Easy Vehicle System

Complete driveable vehicle solution for Unreal Engine 5. Create fully functional vehicles in minutes - no coding required!

v1.0.0 UE 5.3 - 5.6+ 100% C++

📦 Installation

💡 **Before you start:** Make sure you have Unreal Engine 5.3 or newer installed. This plugin does NOT work with UE4.

1

Download & Extract the Plugin

After purchasing, download the `EasyVehicleSystem_v1_0_0_UE5_6.zip` file.

Extract the ZIP file. You should see a folder called `EasyVehicleSystem`.

⚠️ **Important:** Do NOT extract inside the ZIP! Right-click → Extract All, or use 7-Zip/WinRAR.

2

Copy to Your Project's Plugins Folder

Navigate to your Unreal Engine project folder. This is where your `.uproject` file is located.

If you DON'T have a Plugins folder:

1. Create a new folder called `Plugins` (capital P!)

Copy the entire `EasyVehicleSystem` **folder** into your `Plugins` folder.

Your folder structure should look like this:

```
📁 YourProject/ ├── 📁 Content/ ├── 📁 Config/ ├── 📁 Plugins/ ← Create this if it doesn't exist! │ └── 📁 EasyVehicleSystem/ ← Put the plugin HERE │ ├── 📁 Source/ │ ├── 📁 Resources/ │ ├── 📄 EasyVehicleSystem.uplugin │ └── 📄 ... └── 📄 YourProject.uproject
```

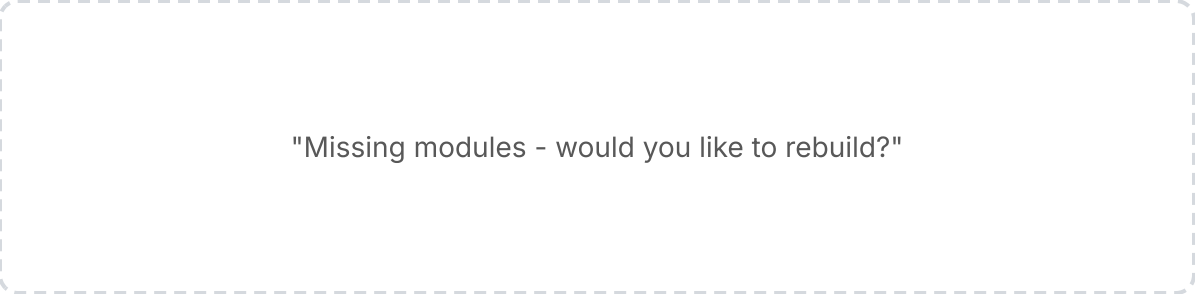
🚫 **WRONG:** Don't put it in `Engine/Plugins` or inside the Content folder!

3

Open Your Project in Unreal Engine

Double-click your `.uproject` file to open Unreal Engine.

If you see this popup:



Click **YES**! The plugin needs to compile for your project.

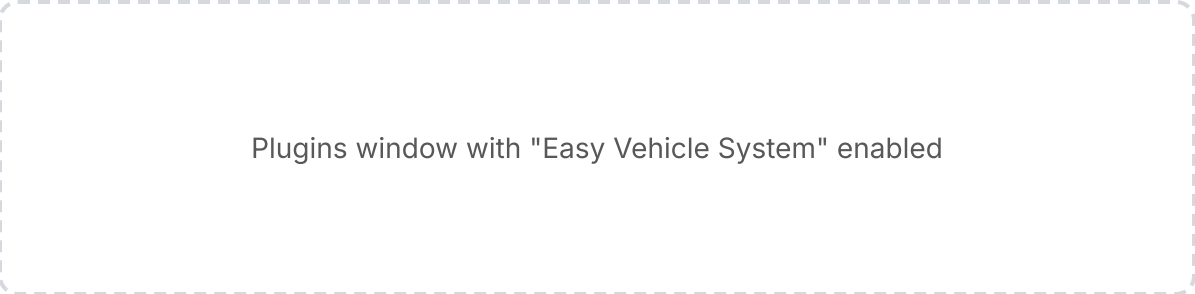
⏱️ **First time?** Compilation may take 1-5 minutes depending on your PC. Grab a coffee! ☕

4

Enable the Plugin (If Not Already Enabled)

In Unreal Engine:

1. Go to **Edit** → **Plugins**
2. In the search box, type `Easy Vehicle`
3. Make sure the checkbox is ✅ **Enabled**
4. If you had to enable it, click **Restart Now**



5

Verify Installation ✅

Let's make sure everything works!

