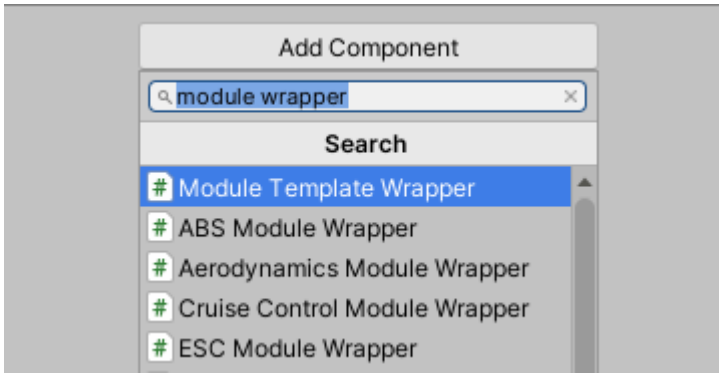
Modules

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

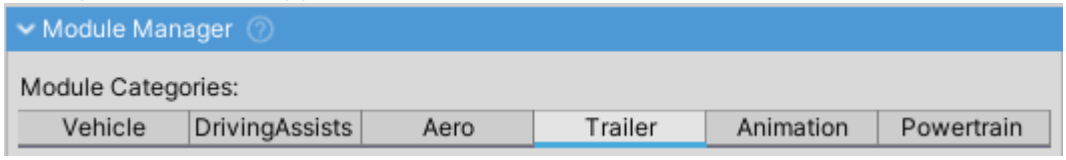
- To add a module click on *Add Component* button at the bottom of the Inspector. Modules must be added to an object that already contains `VehicleController`.
- Each module is wrapped in a MonoBehaviour wrapper `ModuleWrapper`. This is a way to work around lack of multiple inheritance in C#. So, to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

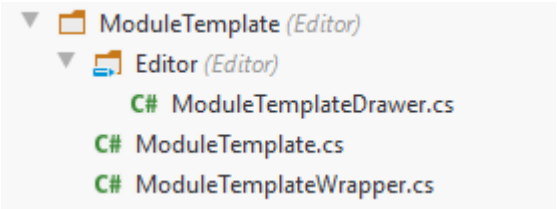
```
myModule.LodIndex = -1;
```

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:



