

Vehicle Physics by **NWH**

Manual

Before You Start

Assuming you just want to open the demo scene and try the asset out without reading through the novel below, here are the steps needed (reading through the manual is still highly recommended):

- Import the package from the Asset Store
- Open demo scene (Island). You will get a few errors related to missing things and inputs.
- Import “Characters” and “Environment” from Standard Assets. Those cannot be included per Asset Store policy.
- Close Unity and copy the project settings files from ProjectSettings.zip over to your ProjectSettings directory ([your_project_name]/ProjectSettings). This will overwrite your existing settings.
- Start Unity again and you are good to go.

These steps are not needed if you are not going to use the demo scene. In that case you will only need to set up input bindings. This is explained in chapter *Input*.

Table of Contents

Setting Up a Vehicle.....	6
Adding Wheels.....	6
Adding Vehicle Controller.....	9
Input.....	10
Steering Wheels.....	11
Setting up a Tracked Vehicle.....	12
Vehicle Controller Script.....	15
Script Structure.....	16
Sound.....	17
Effects.....	19
Skidmarks.....	19
Lights.....	19
Exhausts.....	21
Steering.....	22
Engine.....	23
Forced Induction.....	24
Transmission.....	25
Axles.....	26
Brakes.....	28
Driving Aids.....	28
Damage.....	29
Fuel.....	30
Rigging.....	30
Flip Over.....	30
Trailer.....	31
Ground Type Detection.....	32
Vehicle Changer Script.....	33
Character Vehicle Changer.....	33
Cameras.....	34
Camera Changer Script.....	34
Other Scripts.....	35
Vehicle.....	35
Center Of Mass.....	35
Downforce.....	35
GUI.....	36
Dash GUI Controller.....	36
Analog Gauge.....	36
Digital Gauge.....	37
Dash Light.....	37
Compatibility With Other Assets.....	38
CTS (Complete Terrain Shader).....	38
IK Driver Vehicle Controller.....	38
FAQ.....	39

Thank you for buying NWH Vehicle Physics!

This manual will first explain how to set up your own vehicle model and then go into more detail about each component and how it works.

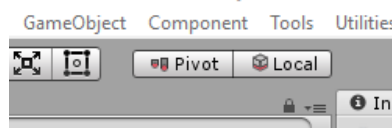
What it will not do is explain each member of each class – for that check the class reference or just open the script you are interested in. All of the public members have both XML and Tooltip explanations.

First things first: Model rotation and pivot points

Before starting it has to be mentioned that this manual assumes that the model is in proper Unity rotation – z-forward, x-right and y-up. Many of the models that were not made for Unity have this rotation wrong and it will have to be fixed before setting up a vehicle. This is because all the scripts assume that z is forward and y is up and therefore all the physics calculations are done that way. Here is a link to official guide on fixing the rotation of an imported model:

<https://docs.unity3d.com/Manual/HOWTO-FixZAxisIsUp.html>

Same goes for pivot point. It does not matter where pivot of the vehicle itself is but make sure that wheels have pivot in the same place their center is. You can check that by trying to rotate the wheel around it's X axis in the editor. This is the way that the script will rotate it. If it rotates properly then it is all good. If it starts moving around this means that the pivot is not in the center. Another way to check is by simply going to the top left corner and pressing Center/Pivot button:



If the handle does not move when clicking on that button this, again, means that pivot is where the center is. If the opposite happens you will have to use the same method for fixing the rotation mentioned in the link above to fix the pivot on all of the wheels. Pivoting point of other objects does not matter (except for steering wheel, more about that later).

All the units in the scripts or in this manual are SI units (meters, newtons, etc.).

Recommended Fixed Timestep: 0.02 – mobile and arcade games, 0.015 – general desktop games, 0.01 – simulation desktop games. Avoid using high center of mass and short springs with fixed timestep of 0.02.

If you have just imported the asset and are getting standard asset errors or input not set errors, check the README or the second page of this manual first.

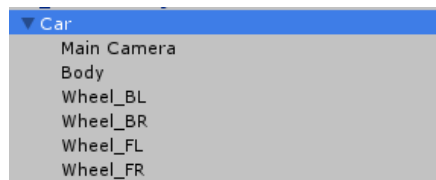
Setting Up a Vehicle

For this section an included car model will be used. Model can be found inside “Models/Low Poly Car” folder.

Adding Wheels

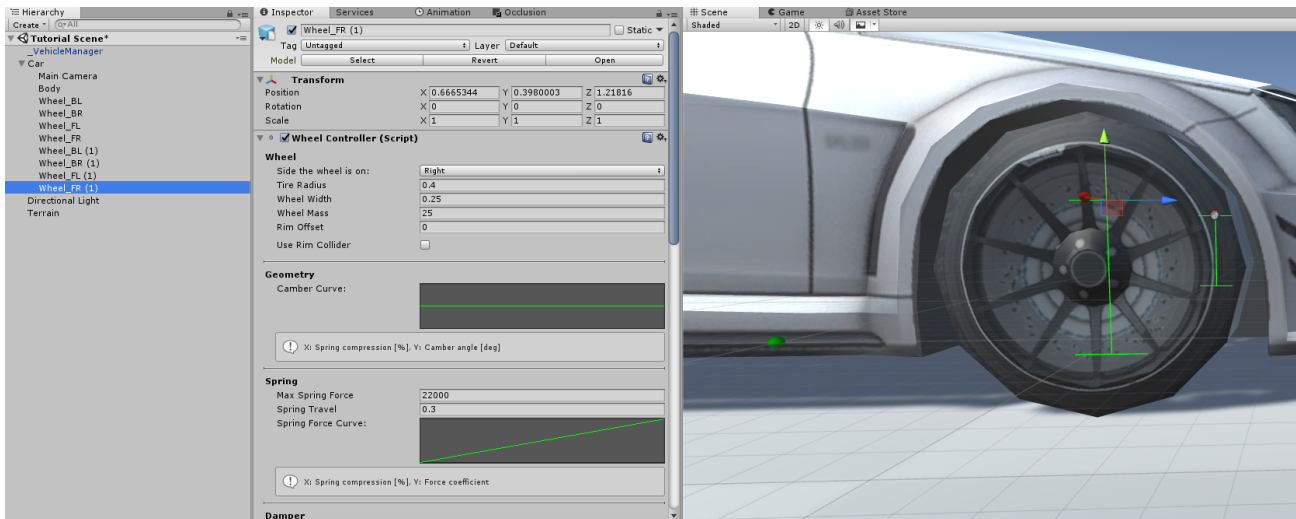
NWH Vehicle Physics (NVP) uses included Wheel Controller 3D (WC3D) for wheel physics, manual for which you can check out if you want a more in-depth explanation of all the segments of the wheel physics and how to adjust them.

This is an initial hierarchy of the Car model in question (ignore the camera):

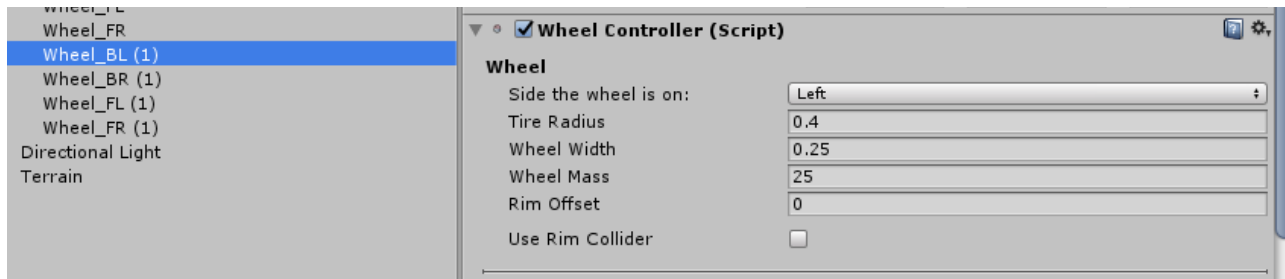


To set up a vehicle with WC3D:

