Quick Start Guide

Importing the Asset

- If upgrading between major versions (e.g. 1.1 to 1.2) make sure to check the Upgrade Notes.
- When importing a message about Package Manager dependencies will pop up. Click "Install/Upgrade". This will install the required InputSystem (the package needs to be present, use of it is optional). If the target is Windows 32-bit the InputManager will throw a warning about having to switch to Windows 64-bit build target (`File > Build Settings > Architecture`).
- After following the steps that pop up and the packages are installed "Import" button might need to be pressed again if the asset has not been imported. After import there might be a few model-related warnings. Ignore those.
- Asset is now ready to go. Try entering play mode in one of the demo scenes.
- To get the look of the demo scene set the `Color Space` under Project Settings > Player to `Linear`.

Vehicle Model Requirements

- A vehicle model with proper Unity rotation and pivots. Make sure your model uses the following axes: Z - Forward, Y - Up and X - Right. If your model has incorrect rotation and/or pivots check this guide.

An example of a vehicle model with Unity-correct rotation.

- Separate wheel objects for each wheel, ideally with `wheel` in the name (script uses the name to detect the wheels automatically, case is ignored).

Example vehicle model hierarchy.

Physics Settings

Adjusting physics settings is optional but it can increase the simulation quality of the vehicles:

- For PC games it is recommended to to change the `Fixed Timestep` from default 0.02 (50Hz) to 0.01667 (60Hz). This will result in smoother physics behavior. If the game is aimed at mobile devices this value can be set to higher than 0.02 (e.g. 0.025 or even 0.03) but can cause instability at high speeds and vehicle jitter if too high. This setting affects performance.

Vehicle Setup Wizard

To set up a vehicle automatically and skip the steps below, `VehicleSetupWizard` can be used. Make sure you vehicle has correct pivots and rotation first. Then, attach `VehicleSetupWizard` to the root object of the vehicle - where the Rigidbody would normally be placed. Follow the steps from the wizard inspector and the vehicle will be up and running shortly.
`VehicleSetupWizard` logs all of its actions in the console along with any errors and/or warnings. Make sure to check for those and correct them manually if needed.

Vehicle Setup Wizard does NOT set up scene input. For that check Input Setup page.

Rigidbody

- Add a `Rigidbody` component to the vehicle root, `Car` in the image above. Set the mass to a reasonable value, e.g. 1400.

Colliders

- Add a `Collider` (BoxCollider, MeshCollider, etc.) to the vehicle.
- Create and assign a new layer (e.g. `WheelControllerIgnore` to all vehicle colliders. This will prevent wheels detecting the vehicle as ground. Do not use `Default` layer or any of the ground layers or they will not be detected by the wheel!

WheelController

For a vehicle to function first of all it needs wheels. NWH Vehicle Physics uses included asset `WheelController` for this instead of default `WheelCollider`. For more info check out WheelController Setup page.

- `WheelController` needs to be attached to a separate object from the wheel. That object will act as a suspension anchor and will represent the beginning of spring travel. Therefore, it needs to be positioned above the wheel center. Easiest way to create those objects is to duplicate the existing wheel objects, remove all components except `Transform` from them and move the duplicated objects slightly above the wheel center.


Vehicle hierarchy with WheelControllers and positioning of front left WheelController on the example vehicle.

- Add `WheelController` component to the objects created in the previous step. `WheelController` will initialize itself with default values and will try to set the necessary fields automatically. Check if the following fields are set:
    - `Parent` - root object of the vehicle. Must have `Rigidbody` attached. *Car* for this

example.