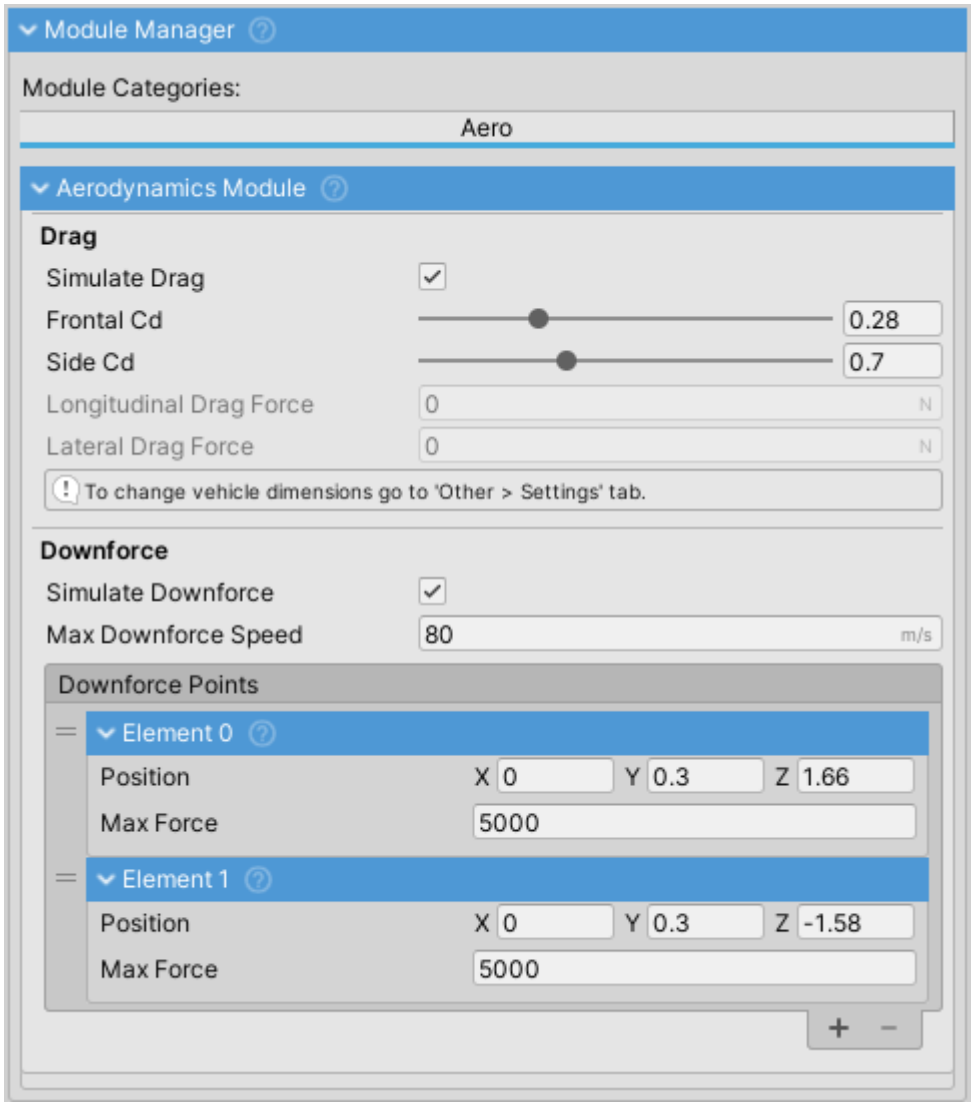
Example module directory structure.

- 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



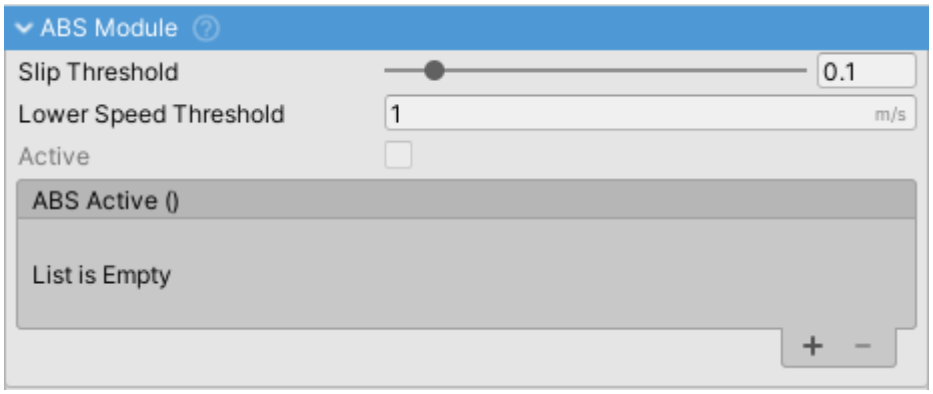
ModuleManager inspector.

`ModuleManager` is a `VehicleComponent` that manages `VehicleModules`.