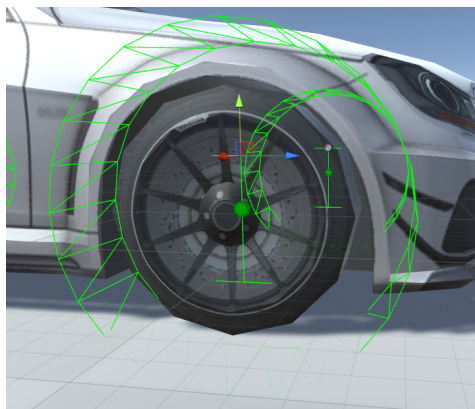
1. Add rigidbody to your parent object (*Car* in this case) and set it's mass to some realistic value – e.g. 1300.
2. Add colliders to your children. At least one collider of any type needs to be present for rigidbody to work properly. e.g. add a mesh collider to the *Body* object.
3. Now that the rigidbody is all set up wheel controller objects need to be created. These will hold the scripts and will not rotate with wheels. Do not add wheel controller script to the objects you want to rotate (*Wheel_BL*, *Wheel_BR*, etc.). Easiest way to create wheel controller objects is by duplicating the wheels (right click → duplicate or Ctrl + D) and then removing mesh filter and mesh renderer objects from the duplicates. Then, move the duplicated objects up to a point where you want the top of your spring travel to be. Finally, add Wheel Controller component to all duplicated objects. Result should look like the following picture where green lines show spring travel.



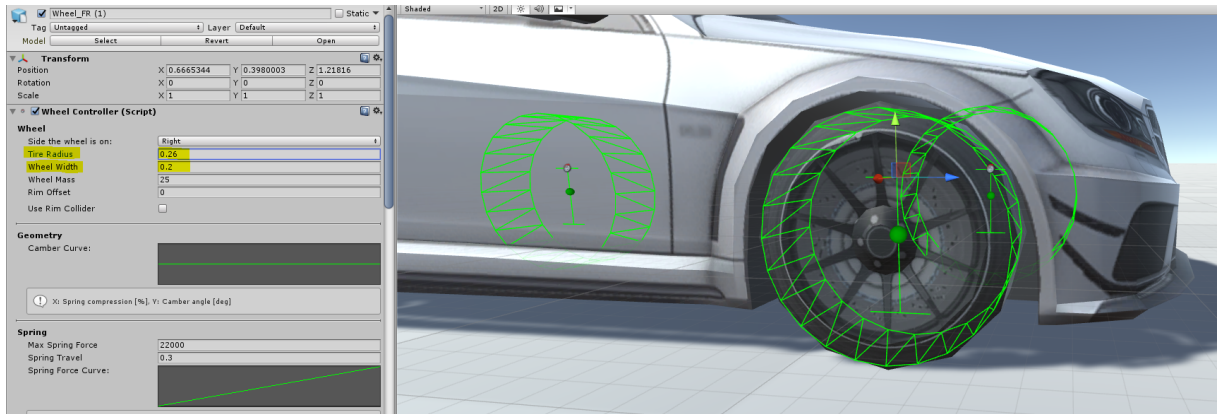
4. Check that the side the wheel is on has been detected correctly. Center wheel will not be auto-detected. If the side is wrong, chances are that you have wrong model rotation (check the beginning of this manual).



5. Scroll to the bottom of the Wheel Controller inspector and you will see an empty *Wheel Visual* field. Drag corresponding wheel visual to this field. If your wheel consists of multiple parts (tire, rim, disk, etc.) group them up under single object making sure that pivot stays where it should be. Result:



- Adjust wheel controller radius and width in the WC3D inspector until the green lines fit the wheel.



- Set up **ignore layer**. This is one of the most important steps and if skipped wheels will detect vehicle itself as ground. Create a new layer, name it however you want and assign it to all of the colliders on the vehicle. In this case only the *Body* object has a collider and so it's layer was set to the newly created layer (in this case *WheelControllerIgnore*). You can change which layer you want to ignore at the bottom of the WC3D inspector:



- Press Play. Your vehicle's suspension will now function if everything was done right. You might get slight vibration if you center of mass is high but that will be solved later.

Adding Vehicle Controller

Vehicle Controller has an automated default setup. If you ever want to return the state of vehicle controller to the way it was when first added you can use Reset option by clicking the cog to the right of the component name. Custom reset code will set up all the default values and clips for you.

To set up a vehicle controller:

1. Go to the parent object (*Car* in this case) and click on Add Component → Vehicle Controller. This will set up all the defaults for you. You can check Axles → Left/Right Wheel in the Vehicle Controller inspector to see if the axles have been assigned correctly.
2. Drag “_VehicleManager” prefab from “Prefabs/Vehicles/” folder to the scene. This prefabs contains all the vehicle changing and input handling scripts needed.
3. Remove “Character Vehicle Changer” component from the _VehicleManager object since we do not need it at this point.
4. Under *VehicleChanger* component set size of Vehicles to 1 and drag you vehicle to the Element 0 field.
5. Press play. You can now drive your vehicle. Default values might not be optimal for your type of vehicle so check out the rest of this manual for more detailed explanation.

Input

Vehicles themselves do not do any input processing but rather rely on external scripts to provide those value. Instead, vehicle controller contains only input states that can be accessed like so:

```
vehicleController.input.[nameOfInput] = someValue;
```

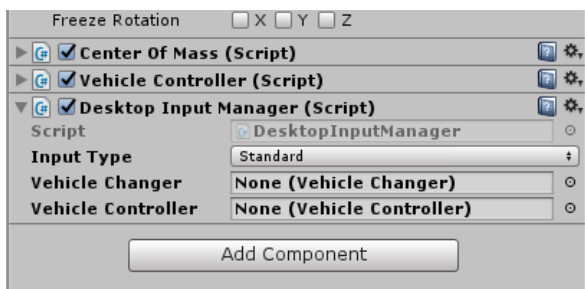
Currently available inputs are: Horizontal, Vertical, ShiftUp, ShiftDown, Clutch, Handbrake, leftBlinker, rightBlinker, lowBeamLights, fullBeamLights, hazardLights, trailerAttachDetach and horn.

With the asset there are two input scripts provided: Desktop Input Manager and Mobile Input Manager. Both scripts have only one function: to set the above listed values. This is so that custom input scripts can be easily made if needed (e.g. compatibility with 3rd party input assets).

So, what both of those managers do is in essence this:

```
if (Input.GetButtonDown("ShiftUp"))  
    vehicleController.input.ShiftUp = true;
```

Both provided input managers have two fields: *Vehicle Changer* and *Vehicle Controller*. If the field *Vehicle Changer* is set to a valid vehicle changer component all the input from the input manager will be rerouted to `vehicleChanger.ActiveVehicleController`. If the field *Vehicle Changer* is left empty input will be routed to either the set *Vehicle Controller* or to the Vehicle Controller component if such is attached to the same GameObject as the input manager.



By default Desktop Input Manager uses Edit > ProjectSettings > Input (Input Manager) button names instead of hard-coded key codes, so following the README is suggested or else demo scene will throw errors if required buttons are not set up.

NVP uses the following axes:

- Horizontal
- Vertical
- EngineStartStop
- LeftBlinker
- RightBlinker
- Lights
- FullBeamLights
- HazardLights
- Handbrake
- ChangeCamera
- TrailerAttachDetach
- ShiftUp
- ShiftDown
- [Optional, for use with e.g. H-shifter] FirstGear, SecondGear, ThirdGear, FourthGear, FifthGear, SixthGear, SeventhGear, EighthGear, NinthGear, Neutral, Reverse
- [Manual clutch only] Clutch
- Horn
- [Split vertical input only] Throttle, Brake

Steering Wheels

Asset will work out of the box with most steering wheels and has been tested with Logitech wheels. In some cases gas and brake pedals will report as separate axes. To fix this go to “Set up USB game controllers” → Select your wheel → Properties → Settings → tick ‘Combined’ option.

Since version 1.7 separate vertical axes option is available and can be used by selecting “Composite” as *Vertical Input Type* inside Desktop Input Manager inspector. This requires “Accelerator” and “Brake” axes to be set up inside Unity input manager.

