Exhibition of Speed

Manual v0.4.0

Coordinate system

The axes of the vehicle are defined as follows:

- **X** axis: Left
- **Y** axis: Up
- **Z** axis: Forward

Collision meshes and graphical meshes

Meshes must be added in the *Resources* section under the *Vehicle* tab. The compatible meshes present in the sandbox folder will be displayed (see next section for location of this folder).

Collision meshes must be provided in the OBJ format and should contain multiple convex polyhedrons. If a polyhedron is not convex, an error will be displayed indicating which submesh is not convex so it can be fixed. In *Blender*, select the offending mesh, and in Edit Mode, select all faces and navigate to *Mesh* > *Convex Hull*. Uncheck *Use Existing Faces* in the operator dialog after applying it to ensure convexity will be within tolerance for nearly flat faces.

Graphical meshes must be provided in GLTF format (or binary GLB).

Once in the resources list, collision meshes can be selected as a chassis' shape and Mesh nodes can instantiate any of the graphical meshes.

Saving and loading vehicles

A vehicle is made of a JSON file plus resources (which could be OBJ and GLTF files, for example) which are referenced in the JSON. These files are stored in the application's sandbox folder.

The application only has access to files within its sandbox folder, which can be found in the following locations:

- **Linux**: ~/.local/share/exhspd
- **Windows**: C:\Users\NAME\AppData\Roaming\exhspd

Vehicles are saved to the *vehicles* subfolder in that location. To add a new vehicle that has been obtained from an external source, move its files to the *vehicles* subfolder and it will appear in the file dialog when loading new vehicles.

To share a vehicle with other users or machines, go into the *vehicles* subfolder and simply share the JSON and resources files of that vehicle maintaining folder structure. It's recommended to keep one vehicle's resources into a designated folder which can be zipped and shared. Other users then simply have to unzip it and move the contained folder into the *vehicles* subfolder to be able to load it in game.

Creating custom wheels and tires

Select a wheel node to be able to open the tire or wheel library in the *wheel properties* panel on the right. In there new tires or wheels can be created from a GLTF file. The contents of the selected mesh in the GLTF file will be evaluated to verify it's cylindrical and obtain measurements.

The axis of rotation of the wheel or tire must be aligned with the X axis in the game, displayed in red on the floor. The positive direction of the X axis indicates the left side of the vehicle. The outside of the wheel must be facing this direction on the left side. The inside of the wheel must be facing this direction on the right side. Depending on the orientation of the mesh in the GLTF file, it's necessary to change the axis from +X to -Y on the left side for example.

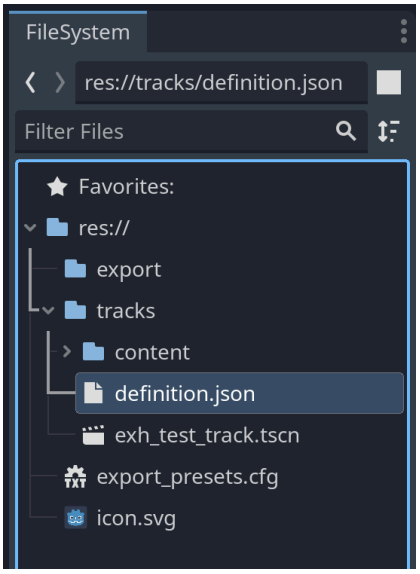
Variations of existing wheels can be created as well. The visual and physical material properties can be altered.

Press **Ctrl+Z** to undo accidental deletions. Press **Ctrl+Y** or **Ctrl+Shift+Z** to redo.

Loading custom maps

New maps can be created in the Godot editor and exported as `.pck` or `.zip` files which can be imported into the game.

The contents of the track are specified in a JSON file that must be located at `res://tracks/definition.json` in your project's *FileSystem*. The track itself is a regular Godot Scene `.tscn` which should be alongside this definition file. The usual practice is to put all other files in a "content" folder (such as textures, gltf, obj...).