2020/04/08 12:36

Included Modules

ABS

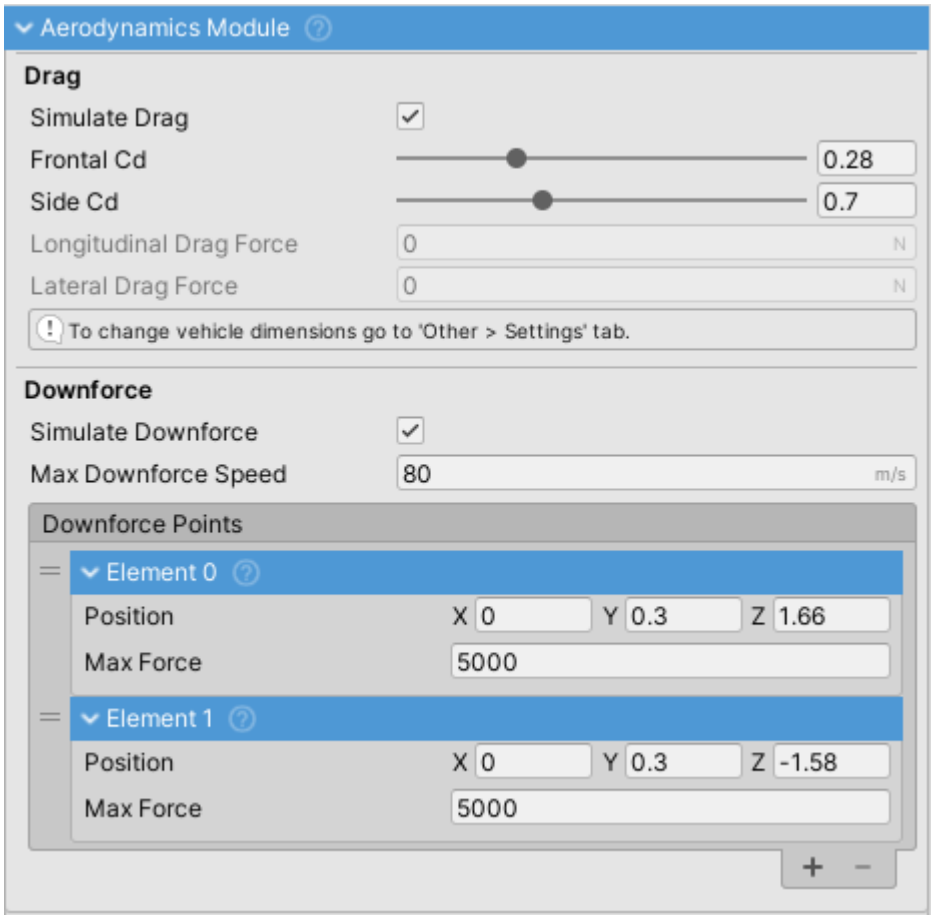


ABSModule inspector.

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 bot 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.

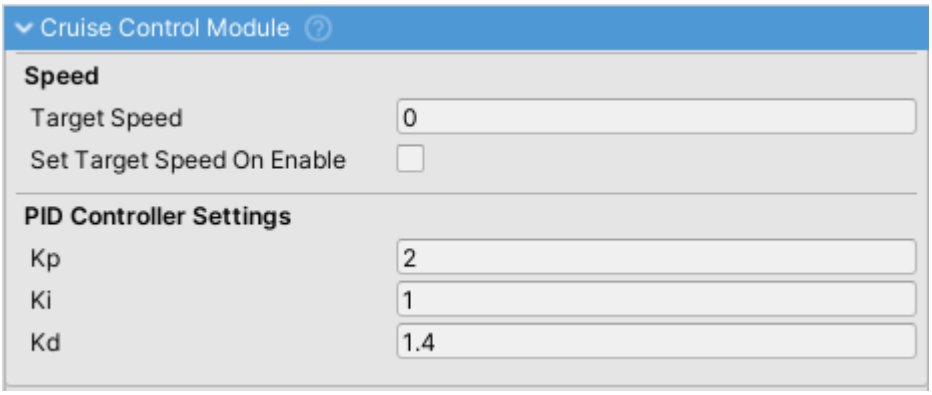
Downforce

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

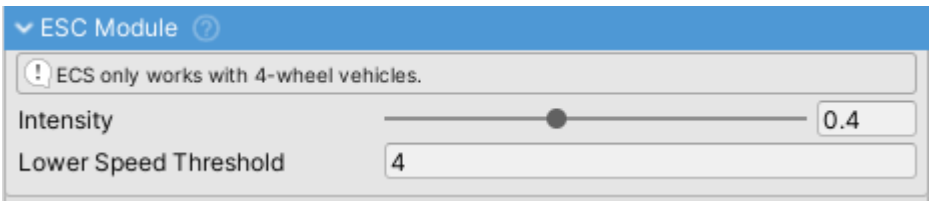


CruiseControl module.