Setting up a Tracked Vehicle

Setting up a tracked vehicle is similar to setting up a normal vehicle from previous chapter.

Requirements

- Number of wheels on left and right sides has to be the same.
- Track will only be able to move if model supports it. Skinned mesh renderer with bones is required – one bone for each wheel.

Features That are not Supported

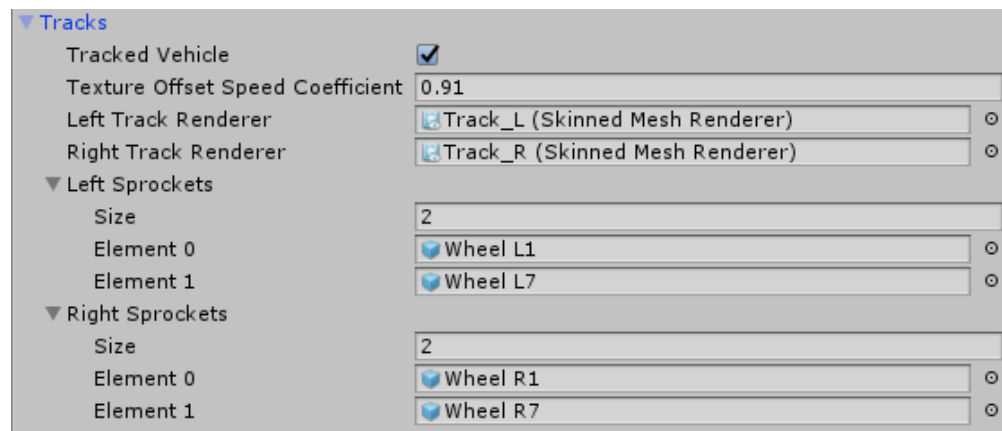
- Friction curves. Since the original focus of the asset is vehicles with tires friction curves are set up for such vehicles. Friction calculation and curves for rubber tires are not suitable for steel tracks so WC3D does separate calculation for tracked vehicle that is not dependent on the surface. Slip and friction coefficients still work so that can be used if different behavior is required.
- Differential Types (both center and axle). Differential types have no effect on tracked vehicles since they use their own torque splitting calculations.
- Wheel / track slip. Tracked vehicles will never spin the wheels or tracks.
- Camber

Steps Needed to Set up a Tracked Vehicle

- Set up the vehicle just like you would a car (previous chapter). After setting up press Play. Vehicle should be driveable as a normal vehicle.
- Either copy the values from the tank in the demo or change the following values manually (values are exemplary):
 - Rigidbody mass – 30,000 (tanks are heavy)
 - Engine Power – 600 (this value needs to be fairly high)
 - Wheel Controller:

- Spring force: 90000 (needs to be adequate to support the heavy rigidbody, ignore the message about it being too low as the message is for 4-wheeled vehicles)
 - Damper bump/rebound: 5000
 - Slip and force coefficients can be left as they are (1). If the vehicle turns too slowly you can adjust side force coefficient to a lower value and by doing that reduce the drag of tracks when turning or alternatively the engine power can be increased.
- Set up axles. Everything under *Geometry* settings should be set to 0, including steer coefficient. *Power coefficient*, *differential strength* and *differential type* can be set to any value as they are not taken into account. Other axle values work normally.
 - Go under *Tracks* foldout and tick *Tracked Vehicle* option. Other settings under this foldout are visual only.
 - Go under *Steering* and check that the *Tracked Steer Intensity* is set to 1. This is a starting point and you can change it later. Higher value will result in tighter turning radius but will also slow down the vehicle more.

Animating the Tracks



It is required that the track be a single mesh with a material that uses the standard shader and has bones for each wheel which move the track as they are dragged around. If your model does not have such tracks it will either need to be edited in

your 3D modeling software of choice or you could possibly use the tracks provided with the demo vehicle.

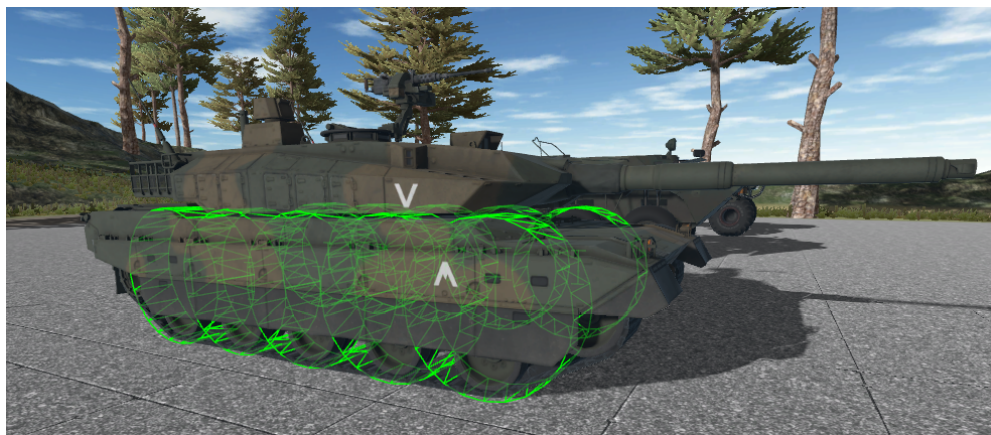
Track rotation animation is done through offsetting the texture through the offset option on the standard shader. To assign tracks use *Left* and *Right Track Renderer* fields. To match the speed of rotation of the track texture to the rotation of the wheels change the *Texture Offset Speed Coefficient* under *Tracks* foldout.

Most tanks have two additional 'wheels' placed in the front and rear that under normal circumstances do not touch the ground but still spin with tracks. Those wheels do not have suspension and should not use *Wheel Controller*. In the inspector those are called *Left Sprockets* and *Right Sprockets*. Any game object attached to this list will rotate at the same speed as the wheels.

Best way to move the bones in the tracks is to assign them to *Nonrotating Objects* field of respective *WheelController* component. This way the bone will stay in a fixed position relative to the wheel and move the track up and down along with the wheel.

Wheel Enlargement

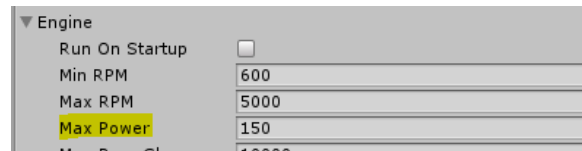
Since v.1.6 *Tracks > Wheel Enlargement* has been added. This option, if set to value larger than 1, will virtually increase the wheel size to make the contact surface of the vehicle flatter and more track-like. This value is a coefficient with which the wheel radius value will be multiplier (up to 2 → double the radius). To use wheel enlargement just set up the tank as per the instructions above and set the enlargement to the value you want. Everything else will be taken care of at runtime (resizing the wheels, changing transmission ratio due to larger wheels, etc.). Just take into account that the tops of the wheel gizmos (image below) should not stick outside of the body colliders or else they might come into contact with some surface when vehicle is upside-down or on it's side which will result in jumping.



Vehicle Controller Script

VehicleController script is the main part of Vehicle Physics package. As the name suggest it controls the vehicle and all of it's aspects – drivetrain, sounds, effects, etc.