Test 1: In Content Browser, click **Settings** (gear icon) → Enable **"Show Plugin Content"**. You should now see `EasyVehicleSystem Content` folder.

Test 2: Create any Actor Blueprint, click **Add Component**, and search for `EVS`. You should see:

- EVS Audio Component
- EVS Lighting Component
- EVS Interaction Component
- EVS Door Component
- EVS Fuel Component
- EVS HUD Component
- EVS Wheel Component

> 🎉 **If you see all components, installation is complete!** Continue to create your first vehicle.

Creating Your First Vehicle

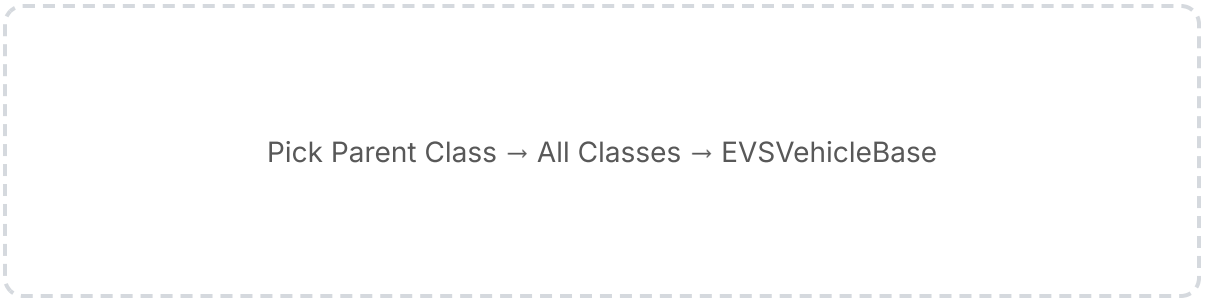
Let's create a working vehicle from scratch. This guide assumes you have NO vehicle mesh - we'll use UE5's default vehicle template!

Goal: In 10 minutes, you'll have a driveable car with engine sounds, lights, horn, and enter/exit functionality.

1

Create a New Blueprint

1. In Content Browser, **right-click → Blueprint Class**
2. In the popup, click **"All Classes"** at the bottom
3. Search for `EVSVehicleBase`
4. Select it and click **Select**
5. Name it `BP_MyFirstVehicle`



Can't find EVSVehicleBase? The plugin isn't installed correctly. Go back to Installation step.

2

Open the Blueprint

Double-click on `BP_MyFirstVehicle` to open the Blueprint Editor.

You'll see the component hierarchy on the left:

```
BP_MyFirstVehicle (Self) ├── Mesh (Skeletal Mesh) ← Your car model goes here ├── VehicleMovement ← Chaos physics (auto-configured) ├── SpringArm ← Camera boom │ └── Camera ← The camera ├── InteriorCameraPoint ← First-person view position ├── AudioComponent ← All sounds ├── InteractionComponent ← Enter/exit detection └── LightingComponent ← All lights
```

Everything is already set up! You just need to add a car mesh.

3

Add a Vehicle Mesh

Option A: Use UE5's Built-in Vehicle (Easiest)

1. Enable the **Vehicle Template** plugin:
   - Edit → Plugins → Search "Vehicle" → Enable "Vehicle Template"
   - Restart editor
2. In your Blueprint, select the **Mesh** component
3. In Details panel, find **Skeletal Mesh**
4. Click the dropdown and search for `SK_SportsCar` or `Vehicle`
5. Select the mesh

Option B: Use Your Own Vehicle Mesh

1. Import your Skeletal Mesh (FBX) into the project
2. Make sure it has a **Physics Asset**
3. Make sure it has wheel bones named consistently (e.g., `Wheel_FL`, `Wheel_FR`, etc.)
4. Select the **Mesh** component → Set your Skeletal Mesh

> 💡 **No mesh yet?** The Unreal Marketplace has free vehicle meshes. Search for "vehicle skeleton" in the Fab marketplace.

4

Configure Wheel Setup

This is the most important step for the car to move properly!

1. Select the **VehicleMovement** component
2. In Details panel, find **Wheel Setups**
3. You should see 4 entries (FL, FR, RL, RR)
4. For each wheel, set:

- **Wheel Class:** Leave default or create custom
- **Bone Name:** The exact name of the wheel bone in your skeleton

Common Wheel Bone Names:

Position	Common Names
Front Left	`Wheel_FL` , `WheelFrontLeft` , `wheel_lf`
Front Right	`Wheel_FR` , `WheelFrontRight` , `wheel_rf`
Rear Left	`Wheel_RL` , `WheelRearLeft` , `wheel_lr`
Rear Right	`Wheel_RR` , `WheelRearRight` , `wheel_rr`

⚠️ **Bone names must match EXACTLY!** Open your Skeleton asset to see the real bone names. Case matters!