- ○ `Visual` - wheel model. *Wheel_FL*, *Wheel_FR*, etc. for this example.
- Turn on Gizmos and select the wheels. The wheel gizmo and suspension gizmo will appear. Adjust `Width` and `Radius` of `WheelController` until they fit the wheel model.
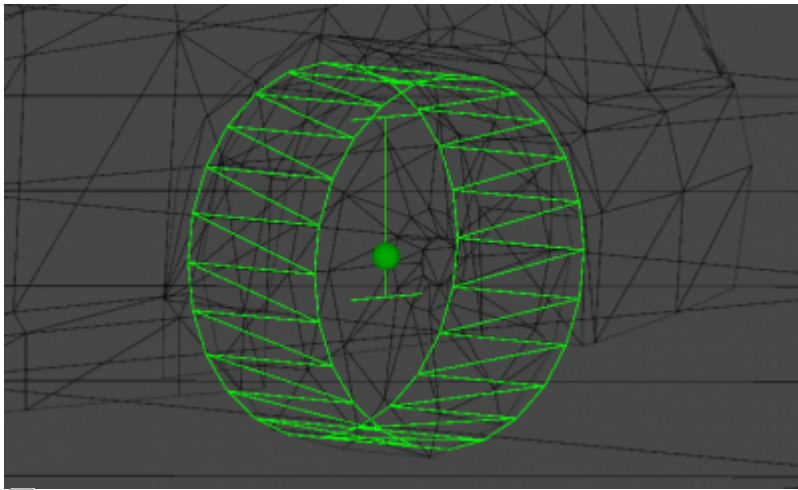
Wheel gizmo. The vertical 'I' represents the suspension travel while the cylinder represents the wheel.

- Press play. Vehicle suspension is now functional. If you experience vehicle flying into the air check that you have a `Rigidbody` set up with correct mass and at least one `Collider` attached to the vehicle. For more info check *Troubleshooting* section of [WheelController Setup](#) page.

Input

- Check [Input Setup](#).

The guide above is recommended for understanding of how input works. Below are the 'quick' steps needed:

- If using InputSystem (new Unity input package) add the following components to any object in the scene, e.g. *SceneManager*:
    - ○ Add `InputSystemVehicleInputProvider`.
    - ○ Add `InputSystemSceneInputProvider`.
- If using InputManager (old/classic Unity input) add the following components to any object in the scene, e.g. *SceneManager*:
    - ○ Add `InputManagerVehicleInputProvider`.
    - ○ Add `InputManagerSceneInputProvider`.

Cameras

- Add a `Camera` as a child of vehicle object (right click ⇒ Camera) and attach `CameraFollow` component to it.
- Assign the vehicle as `Target`.
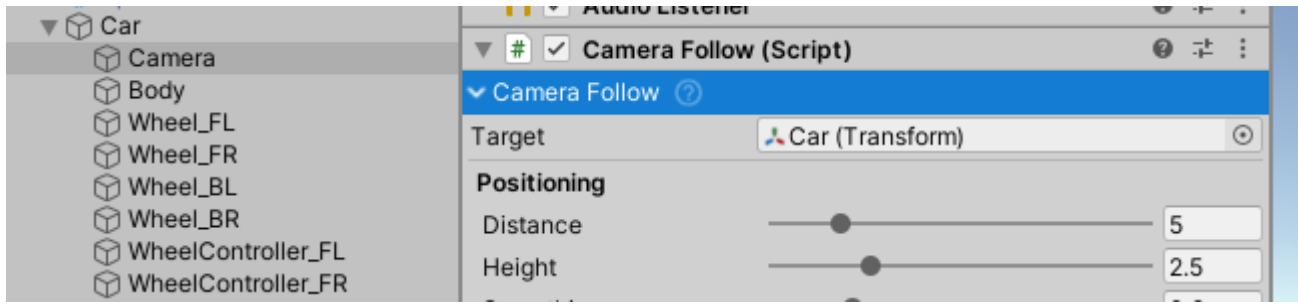
Example single camera setup.

VehicleController

- Add `VehicleController` component to the vehicle root - where `Rigidbody` is.
- `VehicleController` will automatically set up default values and automatically set up `Differentials`, `Wheels` and `WheelGroups`. Check console for output.
- Go to `Settings` tab and click on `Validate Setup` button to for automatic validity check.
- Press play. Vehicle will now function but will not respond to any user input.

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

Permanent link:
http://nwhvehiclephysics.com/doku.php/Setup/QuickStart

Last update: **2020/07/26 16:05**