When making VehicleController script decision was made to keep the default inspector. This enables you to use any field you see in the inspector from inside some other script. E.g. if you want to set max engine power you can do it from the inspector:



Or using code:

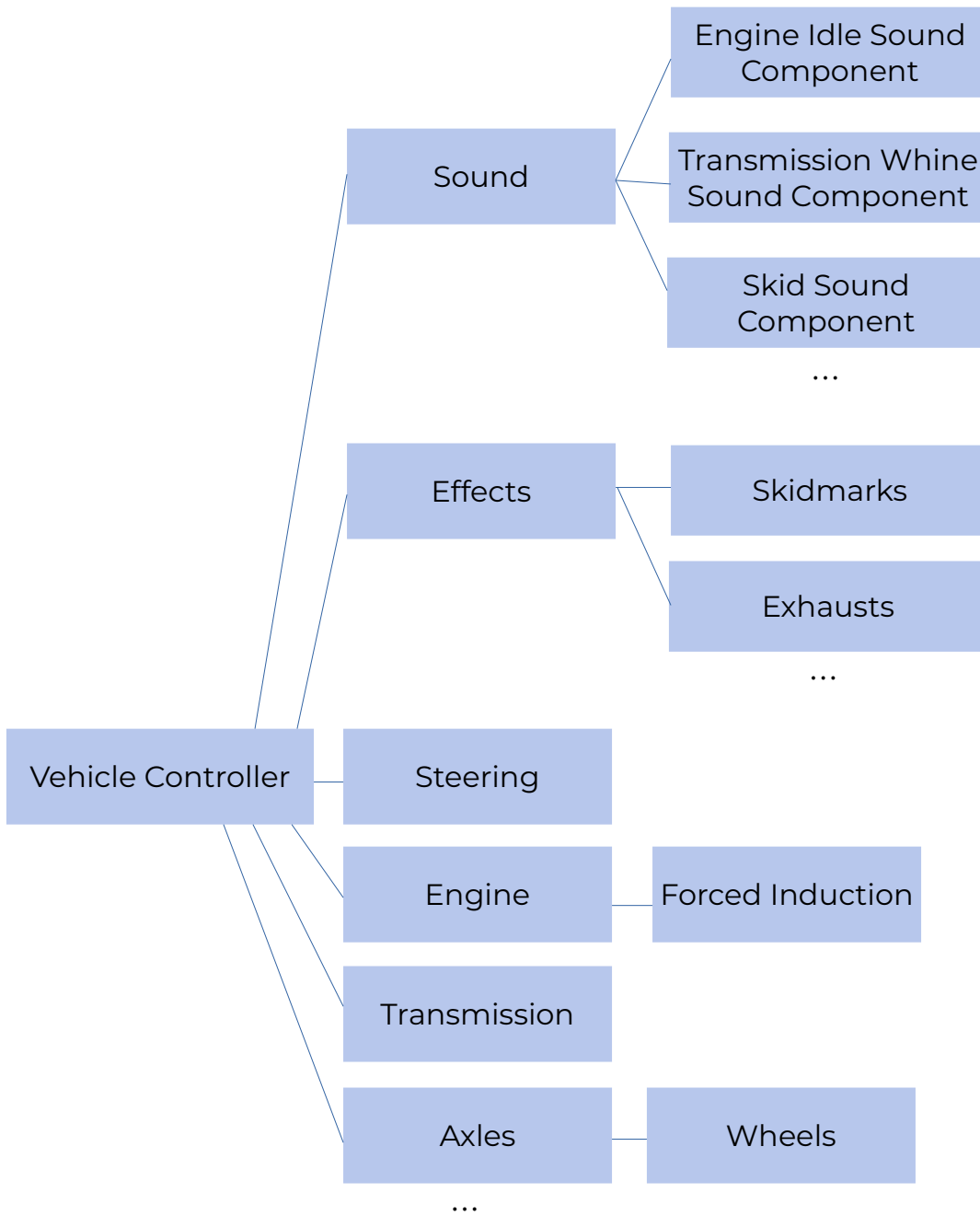
```
vehicleController.engine.maxPower;
```

This holds true for all the fields in the VehicleController.cs script. Check auto-generated documentation for class reference.

Each field has a Tooltip explanation. Because of this individual fields will not be explained in detail in this manual. If you want to know what a field does just hover the pointer over its name.

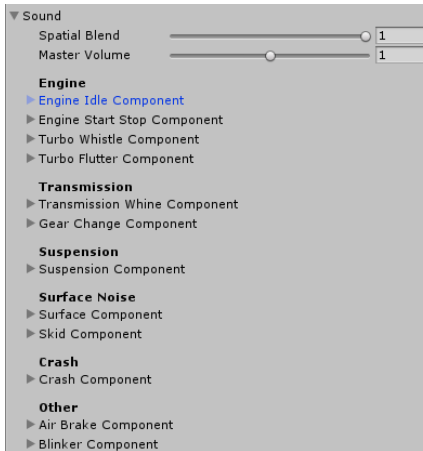
Script Structure

Check out the diagram below to help you with understanding how the VehicleController script is structured in general:



Sound

Sound section contains all of the fields related to vehicle's sound effects. Each field affects only the vehicle to which the script is attached. If you want to modify audio output for all the vehicles you can use *VehicleAudioMixer* ("Resources/VehicleAudioMixer.mixer") to which all out the sound output from each vehicle is routed.



Audio Sources are not added manually but are instead generated by the script when scene is started.

Each sound component inherits from *SoundComponent* class and thus has common fields: *volume*, *pitch* and *clips*. Some sound components also have additional fields such as volume or pitch range. General explanation for those fields is:

- Volume – volume. In cases where there is also a volume range option volume represents base volume to which the range will be added. Changing this value at run time through the VehicleController inspector will not necessarily change the volume of the AudioSource as most sound components set their volume and pitch only when AudioSource is created (at Start). You can still adjust the volume at the runtime through scripting or by adjusting the AudioSource directly – just note that the value will not be kept when exiting play mode.
- Pitch – same as volume, just for pitch.
- Volume Range – range which can be added to the (base) volume based on the state of the part of the vehicle sound component represents.
- Pitch Range – same as volume range, just for pitch.

- Clips – A list of Audio Clips. Depending on sound component either the first clip will be used (e.g. engine idle component) or one of the clips will be chosen at random (e.g. gear change or crash components). In general, sounds that are repeated or triggered accept any number of clips of which one will be chosen at random when the sound is triggered – hover the mouse over the component name (e.g. “Air Brake Component”) to see the tooltip explanation and if multiple clips can be used. If you use multiple clips on a component that supports only one, only the first one will be used. If you assign no clips (list size is 0) sound component will be turned off.

Skid sound effects are directly affected by the Forward and Side Skid threshold at the bottom of the inspector under the dropdown “General”.

Sound components that are wheel-specific will generate an Audio Source for each wheel individually (skid, surface noise, etc.).

If you want to add your own sound effect check out the *Sound* and *SoundComponent* classes.

Effects

There are three visual effects supported in v1.0, but more will be added with new updates.

Skidmarks

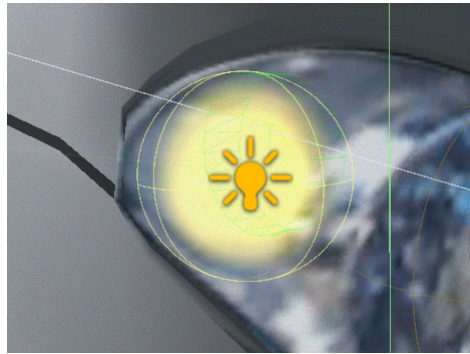
Skidmarks are generated when wheel skids / slips over the hard surface. To define when the wheel is slipping you can adjust the Forward and Side Slip Threshold at the bottom of the VehicleController inspector under “General” dropdown.

The most important field here that needs explanation beyond the tooltip is “Persistent Skidmarks” option. Default behavior for skidmarks is to delete the oldest skidmark (under skidmark here I am talking about a set of two triangles that together create a rectangle which is in this context called a skidmark). This way an effect similar to the old snake game will be achieved with the length of the whole skidmark trail always about the same length. Length of this “trail” can be adjusted by changing the *Max Marks Per Section* field – this is the maximum number of skidmarks that will be kept IF the persistent option is disabled. If the persistent option is enabled each time number of skidmarks reaches *Max Marks Per Section* number a new skidmark object will be generated and the old section of the skid trail will be left intact. This way old skidmarks are never deleted. With time this will cause the number of triangles in the viewport to increase and so there is also a *Persistent Skidmark Distance* option. When skidmark’s distance from the vehicle that created it goes beyond this value skidmark will delete itself.