The `definition.json` file specifies the paths to the main scene, collision meshes, among others. Following is an example of what this file could look like:

```
{
    "name": "Test Track",
    // Path to Godot Scene.
    "scene": "tracks/exh_test_track.tscn",
    // Path to the center line of the race track. It is a list of vertices
    // located at the center of the track surface ordered in either direction.
    // They wrap around at the end forming a loop.
    "centerLine": "tracks/content/CenterLine.obj",
    // Path to the racing line. This is used by autonomous driving systems.
    // It is also a list of vertices along the track surface.
    "racingLine": "tracks/content/RacingLine.obj",
    // The track limit lines have the same number of vertices as the racing line
    // and the same indexing. They represent the outer and inner boundaries of the
    // track surface.
    "trackLimitL": "tracks/content/TrackLimitL.obj",
    "trackLimitR": "tracks/content/TrackLimitR.obj",
    // A number in [0,1] that indicates where in the centerLine the start/finish
    // line is located, where 0 corresponds to the first vertex in the list.
    "origin": 0.2435,
    // Whether the racing direction is the inverse of the order in which vertices
    // are specified in centerLine.
    "reversed": true,
    // Lists of collision shapes of different types.
    "collision": {
        // Trimeshes are large concave objects, such as the track surface
        // and terrain, which normally should be merged into a single
        // continuous mesh to avoid internal edge collisions.
        "trimesh": [
            {
                "path": "tracks/content/ExhTestTrack.obj",
                "friction": 1.05,
                "restitution": 0.16,
```

```
                "thickness": 0.5
            }
        ],
        // Compounds are concave objects made of multiple convex shapes.
        "compounds": [
            {
                "path": "tracks/content/Grandstand.obj",
                "friction": 0.5,
                "restitution": 0.05
            }
        ],
        // Polyhedrons will be interpreted as lists of individual convex shapes.
        "polyhedrons": [
            {
                "path": "tracks/content/InnerBarrier.obj",
                "friction": 0.6,
                "restitution": 0.3
            }, {
                "path": "tracks/content/OuterBarrier.obj",
                "friction": 0.6,
                "restitution": 0.3
            }, {
                "path": "tracks/content/PitlaneInnerBarrier.obj",
                "friction": 0.6,
                "restitution": 0.3
            }, {
                "path": "tracks/content/PitlaneOuterBarrier.obj",
                "friction": 0.6,
                "restitution": 0.3
            }, {
                "path": "tracks/content/Pitstop.obj",
                "friction": 0.5,
                "restitution": 0.05
            }
        ]
    },
    // Cut sensors are specified as convex polyhedrons.
    "cutSensor": {
        "polyhedrons": [
            "tracks/content/CutSensors.obj"
        ]
    },
    // Location of each position on the starting grid.
    "gridTransform": [
        {
            "rotation":[
                -0.008130297996103764,
                -0.7287766337394714,
                -0.01645275019109249,
                0.6845055818557739
            ],
            "translation":[
                4.40433931350708,
                16.970674514770508,
                -75.86822509765625
            ]
```

```
            },
            // ...
        ]
    }
```

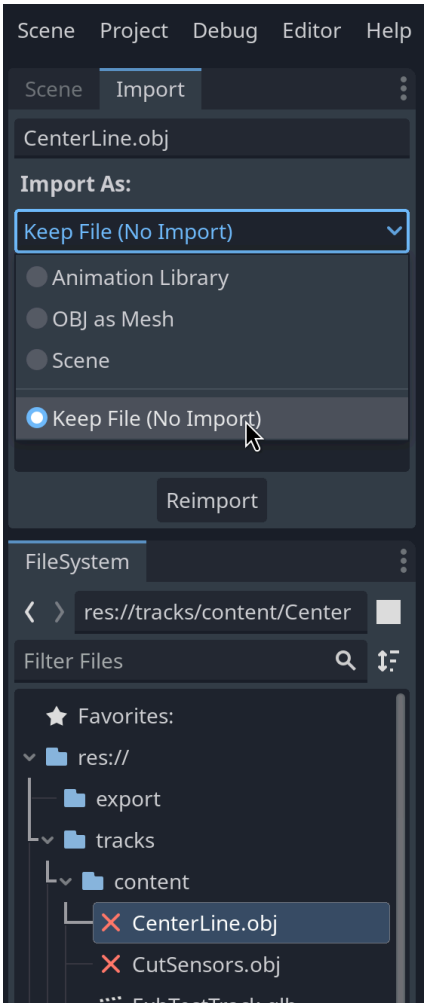
The definition system can be improved. The goal is to eventually move all the details specified in the `definition.json` to the Godot Scene and get rid of this file.

The graphical mesh normally is imported into the scene as a GLTF file exported from an application such as Blender.

Obj file notes

The center line and racing line must be included as an `.obj` file in the project. The Godot editor might crash if the obj file has a only a segment and not a face, so it's ok to close a face before exporting the obj (e.g. select all vertices and press `F` in Blender).

The obj files must be imported as files, not resources, as they're not meant to be rendered but instead, be used as collision meshes by the physics engine and segments for the racing logic. Thus, select each obj file in the Godot editor and in the Import tab, Import As, select `Keep File (No Import)` in the drop down menu.