- 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



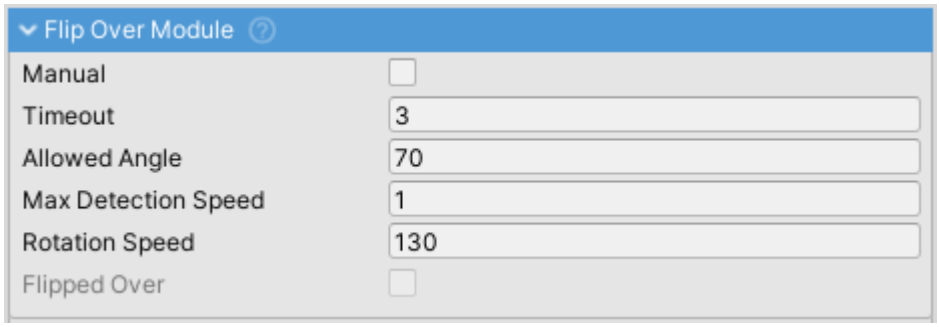
ESC module inspector.

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



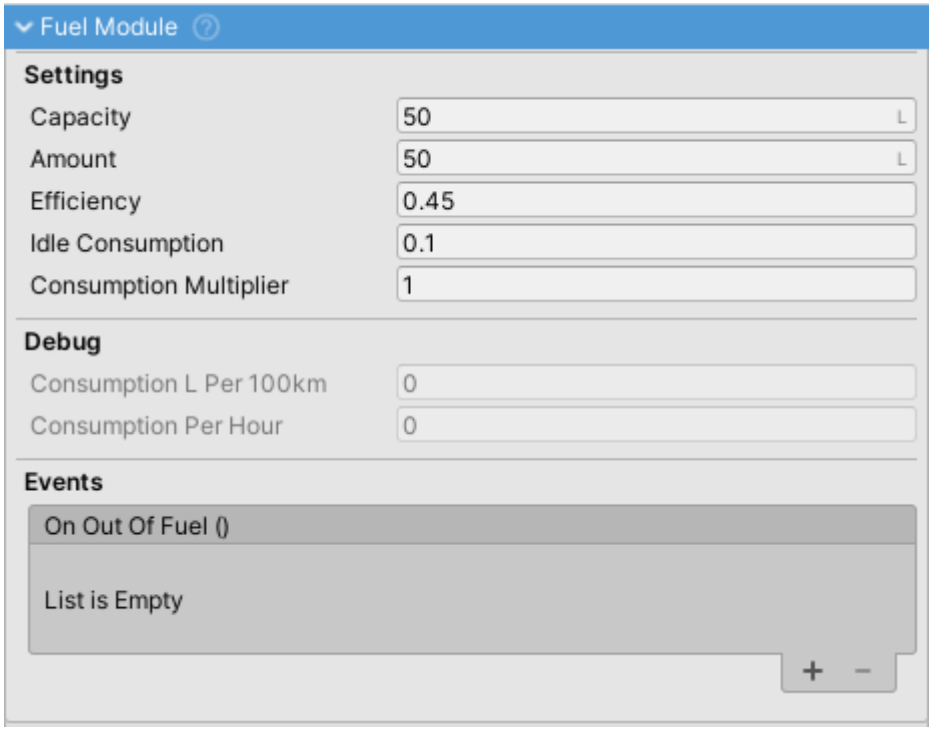
FlipOver module.