Lights

All lights inherit from the same class and therefore have the same fields. The only options available here are:

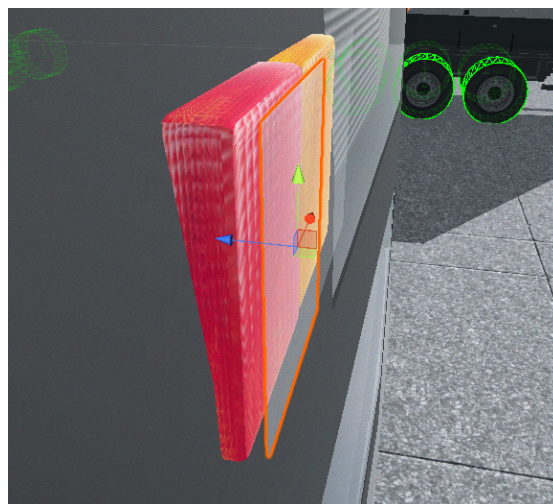
- **Light Sources** – can be any Unity light source, type does not matter. Main use it to create spotlights for low and high beam lights. If a model is low poly model with no separate light meshes point lights can also be used to create an imitation of e.g. stop light. In the following picture a point light is used on a single-mesh vehicle to create an imitation of a blinker:



If you want to add some realism to your low or high beam spot lights you can add the included “Headlight Cookie” to them.

- **Light Mesh Renderers** – Accepts any mesh renderer that uses standard shader. When light is turned on *Emission* property of the shader will be turned on. To set up a mesh with emission tick the emission option in the shader options and adjust the color and intensity of the emission to your liking. When done deactivate the emission property.

Light Mesh Renderers can be used on vehicles that have separate meshes for lights. It can also be used as an overlay over the light of a primitive shape (bus and semi trailer in the demo scene) when the mesh is not separate, such as in the following example where a transparent quad with emission set up is placed over an actual brake light which is part of a single vehicle mesh:



Exhausts

Exhausts are just regular particle emitters. Exhaust prefab can be found in the “Effects/Particles/Prefabs” directory. You can have as many exhausts as you want on a single vehicle. To add an exhaust drag the prefab into the scene as a child to your vehicle and position it as you want. Then, add the object to the particle systems list. You do not have to use the prefab and you can modify the particle system in any way you want.

Some parts of the particle systems are adjusted through the script – things like emission rate at idle, under load and color of the particles. All of the exhaust-related settings can be adjusted under Effects→Exhausts dropdown in the vehicle controller inspector. Check Tooltips for more details on each field.

Steering

Steering is also adjusted on per-vehicle basis. This is because not all types of vehicles have same steering parameters and thus global steering adjustment would not make sense.

NVP features changeable steering angles for better vehicle stability at higher speed. Steering angle value is interpolated between low speed and high speed angle, where interpolation starts at 0 and ends at crossover speed. This means that at 0 m/s steering angle will be exactly low speed angle, at crossover speed and above exactly high speed angle, at half the crossover speed maximum steering angle will be midpoint between low and high speed angle, etc.

Since v1.5 *smoothing* option has been replaced with *Degrees Per Second Limit* field. Smoothing was introducing lag into the input which is always unwanted and was not consistent. The new field states how many degrees per second the steer angle will be able to change per second. So if the maximum steer angle is 30 degrees and this setting is set to 90 degrees vehicle will need 1/3 of a second to achieve a full lock starting from natural steering position. Useful for trucks, buses and other heavy vehicles. Also works on tracked vehicles.

If you vehicle has a separate steering wheel object you can add it to the *Steering Wheel* field. Steering wheel object will be rotated around it's Y axis so make sure that rotation and pivot point of the steering wheel are correct – check the first section of this manual. Steering wheel turn ratio is a multiplier by which the steering angle will be multiplied and the steering wheel object will be rotated by the result. So if you want your steering wheel to rotate more, just increase this number.

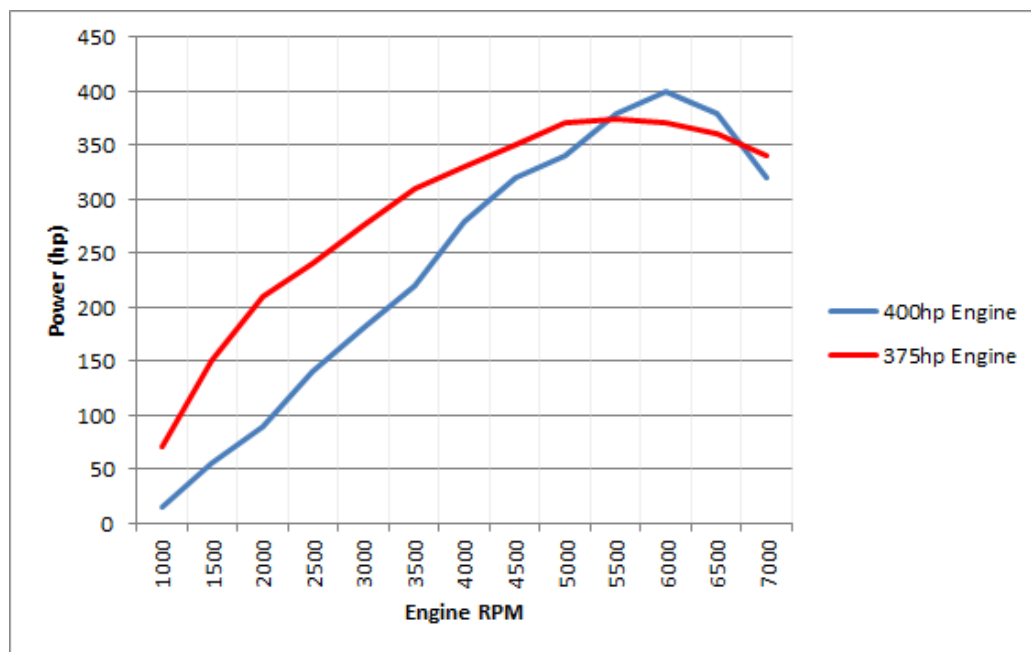
Tracked Steer Intensity is valid only for tracked vehicles and determines the percentage of the braking force set in *Brakes/Max Torque* will be used to brake the track on the side of the vehicle the vehicle is turning towards. Lower values will make tracked vehicles rotate in a bigger circle.

Engine

Most of the fields in this part of the inspector are self explanatory. There are *Min* and *Max RPM* fields which indicate idle RPM and redline RPM respectively. There is also a *Max Power* field which indicates power at the top of the power curve in kW (multiply by 1.4 to get horsepower).

Max Rpm Change field is the first one that is not obvious. It indicates maximum RPM difference in one second, crudely imitating the weight of the flywheel an engine has. For trucks and similar heavy duty vehicles I would suggest a value of about 2000, for normal cars about 5000 and for race cars with light clutch 10,000+.

Power Curve is also an important field that severely alters the behavior of the vehicle. Value on the curve should be between 0 and 1 on both axes. X axis represents RPM percentage (0 is 0, 1 is max rpm) and Y represents power (1 is max power). There is torque curve adjustment as the torque curve is derived from the power curve and the engine's RPM. Here are two typical power curves for two different engines, simply for reference purposes (note that the power in the image is in hp while NVP uses kW, divide hp by 1.4 to get kW):



Throttle Smoothing was introduced with v1.5 to avoid jerky starts on high power / low gear ratio vehicles such as the truck in the demo scene. Setting tells vehicle how many seconds it will need to reach full throttle.

Forced Induction

At the end of the Engine section there are forced induction settings. Forced induction is entirely optional and can be used for imitating turbocharger, supercharger or just intake or engine fan noise. For everything except for turbocharger spool up time should be 0 as this way there will be no lag between engine's rpm and forced induction rpm which also affects sound effects. *Max Power Gain* multiplier is amount of power (0.4 = 40%) that will be added to the *Max Power* of the engine when forced induction is fully spooled up.

If you are using forced induction just for intake or engine fan noise set both spool up time and max power gain multiplier to 0.

Transmission

NVP uses gear ratios – just like the real transmission does. If gears are not set up properly the vehicle will not function.