5

Place Vehicle in Level & Test!

1. **Compile** and **Save** your Blueprint (buttons at top)
2. Drag `BP_MyFirstVehicle` from Content Browser into your level
3. Make sure the vehicle is **above the ground** (not clipping through)
4. Add a **Player Start** nearby (so you can walk to the car)
5. Press **Play**!

Test your vehicle:

- ✓ Walk to the car - you should see "Press [F] to Enter" prompt (or your character enters interaction range)
- ✓ Press `F` to enter the vehicle
- ✓ Press `E` to start the engine
- ✓ Press `W` to accelerate
- ✓ Press `F` again to exit

🎉 **Congratulations!** You have a working vehicle! Continue reading to add sounds, set up input properly, and customize everything.

🎮 Input Setup (Enhanced Input)

The Easy Vehicle System uses Unreal's **Enhanced Input System**. You need to create Input Actions and a Mapping Context for full control.

> ⚠️ **Why isn't my vehicle responding to input?** You probably haven't assigned Input Actions yet! Follow this section carefully.

1

Create Input Actions Folder

1. In Content Browser, create a new folder: `Input`
2. Inside Input, create another folder: `VehicleActions`

Your structure: `Content/Input/VehicleActions/`

2

Create All Input Actions

For each action below, **right-click → Input → Input Action**:

Name	Value Type	What It Does
`IA_Vehicle_Throttle`	Axis1D (Float)	Accelerate
`IA_Vehicle_Brake`	Axis1D (Float)	Brake/Reverse
`IA_Vehicle_Steering`	Axis1D (Float)	Turn left/right
`IA_Vehicle_Handbrake`	Digital (Bool)	Handbrake
`IA_Vehicle_Engine`	Digital (Bool)	Start/Stop engine
`IA_Vehicle_Horn`	Digital (Bool)	Honk horn
`IA_Vehicle_Lights`	Digital (Bool)	Toggle headlights
`IA_Vehicle_LeftIndicator`	Digital (Bool)	Left blinker
`IA_Vehicle_RightIndicator`	Digital (Bool)	Right blinker
`IA_Vehicle_Camera`	Digital (Bool)	Switch camera view
`IA_Vehicle_Exit`	Digital (Bool)	Exit vehicle
`IA_Vehicle_LookAround`	Axis2D (Vector2D)	Free look camera
`IA_Vehicle_Radio`	Digital (Bool)	Toggle radio
`IA_Vehicle_NextStation`	Digital (Bool)	Next radio station
`IA_Vehicle_PrevStation`	Digital (Bool)	Previous station
`IA_Vehicle_Wipers`	Digital (Bool)	Toggle wipers

💡 **How to set Value Type:** Double-click the Input Action → In Details panel, set "Value Type" to the type shown above.

3

Create Input Mapping Context

1. Right-click in your Input folder → **Input** → **Input Mapping Context**
2. Name it `IMC_Vehicle`
3. Double-click to open it
4. Click the **+** button to add mappings

Add these mappings:

Input Action	Key	Modifiers (if needed)
IA_Vehicle_Throttle	`W`	-
IA_Vehicle_Brake	`S`	-
IA_Vehicle_Steering	`A`	Negate (for left)
IA_Vehicle_Steering	`D`	- (for right)
IA_Vehicle_Handbrake	`Space`	-
IA_Vehicle_Engine	`E`	-
IA_Vehicle_Horn	`H`	-
IA_Vehicle_Lights	`L`	-
IA_Vehicle_LeftIndicator	`Q`	-
IA_Vehicle_RightIndicator	`R`	-
IA_Vehicle_Camera	`C`	-
IA_Vehicle_Exit	`F`	-
IA_Vehicle_LookAround	`Mouse XY`	-
IA_Vehicle_Radio	`M`	-
IA_Vehicle_NextStation	`.`	-