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



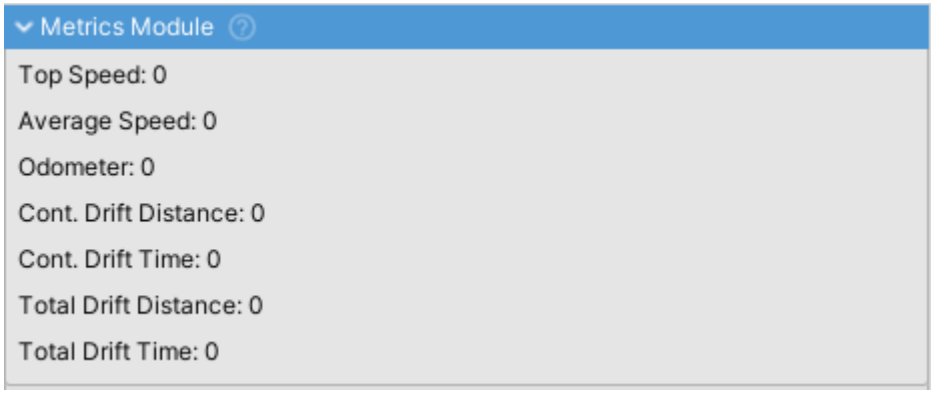
FuelModule inspector.

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

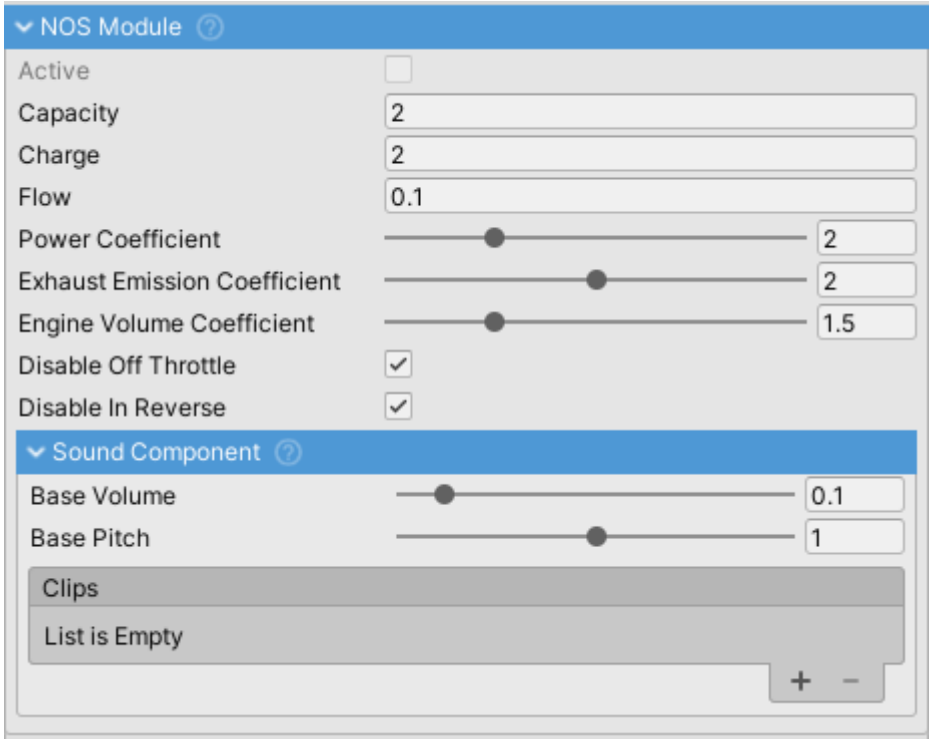


MetricsModule inspector.

`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



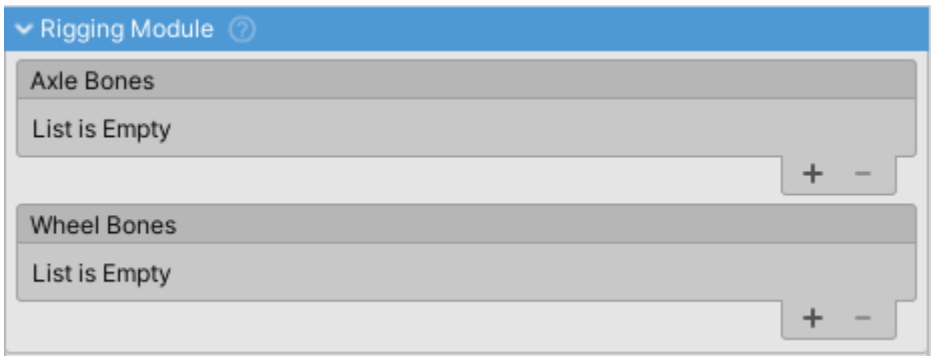
NOSModule inspector.