To set gear ratios either *Forward Gears* or *Reverse Gears* list can be used for setting forward and reverse gear ratios, respectively. Size of those lists can be unlimited so vehicle can have multiple gears in both directions. First element in the list represents first (forward or reverse) gear, second element is second gear, etc.

Gear Multiplier field just multiplies current gear ratio from the Gears list to arrive at the total gear ratio. Also called final gear ratio. So if current gear has gear ratio of 4 and final gear ratio is 2, total gear ratio will be 8.

Target Shift Up and *Target Shift Down* points are points at which transmission will change gears if set to Automatic or Automatic Sequential. Transmission will shift at exactly those RPMs if the *Dynamic Shift Point* option is disabled (+/- the randomness if set by *Shift Point Randomness*). If enabled, script will adjust shift points according to engine state with randomness also applied. This means that the vehicle will shift later if under heavy acceleration or shift earlier if just rolling downhill. Shift up RPM will never exceed max RPM and shift down RPM will never go under min RPM.

Shift Duration determines how long shifting will take. While shifting engine will not be connected to the wheels and the shifting sound will be played. To reduce repetitiveness of the shifting two *Randomness* options have been added – shift point and shift duration. This number represents a percentage of the corresponding value which will be randomly added to the value. e.g. 0.1 = 10% so if shift duration is 1s it will actually be anywhere from 0.9s to 1.1s.

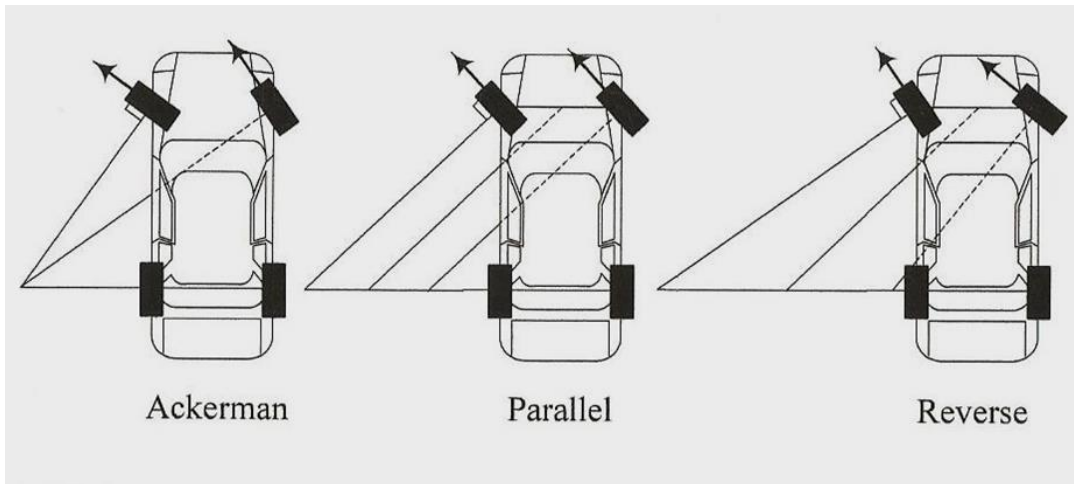
Post Shift Ban is used just so vehicles do not shift too often / hunt for gears – which can be irritating.

Since v1.1 NVP supports automatic and manual clutch operation. If set to automatic clutch vehicle will hold RPMs of the engine at the given *Target Clutch RPM* until RPMs received from the wheels catch up after which point RPMs of the engine will continue to rise (or fall) normally. If you want to use manual clutch untick the *Automatic Clutch* option and set the key / axis bindings in the input manager (Edit > Project Settings > Input) for “Clutch”.

Axles

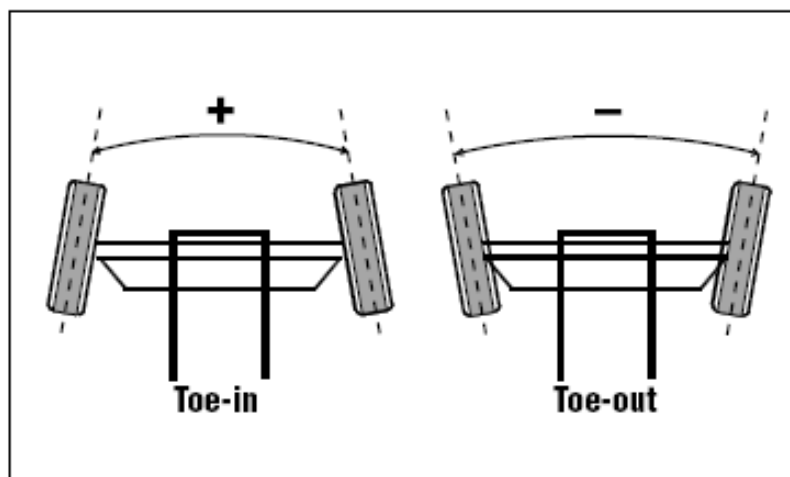
NVP supports any number of axles. You could have 1 or you could have 10. Axles are stored in a list and so you can set the size of the list to set the number of axles vehicle has. When initially setting up the vehicle or using the Reset function on the script axles will be assigned automatically. If you are adding additional axles or changing them after the script has been initialized you will have to assign left and right wheel controllers manually. All axle objects are instances of the same class so they all also have the same fields. Following geometry fields are available:

- **Ackermann Percent** – 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:



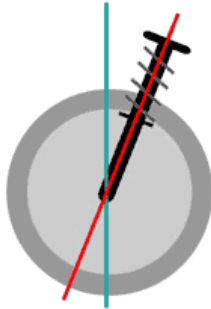
• T
o
e

Angle – toe angle in degrees.

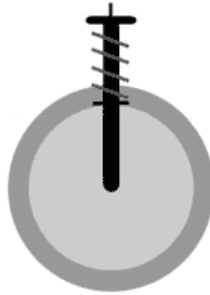


- **Caster Angle** – caster angle in degrees.

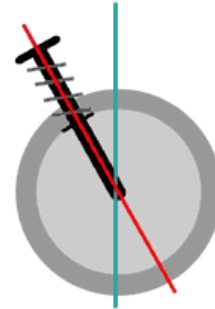
POSITIVE CASTER



NEUTRAL CASTER



NEGATIVE CASTER



← FRONT OF VEHICLE

- **Camber At Top** and **Camber At Bottom** – wheel camber angles at the top and bottom of the spring travel. Value will be interpolated in linear fashion between the two values. If you do not want the linear camber curve set both values to 0 and use the inspector of WC3D where you can adjust the camber curve. If you are using solid axle option camber settings will be overwritten with whatever correct camber for the axle is at that frame.
- **Anti-roll bar force** – prevents vehicle from rolling over. Before increasing the anti roll bar force try to adjust center of mass to be a bit lower. Also, do not set anti-roll bar force to a value much higher than the spring force of your vehicle.

The rest of the fields related to axles have detailed tooltip explanations and will not be further explained here.

Brakes

Only a few straightforward fields here:

- **Max Torque** – maximum brake torque that can be applied to the wheel. Each axle multiplies this number by handbrake coefficient or brake coefficient to get brake torque for that specific axle.
- **Friction Torque** – torque that is applied when vehicle is standing still so that the vehicle does not roll on its own with no throttle applied.
- **Smoothing** – time in seconds needed for brakes to reach full torque.
- **Air Brakes** – does not have any effect on braking, only on sound effects.

Driving Aids

Following driving aids are available:

- **ABS** – If set to 1 wheels will never slip while braking.
- **TCS** – If set to 1 wheels will never slip while accelerating.
- **Stability** – reduces vehicle's tendency to spin by multiplying the current friction curve by a linear curve originating in [0,0] and ending at [1, intensity] which means that the friction force gets higher with slip – you can also think of it as a dynamic lateral coefficient in the wheel controller settings. Can be used to get more arcade-y feeling vehicles. Does not work at low speeds to allow for doughnuts.
- **Drift Assist** – calculates the angle between the vehicle's forward direction and its vector and applies corrective force based on the value of the angle, effectively limiting the angle of the drift so that it does not turn into a spin. Higher value will allow less drifting while a low value will allow spins.

