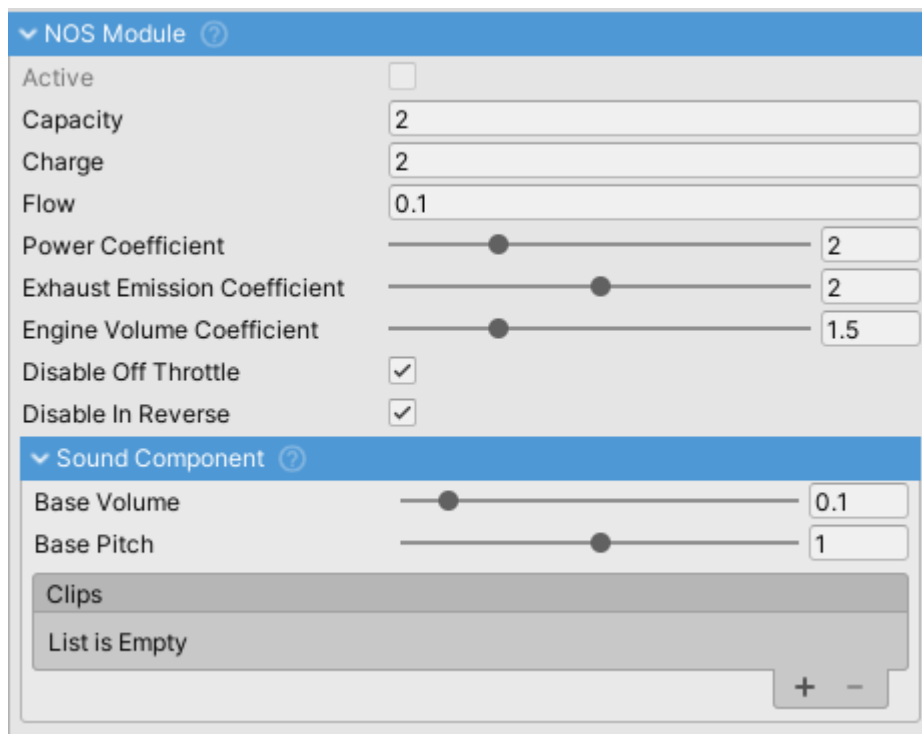
Total Drift Time: 0

MetricsModule inspector.

MetricsModule is used to record data about vehicle behavior. That data can then be used for display purposes, achievements or e.g. a boost system using NOS.

2020/04/08 12:37

NOS



▼ NOS Module ?

Active ☐

Capacity

Charge

Flow

Power Coefficient

Exhaust Emission Coefficient

Engine Volume Coefficient

Disable Off Throttle ☒

Disable In Reverse ☒

▼ Sound Component ?

Base Volume

Base Pitch

Clips

List is Empty

+ -

NOSModule inspector.

NOS (Nitrous Oxide System) module.

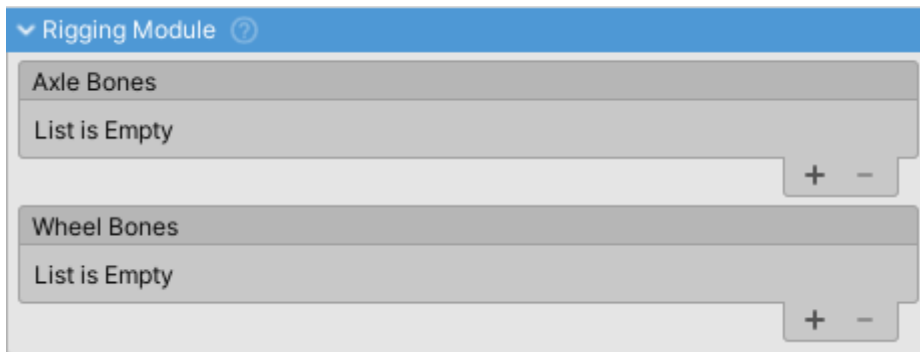
- Adds power to the engine.
- Affects engine sound by increasing volume.
- Has it's own SoundComponent which imitates hiss caused by releasing highly pressurised NOS

from the bottle.

- If ExhaustFlash effect is enabled it will be active while NOS is active.

2020/04/08 12:37

Rigging



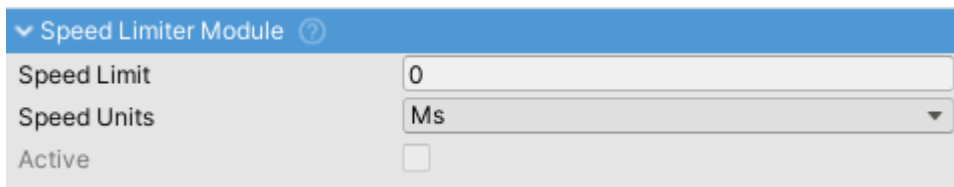
RiggingModule inspector.

Module used to animate rigged models by moving the axle/wheel bones.

- Axle Bones - list of handles controlling the axle bones. Each item is a single axle handle.
- Wheel Bones - list of handles controlling the wheel bones. Each item is a single wheel bone handle.