IA_Vehicle_PrevStation	`,`	-

4

Assign to Your Vehicle Blueprint

1. Open your vehicle Blueprint (`BP_MyFirstVehicle`)
2. Select the root component (**Self**)
3. In Details panel, scroll down to **"EVS Input"** section
4. Assign each Input Action to its corresponding slot:

```
EVS Input Section: ├── Vehicle Mapping Context → IMC_Vehicle ├──
Throttle Action → IA_Vehicle_Throttle ├── Brake Action →
IA_Vehicle_Brake ├── Steering Action → IA_Vehicle_Steering ├──
Handbrake Action → IA_Vehicle_Handbrake ├── Toggle Engine Action →
IA_Vehicle_Engine ├── Horn Action → IA_Vehicle_Horn ├── Toggle Lights
Action → IA_Vehicle_Lights ├── Left Indicator Action →
IA_Vehicle_LeftIndicator ├── Right Indicator Action →
IA_Vehicle_RightIndicator ├── Toggle Camera Action → IA_Vehicle_Camera
├── Exit Vehicle Action → IA_Vehicle_Exit ├── Look Around Action →
IA_Vehicle_LookAround ├── Toggle Radio Action → IA_Vehicle_Radio ├──
Next Station Action → IA_Vehicle_NextStation └── Prev Station Action →
IA_Vehicle_PrevStation
```

✅ **Done!** Compile, save, and test. All inputs should now work!

Audio System

The **EVSAudioComponent** handles all vehicle sounds. It supports engine sounds with RPM-based pitch, horn, doors, indicators, wipers, and even a radio system!

Adding Sounds to Your Vehicle

1. Open your vehicle Blueprint
2. Select **AudioComponent** in the Components panel
3. In Details panel, find the **"EVS Audio"** categories
4. Assign your sound assets to each slot

Available Sound Slots

Category	Sound Slot	Description	Type
Engine	Engine Start Sound	Played when starting engine	One-shot
	Engine Stop Sound	Played when stopping engine	One-shot
	Engine Loop Sound	Continuous engine sound (pitch changes with RPM)	Looping
Horn	Horn Sound	Played while holding horn button	Looping
Doors	Door Open Sound	When entering/exiting	One-shot
	Door Close Sound	After entering/exiting	One-shot
Lights	Light Switch Sound	Click when toggling lights	One-shot
	Indicator Tick Sound	Blinker tick sound	One-shot (repeats)
Wipers	Wipers Loop Sound	While wipers active	Looping
Radio	Radio Stations (Array)	Music tracks	Looping

	Radio Static Sound	When changing stations	One-shot
	Radio Toggle Sound	On/off click	One-shot
Misc	Handbrake Sound	Handbrake engage sound	One-shot
	Seatbelt Sound	Buckle click	One-shot

Engine Sound Settings

The engine loop sound's pitch changes based on RPM. Configure these in AudioComponent:

Setting	Default	Description
Engine Min Pitch	0.8	Pitch at idle (0 RPM)
Engine Max Pitch	2.0	Pitch at max RPM
Indicator Tick Interval	0.5	Seconds between blinker ticks
Radio Volume	0.7	Radio music volume (0-1)

Pro Tip: For realistic engine sounds, use a **Sound Cue** with multiple layers (idle, low RPM, high RPM) and crossfade between them. The EVS system will still control the pitch!

Radio System

Add music tracks to the **Radio Stations** array. Players can cycle through them with Next/Previous station keys.

1. Select AudioComponent
2. Find **Radio Stations** array
3. Click **+** to add entries
4. Assign Sound Wave or Sound Cue assets

💡 Lighting System

The **EVSLightingComponent** creates dynamic lights for your vehicle: headlights, tail lights, brake lights, reverse lights, and indicators.

How It Works

Lights are created automatically at runtime based on position offsets you configure. No need to manually add light components!

Configuring Light Positions

1. Open your vehicle Blueprint
2. Select **LightingComponent**
3. In Details, find **"EVS Lighting | Positions"**
4. Adjust the offset vectors to match your car mesh

Position Offsets

Light	Default Offset (X, Y, Z)	Tips
Left Headlight	(220, -70, 50)	X = Forward, Y = Left
Right Headlight	(220, 70, 50)	X = Forward, Y = Right
Left Tail Light	(-220, -70, 60)	Negative X = Rear
Right Tail Light	(-220, 70, 60)	
Left Indicator Front	(230, -80, 45)	Usually outer edge
Right Indicator Front	(230, 80, 45)	
Left Indicator Rear	(-230, -80, 55)	
Right Indicator Rear	(-230, 80, 55)	

Light Settings

Setting	Default	Description
Headlight Color	Warm White	Color of headlights
Headlight Intensity	20000	Brightness (candelas)
Headlight Range	2000	Attenuation radius
Tail Light Color	Red	
Tail Light Intensity	500	Normal brightness
Brake Light Intensity	3000	When braking
Indicator Color	Orange	
Indicator Blink Rate	0.5	Seconds per blink

Debugging lights: In Play mode, toggle **Show** → **Lights** in the viewport to see light radii. Adjust positions until they align with your mesh.