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



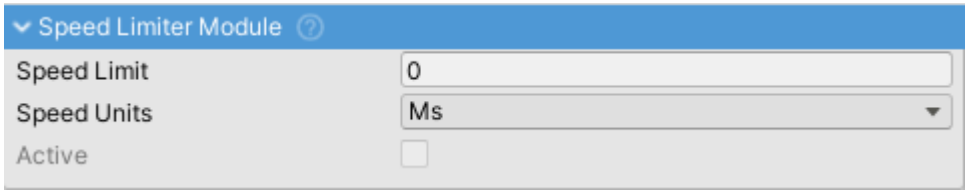
RiggingModule inspector.

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

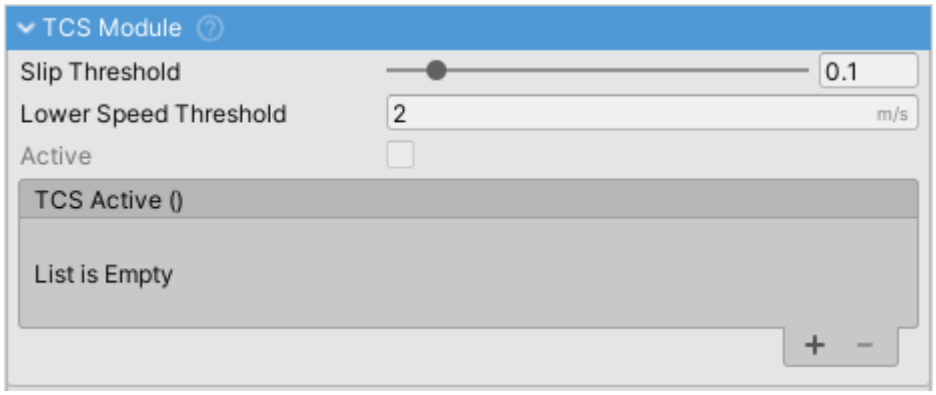


SpeedLimiter module inspector.