2020/04/08 12:37

Speed Limiter



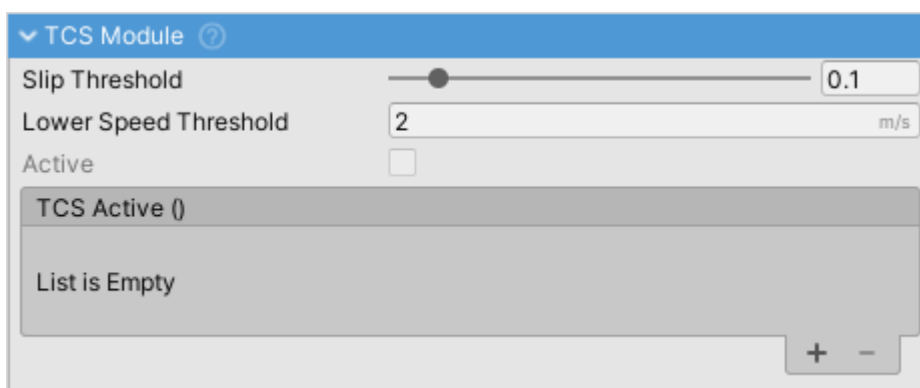
SpeedLimiter module inspector.

Module that limits vehicle speed to the set speed limit.

Only limits throttle application, does not apply brakes. For that use CruiseControlModule.

2020/04/08 12:37

TCS

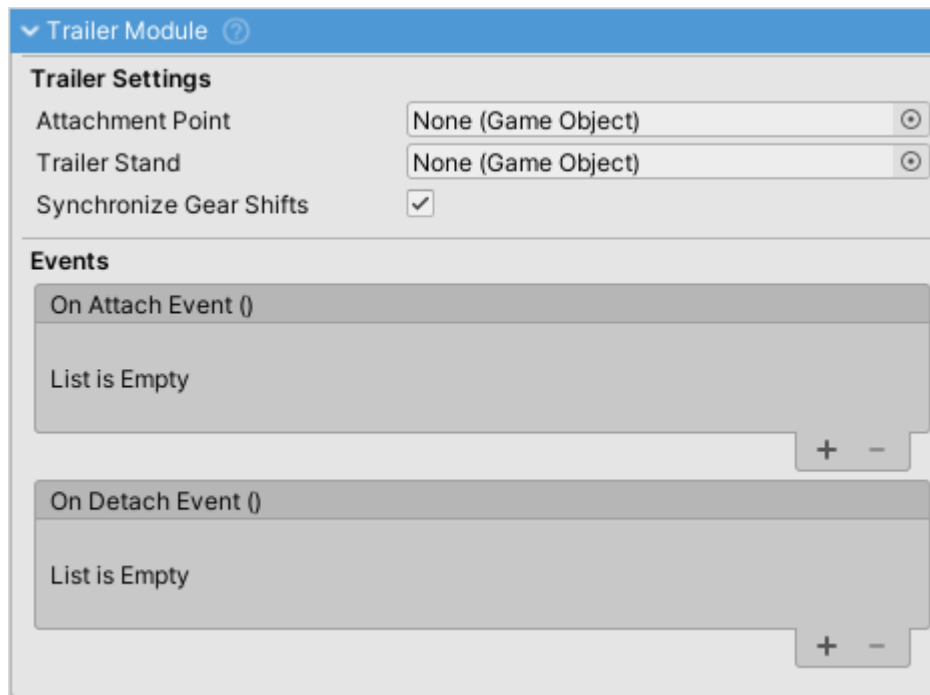


TCSModule inspector.

Traction Control System (TCS) module. Reduces engine throttle when excessive slip is present.

2020/04/08 12:37

Trailer



TrailerModule inspector.

TrailerModule works in tandem with TrailerHitchModule. VehicleController that has TrailerModule is able to attach to a VehicleController that has TrailerHitchModule.

- One vehicle can have both TrailerHitchModule and TrailerModule.
- Attachment Point is the point at which the trailer will be attached to the towing vehicle.
- Trailer Stand is the object which will be enabled if the trailer is detached and vice versa. It prevents the trailer from tipping forward on trailers with only the back axle.
- If Synchronize Gear Shifts is enabled the trailer object will be kept in the same gear. This allows for powered trailer or vehicles that are constructed out of two Rigidbodies.

Also check [Trailer Hitch module](#).

2020/04/08 12:37

Trailer Hitch

Trailer Hitch Module ?

Attachment

Attachment PointNone (Game Object)

Attach Distance Threshold0.5

Attach On Play☐

Detachable☒

Trailer In Range☐

Joint

Break ForceInfinity

Use Hinge Joint☐

Events

On Attach Event ()

List is Empty

+ -

On Detach Event ()

List is Empty

+ -

TrailerHitch module.

VehicleController with TrailerHitchModule can attach a VehicleController with TrailerModule as a trailer.

- Both TrailerHitchModule and TrailerModule can be present on one vehicle at the same time.
- AttachmentPoint is the point at which the trailer will be attached. The trailer will be moved so that both trailer and hitch AttachmentPoints are at the same position. This is where the physics joint gets created.

Also check [Trailer module](#).

2020/04/08 12:37

2020/04/08 12:36

From:

<http://nwhvehiclephysics.com/> - **NWH Vehicle Physics 2 Documentation**

Permanent link:

<http://nwhvehiclephysics.com/doku.php/Setup/VehicleController>

Last update: **2020/06/19 15:30**