Enter/Exit System

The **EVSInteractionComponent** handles proximity detection and smooth enter/exit transitions.

How It Works

1. Player walks near the vehicle (within Interaction Radius)
2. Component detects the player and fires `OnCharacterEnteredRange`
3. Player presses the Exit/Enter key (default: `F`)
4. Vehicle's `EnterVehicle()` function is called
5. Player character is hidden and Player Controller possesses the vehicle

Settings

Setting	Default	Description
Interaction Radius	200	How close player must be (cm)
Driver Door Offset	(0, -120, 0)	Where the "entry point" is
Exit Offset	(0, -180, 50)	Where player appears on exit
Show Interaction Prompt	true	Enable/disable UI prompt
Interaction Prompt Text	"Press [F] to Enter"	Text to show

Safety Settings (on Vehicle Base)

Setting	Default	Description
Allow Exit While Moving	false	Can player exit at speed?
Safe Exit Speed (km/h)	5	Max speed for safe exit

⚠️ **Player not entering?** Make sure your player character has collision and the interaction radius is large enough. Try increasing it to 300-400 for testing.

Fuel System (Optional)

The **EVSFuelComponent** adds realistic fuel consumption. It's optional - if you don't want fuel mechanics, just disable it!

Enable/Disable Fuel System

1. Open your vehicle Blueprint
2. Click **Add Component** → **EVS Fuel Component**
3. Or if already present, set **Fuel System Enabled = false** to disable

Fuel Settings

Setting	Default	Description
Fuel System Enabled	true	Master on/off
Max Fuel	60 L	Tank capacity
Starting Fuel	60 L	Fuel at spawn
Idle Consumption	1 L/hour	Fuel use when idling
Max Consumption	15 L/hour	Fuel use at full throttle
Low Fuel Threshold	15%	When to trigger warning
Stop Engine When Empty	true	Auto-stop on empty tank

Blueprint Events

- `OnLowFuel` - Fires when fuel drops below threshold
- `OnFuelEmpty` - Fires when tank is empty

- `OnRefueled(float NewAmount)` - Fires after adding fuel

Refueling

```
// Blueprint or C++: FuelComponent→AddFuel(30.0f); // Add 30 liters
FuelComponent→FillTank(); // Fill to max FuelComponent→SetFuel(50.0f);
// Set exact amount
```

⌨️ Default Controls

These are the recommended key bindings. You set these up in your Input Mapping Context.

Driving

Action	Keyboard	Gamepad
Accelerate	`W`	Right Trigger
Brake / Reverse	`S`	Left Trigger
Steer Left	`A`	Left Stick
Steer Right	`D`	Left Stick
Handbrake	`Space`	A Button

Vehicle Controls

Action	Keyboard	Gamepad
Start/Stop Engine	`E`	Y Button
Horn	`H`	D-Pad Up
Toggle Headlights	`L`	D-Pad Down
Left Indicator	`Q`	D-Pad Left
Right Indicator	`R`	D-Pad Right
Toggle Wipers	`V`	-

Camera & Interaction

Action	Keyboard	Gamepad
Switch Camera	`C`	Right Stick Click
Look Around	Mouse	Right Stick
Enter/Exit Vehicle	`F`	B Button

Radio

Action	Keyboard	Gamepad
Toggle Radio	`M`	-
Next Station	`.`	RB
Previous Station	`,`	LB

🔧 Troubleshooting

❌ **Vehicle doesn't move at all**

❌ **Vehicle flips or behaves strangely**

❌ **Can't enter the vehicle**

❌ **No sounds playing**

❌ **Lights not visible**

❌ **Compilation errors**

❌ **Plugin not showing up**

FAQ

Can I use this for multiplayer?

Does it work with AI?

Can I use my own vehicle mesh?

Can I have multiple vehicles in one level?

How do I make a faster car?

How do I disable fuel consumption?

Can I modify the source code?

Support

Marketplace Q&A

Fastest response! Ask questions directly on the Marketplace product page.

Website

Visit adrenalinegames.pl for more resources.

Email

support@adrenalinegames.pl

When reporting issues, please include:

- Unreal Engine version
- Steps to reproduce the issue
- Error messages (screenshot if possible)
- What you've already tried

Version 1.0.0 • Made with ❤️ by Erak Dev