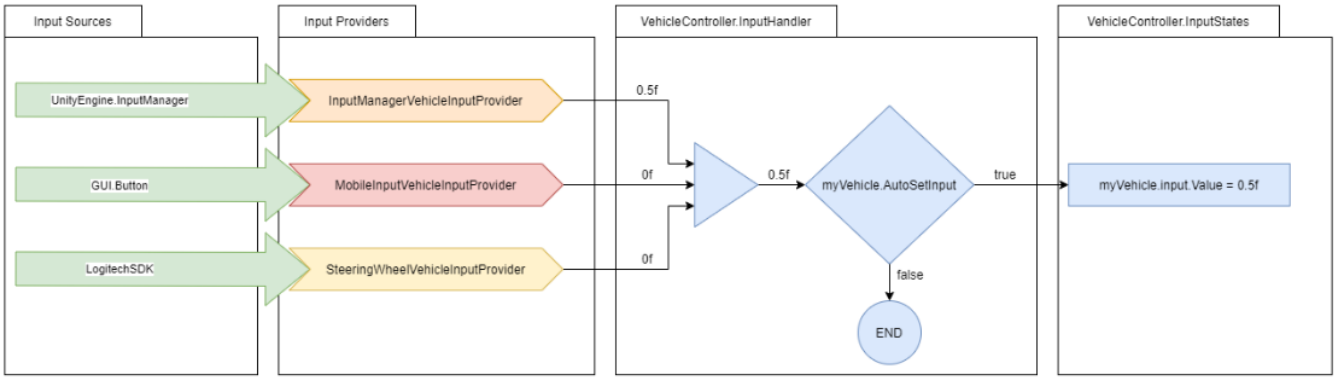
Make sure to use *Project Settings* ⇒ *Player* ⇒ *Input Handling* ⇒ *Both* or multiple errors will pop up.

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.

Multiple `InputProviders` can be present at the same time, meaning that `MobileVehicleInputProvider`, `InputSystemVehicleInputProvider` and `SteeringWheelVehicleInputProvider` can all be present at the same time and the input will be combined from the mobile, keyboard/gamepad and steering wheel input.

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

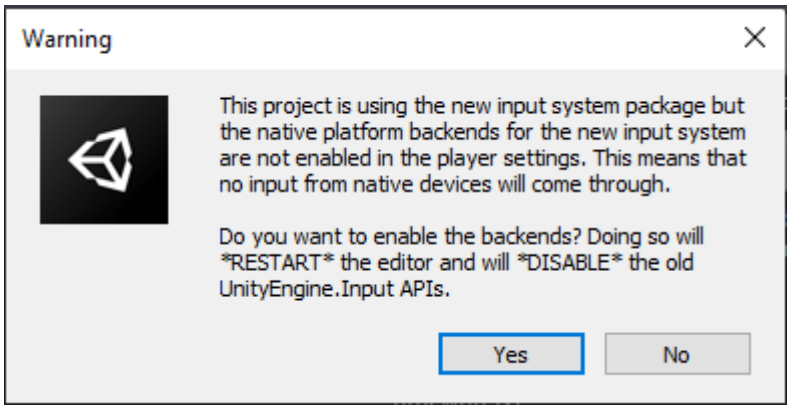
- A single *InputProvider* provides the input for all the vehicles so it is recommended to only have one *InputProvider* of particular type (e.g. one desktop and one mobile input provider - this will allow the vehicles to get both desktop and mobile input at the same time).
- Input providers just provide 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		
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.

Name	Type	Keyboard Defaults	Gamepad Defaults	Description
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* ⇒ *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.