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

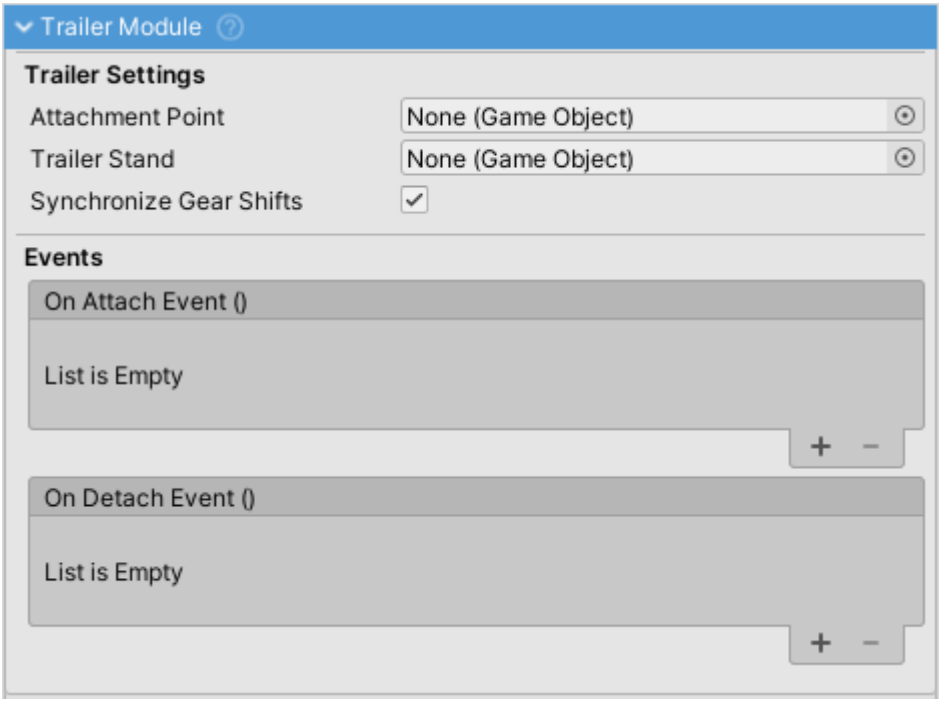


TCSModule inspector.

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

2020/04/08 12:37

Trailer



TrailerModule inspector.

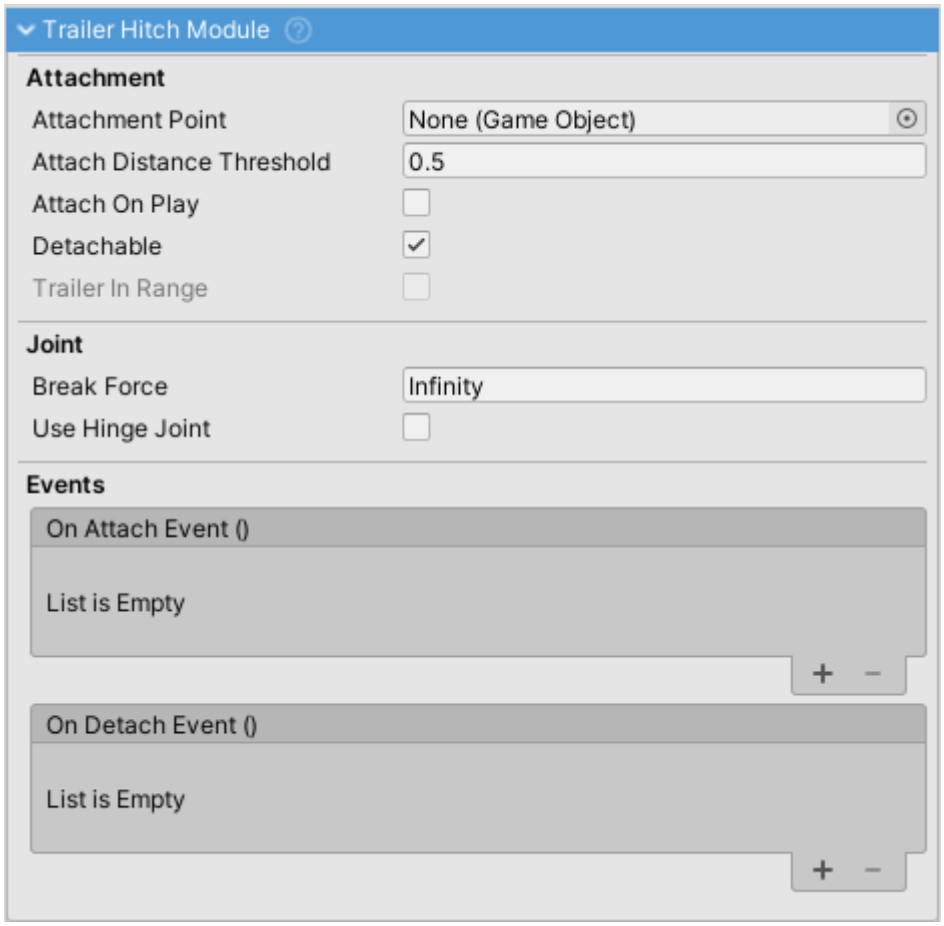
`TrailerModule` works in tandem with `TrailerHitchModule`. `VehicleController` that has `TrailerModule` is able attach to a `VehicleController` that has `TrailerHitchModule`.

- 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



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

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

Permanent link:
http://nwhvehiclephysics.com/doku.php/NWH/VehiclePhysics2/Modules/index

Last update: **2020/09/10 18:16**