Damage

If enabled damage module will deform 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. Allowed damage number that can be adjusted to adjust allowed damage before vehicle stops functioning is actually a sum of all decelerations that were caused by crashing. 1000 damage is therefore equal to a crash with a peak deceleration of 1000m/s^2 . 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.

Mesh deformation that is the product of crashing is queue based. This means that only limited number of meshes will be deformed per frame. You can adjust this value 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 drop. This will depend on the device the final game will run on. Deformation radius, strength and randomness all affect how the mesh will be deformed – how much 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* field is the final field and it is simply a deceleration threshold above which crash will count as actual crash.

Ignore Tags list has been added in v1.4 and all the objects containing a tag from this list will not cause damage to the vehicle.

Fuel

Fuel consumption is calculated from the energy content of the fuel and the efficiency of the engine. If used with keyboard fuel consumption might appear a bit high and that is since engine is always under 100% throttle. If you want your vehicle to consume less fuel simply increase the efficiency.

Capacity and *Amount* fields are in liters.

If you want to display fuel consumption to the player you can do so in l/100km, km/l and US mpg. Check the Fuel.cs script for implementation details and the available functions.

Rigging

Rigging enables you to use a rigged model together with NVP. Your model will need to have either per-axle or per-wheel handles which will move corresponding parts of your vehicle when dragged. You can try it out first by hand and then assign the handle objects to the script. If solid axle is enabled and axle handle is assigned, handle will be always moved so that it is midpoint between the wheels and looking at both with its ends.

Flip Over


Flip over, when enabled, flips the vehicle over when the vehicle has been on its side or roof for the *Timeout* amount of time. Except for timeout setting there is also an *Allowed Angle* setting which is minimum angle between the surface normal and vehicle transform's up direction at which flip over will activate. *Max Detection Speed* setting determines the maximum speed at which flip over will be able to activate and *Rotation Speed* is speed in degrees per second with which the vehicle will be rotated back to its proper rotation.

Note: Flip over function does not prevent vehicle from ending up upside-down, just rotates the vehicle back if this happens.



Trailer

Trailer in NVP is just another vehicle. Only difference is the “Is Trailer” option. When ticked some fields in the inspector will change.

Vehicle’s trailer settings:

Is Trailer	<input type="checkbox"/>
Attachment Point	None (Game Object) 
Acceptable Trailer Tag	Trailer
Attach Distance Threshold	0.5
Attach On Play	<input type="checkbox"/>
Joint Break Force	Infinity

Trailer’s trailer settings:

Is Trailer	<input checked="" type="checkbox"/>
Attachment Point	None (Game Object) 
Trailer Stand	None (Game Object) 

Since trailer uses the same script as all the vehicles it also has most of the functionalities enabled. Actually, only the things like transmission and engine are disabled. It still produces effects, can be damaged, etc. and since all the input is routed from the towing vehicle it also has functional lights, blinkers and can even steer.

First field – *Attachment Point* – exists in both trailer’s and vehicle’s inspector. Attachment Point can be any game object, and in case of trailer it will have to have a *Acceptable Trailer Tag*. Let’s say you have two types of vehicles in the scene that both can attach trailers of different types, e.g. a semi and a car. You would make two new tags: “SemiTrailer” and “CarTrailer”. Attachment point game object on all car trailers would have “CarTrailer” tag and attachment point on all semi trailers would have “SemiTrailer” tag. Also, all your semis that can attach semi trailer need to have “SemiTrailer” set as *Acceptable Trailer Tag*. Same for cars, just with “CarTrailer” tag. When a vehicle with attachment point object gets near to the trailer that also has an attachment point object, *trailerInRange* variable will be set to true. At this point you can use `AttachTrailer()` function to attach the trailer. You can modify the maximum distance between the two attachment points at which *trailerInRange* will be set to true by changing the attach distance threshold value.

Ground Type Detection

Ground detection is a component that handles all of the surface-specific effects and sounds, ground type detection and surface-specific friction settings.

Ground detection is an optional part of the vehicle that is not visible in the vehicle controller's inspector. Unlike other vehicle settings, ground detection component adjusts settings for all the vehicles in the scene and only one ground detection component should be present (or 0 if you do not want to use ground detection).

You can use a “_VehicleManager” prefab which already has ground component attached to it, as well as other useful components, or you can add ground detection component to any object in the scene. If there is no ground detection in the scene surface effects and sounds will not work and you will get a warning. Vehicles will find ground detection automatically and there is no need to assign it.

Ground detection component consists of two fields for dust and smoke prefabs. Those can be found in the prefabs folder and are just game objects with particle emitters that will be added to the wheel objects on start. You can use any empty object with particle emitter you want and assign it here.

Below the two prefab fields is a ground entity list. This is a list of surfaces that appear in the scene. Surfaces are detected by either using terrain splat map data or by using object tags. Each surface (asphalt, gravel, sand, grass, etc.) can use any number of splat map texture indices and/or object tags where splat map index is a number of terrain texture in the terrain inspector:



You can use any number of ground entities, each with its own surface and skid sounds, different dust color and friction presets which ground detection loads from WC3D. Basic friction curve presets are already available and you can manually add new friction curves inside the WheelController.Firction.cs file, check WC3D manual for more info on that.

Vehicle Changer Script

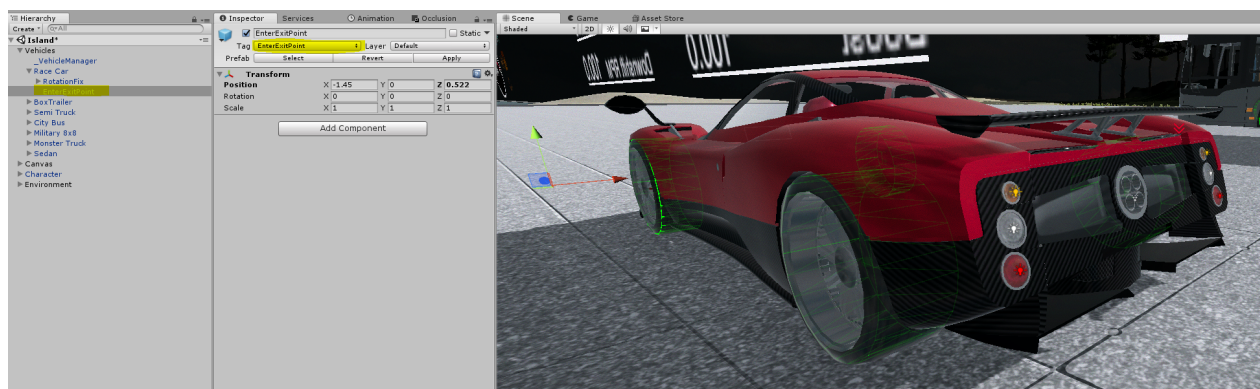
Vehicle changer script does just what it says it does. Before using it you will need to assign “ChangeVehicle” input if you are using desktop input manager script provided with this package or assign a GUI button if you are using the mobile input manager.

Vehicle changer will start with the first vehicle on the list and cycle through until it reaches the last vehicle and then will go back to the first one. You can either manually assign all the vehicles to the list or leave the list empty (size 0) and assign the tag set under field *Vehicle Tag* to all of the vehicles.

Character Vehicle Changer

If you want your character to be able to enter and exit vehicles you can add *CharacterVehicleChanger* component to the same object that contains the vehicle changer script – it does not matter which object that is. For each field there is a tooltip explanation, but some explanation about Enter Exit Points might be needed.