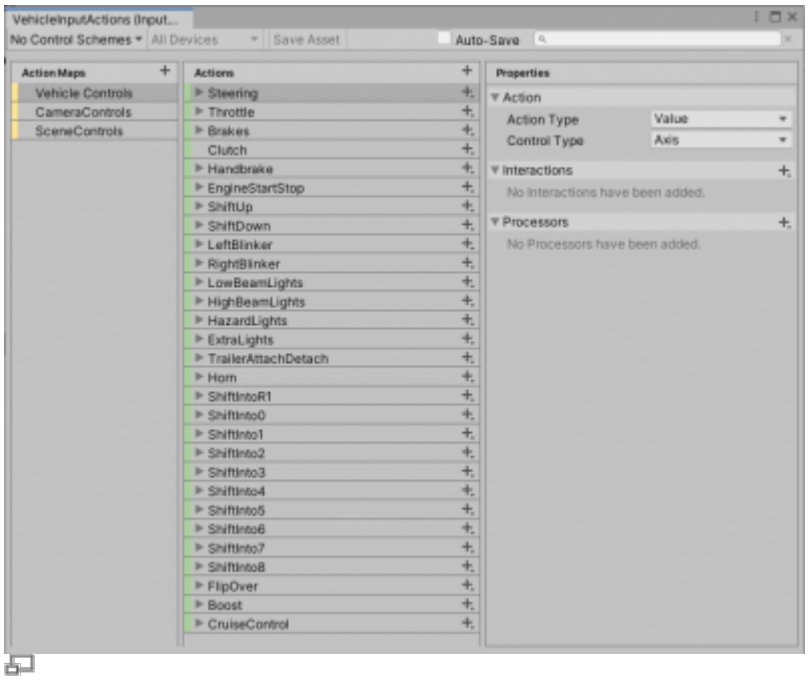
Scene Setup

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.

Rewired Input Provider

Since v1.7.1 NWH Vehicle Physics 2 also supports Rewired input.

Installation

- Download and import Rewired from the Unity Asset Store.

- Follow the Rewired install wizard. Make sure to leave both InputSystem and InputManager enabled. To use only InputManager, as recommended by Rewired, InputSystemVehicleInputProvider and InputSystemSceneInputProvider and their respective editor scripts need to be removed from the project first.
- With the Rewired imported and installed, double click on *NWH ⇒ Vehicle Physics 2 ⇒ Scripts ⇒ Optional Packages ⇒ Input ⇒ Rewired* to import files needed for Rewired/NVP2 integration.
- Open the newly imported Rewired folder and add `Rewired Input Manager` to the scene. It already contains the Rewired `InputManager`, as well as the `RewiredVehicleInputProvider` and `RewiredSceneInputProvider` needed for NVP2 control.
- Remove any existing `InputSystemVehicleInputProvider`, `InputManagerVehicleInputProvider`, `InputSystemSceneInputProvider` and `InputManagerSceneInputProvider` scripts from the scene to prevent input duplication. All these can technically be present at the same time but there are no benefits to it and duplicate inputs might happen.

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): <code> float throttle = InputProvider.Instances[0].Throttle; </code> 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.autoSetInput = false; // Tells the vehicle to stop
```

```
retrieving input values automatically.
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. `ExampleVehicleInputProvider` and make it inherit from `VehicleInputProvider` class:

```
public class ExampleVehicleInputProvider : VehicleInputProvider{}
```

- Override the methods that you want to use, e.g. `GetThrottle()`.
- 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.
- The newly created `ExampleVehicleInputProvider` now can be added anywhere in the scene as the included `InputSystemVehicleInputProvider` or `InputManagerVehicleInputProvider` would be.

Example custom input script is below. Note that to reference NWH.Common.Input the script will either

need to be generated inside Scripts > Input folder of the asset or referenced inside the project .asmdef file if the script is placed outside of the VehiclePhysics directory. For more info about assembly definitions check out the Import guide.

```
using NWH.Common.Input;
using UnityEngine;
using UnityEngine.InputSystem;

/// <summary>
///     Example class for handling input.
/// </summary>
public class CustomVehicleInputProvider : VehicleInputProviderBase
{
  public override void Awake()
  {
    base.Awake();
    // Your initialization code here (if needed).  Just a standard
MonoBehaviour Awake().
  }

  public void Update()
  {
    // Your Update() code here (if needed). Just a standard MonoBehaviour
Update().
  }

  public override Throttle()
  {
    // Return your custom value here, example:
    return 0.5f; // Replace this line with e.g. player.GetAxis("Throttle")
for Rewired.
  }

  public override Steering()
  {
    // Return your custom steering value here.
    return 0.123f;
  }

  // ...and so on. Override the functions that you want to use. If you do
not need Clutch() for example,
  // do not override it.
}
```