Any game object can be enter/exit point. You can use the vehicle itself as the point by setting the *Enter Exit Tag* value to your vehicle tag, e.g. “Vehicle”. *Enter Distance* will be measured from the character to the nearest enter exit point. This means that if you use a vehicle’s center as the point you will be able to enter the vehicle from any side as long as you are in the range. If you want your character to be able to enter the vehicle only from e.g. driver’s side you can place an empty object near the driver’s door and assign the value from the field *Enter Exit Tag* as the tag of that object. This is the way it has been done in the demo scene:



The

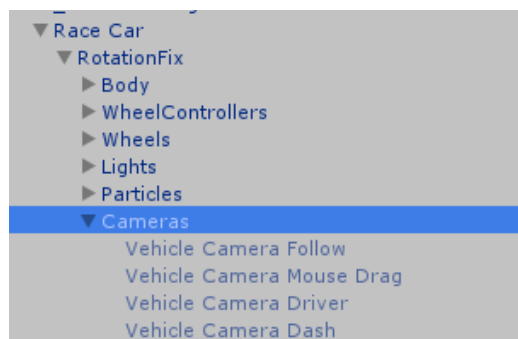
character will only be able to enter when distance to the point in picture is lower than *Enter Distance*. There can be multiple enter exit points. Characters will exit at the point where they entered the vehicle.

Cameras

Three camera scripts are included in the package plus the camera changer. You can use any camera you want with the vehicle controller – including the ones in the standard assets. Camera changer will also work with any camera script.

Camera Changer Script

Camera changer script is quite simple. It uses a list of objects that each contain a Camera component and cycles through them in order, starting with the value *Current Camera*. Camera hierarchy should look something like this but it is not imperative that cameras be children of the object containing Camera changer component (“Cameras” object in this case):



If you want cameras to be detected automatically set the size of *Vehicle Cameras* list 0 zero and set the value of the field *Camera Tag* to your cameras as a tag. Also, if you are using automatic detection of cameras all the cameras that you want to be detected need to be children of the object containing camera changer script or they will not be found. It is suggested to assign cameras manually to ensure the correct changing order.

Make sure that all the objects containing cameras are not active before starting the scene. Script will activate the correct camera when you enter or switch to the vehicle. If all camera objects are active at the start of the scene you will get multiple Audio Listener warning.

Of course, you can use any camera system with vehicle controller and are not limited to the included one.

Three cameras are included: **CameraFollow**, **CameraMouseDrag** and **CameraOnboard**.

Other Scripts

Vehicle

Center Of Mass

Center of mass is a required component for vehicle controller and will be added when you add vehicle controller to the vehicle. If you tick “Show COM” a green sphere will be drawn at the current center of mass (requires rigidbody and colliders to be set up). The only option center of mass script offers is *Center Of Mass Offset*. This is an offset from the Unity calculated center of mass which is generally too high as Unity calculates center of mass as a center of volume. Since cars have most of their weight in the engine, transmission and other fairly low mounted components this center of mass is not adequate and will need to be adjusted. This holds true for default wheel collider (and is mentioned in Unity’s documentation) and also holds true for Wheel Controller 3D that this package uses. By default vehicle controller will set the value of the offset to be exact center between all the wheel controller objects. For sports cars you can lower it a bit from that value and for other vehicles you can leave it as-is or rise it. If you raise it too much you might induce oscillation in case the fixed time step is too large.

Downforce

Downforce is also a fairly simple script and is optional. Any number of downforce points can be defined with the *Max Force* that will be applied at the *Max Downforce Speed*. Max downforce speed should be set a bit higher than what the vehicle is capable of going – just note that the speed is m/s – multiply the value by 3.6 to get km/h or by 2.5 to get m/h. Force applied will be increased in exponential fashion. If you set downforce value too high (value is in Newtons) your vehicle might get hammered into the surface. Suggestion is to use up to half the spring strength as set in the wheel controller per each axle. Check the “Race Car” in the demo scene as it has downforce script attached and set up.

GUI

Dash GUI Controller

You can use Dash GUI Controller script to control you vehicle's dash gauges and dash lights – be it inside the vehicle or on the HUD (demo scene uses it for both). Both analog and digital speed and rpm gauges can be set, along with blinker, light, driving aids and check engine indicators. All of the fields are optional so if you want to use only a single gauge or indicator you can.

Analog Gauge

Controls analog gauges. Check “GUI/Prefabs” folder for speed and RPM analog gauge prefabs. Following fields are available:

- **Max Value** – value at the end of needle travel, at the *end angle*.
- **Start Angle** – angle of the needle at the lowest value. You can use lock at start option to adjust this value while in play mode.
- **End Angle** – angle of the needle at the highest value. You can use lock at end option to adjust this value while in play mode.
- **Needle Smoothing** – smooths the travel of the needle making it more inert, as if actually had some mass and resistance.
- **Lock at Start / End** – locks the needle at start or end angle while in play mode.

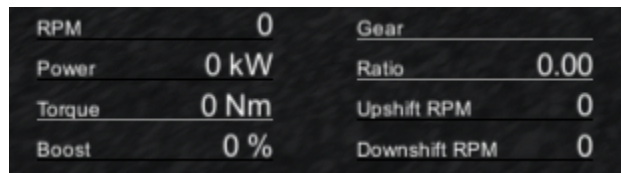
To adjust the needle travel you can enter play mode and the tick the one of the lock options. While the needle is locked at start or end adjust the start or end angle until needle is where it should be at the start position. If you are replacing background or needle graphics or making your own analog gauge object make sure that the needle rotates around center properly (if not you might want to adjust pivot point of the rect) and that it is a child of the object containing analog gauge script as the script will search for the child with the name of “Needle”.

In the picture below you can see two analog gauges added to the World Space canvas inside the sedan demo vehicle.



Digital Gauge

Digital Gauge consists of title, readout and line (which is optional).



RPM	0	Gear	
Power	0 kW	Ratio	0.00
Torque	0 Nm	Upshift RPM	0
Boost	0 %	Downshift RPM	0

As with analog gauge you can find digital gauge prefab inside “GUI/Prefabs/” directory.

Digital gauge accepts:

- Numerical values – can be smoothed and also progress line can be shown.
- String values – just a piece of text.

If you want to use numerical value tick “Display Numerical”.

- **Format** field accepts formatting that will be used when converting number to string, e.g. “0.0” for one decimal place or “0.000” for three.
- **Smoothing** field will smooth out the numerical value so it is easier to read. Number represents time in seconds needed to reach the set value.
- **Unit** field accepts a string which will be added after string or numerical value.
- **Max Value** is only required if using progress line to indicate at which value line will be fully filled.

If you do not want to use any lines delete both Line and LineBG objects.

Dash Light

A very simple script which attaches to any canvas Image object. It will remember the initial color of the image and then toggle between it and white color when *Active* property is set to true or false.

Compatibility With Other Assets

CTS (Complete Terrain Shader)

To make CTS work properly with NVP, untick *Strip Textures* option inside CTS profile:



IK Driver Vehicle Controller

Manual for integration of IK Driver Vehicle Controller with NVP is available on the IK's asset page.

FAQ

1) I am using a custom mesh / Easy Roads 3D / quad for a terrain and the vehicle keeps randomly jumping.

- To solve this disable the “Queries Hit Triggers” and “Queries Hit Backfaces” under physics options.

2) I dragged a prefab into an empty scene and it does not react to my input.

- This is because the vehicle itself only stores the input states, but does not set them. To make the vehicle move attach Desktop Input Manager to the vehicle. You can leave all the fields empty. The other option is to use the vehicle changer if you have multiple vehicles in the scene, check the beginning of the manual for step-by-step guide about this.

3) Vehicle jumps when wheel gets hit from the side / top or something falls on it.

- Check the included Wheel Controller 3D manual and it's FAQ section for more info about this. In short: enable rim collider option inside WC3D inspector.

4) I am getting errors about inputs not being set up.

- Copy the “InputManager” file from ProjectSettings inside the VehiclePhysics/ folder to your project's ProjectSettings. Note that this will overwrite any previously setup inputs.

5) I just imported the asset and I get a few errors about Standard Assets missing.

- Check the readme.

6) Blinkers or other emissive meshes are not working in build but they work inside editor just fine.

- Add “Standard Shader” to Project Settings → Graphics → “Always Included Shaders” list.

If you encounter any problems, need help setting up or would like to see some feature in the next version you can contact us at:

nwhcoding@gmail.com