Center line and racing line

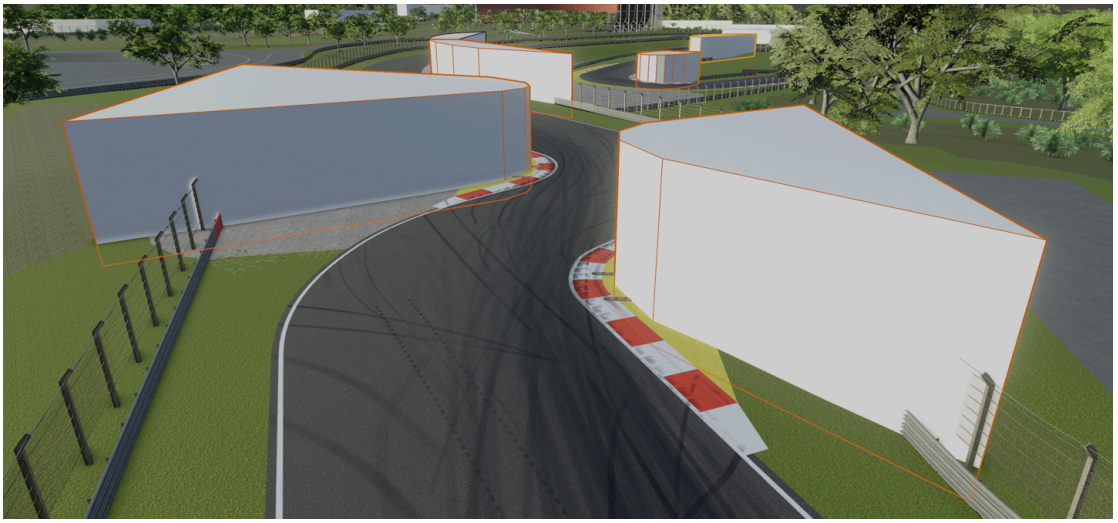
To see if the center line is reversed, check in which direction the vertex indices grow. To show vertex indices in Blender, enable Developer Extras in the Interface preferences, and then an Indices checkbox will appear under the Viewport Overlays.

The origin is where the start/finish is located along the center line. To find this number, right click your center line and choose `Convert To > Curve`, create an Empty object and assign a Follow Path constraint to it and choose your center line as the target. Adjust the Offset parameter towards a **positive value** until the Empty object lies on the finish line on your race track. Take this Offset value and divide it by 100 to keep it in the unit [0, 1] range.

The track limits can be built as a modified copy of the racing line. In Blender, the Shrink Wrap modifier can be used to push the racing line towards the outer and inner boundaries of the race track. Use vertical walls that extend from the track boundaries as the target.

Cut sensors

Cut sensors are convex polyhedrons placed on the inside of turns and other places where a car could take a shortcut. If the chassis of a vehicle comes into contact with one of these shapes, their current lap is considered invalid and other penalties can be applied.



Grid positions

The `gridTransform` entry contains a list of rotations and translations of each location on the starting grid.

One way to build such a list using Blender, is to position `Empty` objects along the center line using a `Follow Path` constraint (with `Follow Curve` checked) and then select them all and export as GLTF (selected objects only). Then simply copy-paste the transforms from the GLTF file into the `gridTransform` array in `definition.json`.

The vehicle is spawned at the first grid position on start up.

Material properties and per-vertex friction coefficient

Trimeshes can have per-vertex friction and restitution coefficients stored as RGB vertex colors in the `.obj` file. This allows pavement to have higher friction than grass, for example.

The red component is friction, green is restitution and blue is material ID.

Friction and restitution are interpreted as values in [0,1] and they're multiplied by the `"friction"` and `"restitution"` values in the JSON file.

Material IDs are calculated from the blue vertex color multiplied by 255 then rounded. They are required to play adequate sound effects and render the markings left by the tires on the ground. Following is the list of supported material IDs:

- Asphalt = 1

- AsphaltYellow = 2
- AsphaltRough = 3
- Kerb = 4
- GroundYellow = 5
- ConcreteRoad = 6
- ConcreteSmooth = 7
- Gravel = 8
- Grass = 9
- GrassLeaves = 10

Export notes

In Export, Resources tab, make sure to include `*.json, *.obj` in *Filters to export non-resource files/folders*.

Export as `.zip` if you want your map to be editable by others.

Default Controls

General

Press **Ctrl+S** to save changes to the current vehicle and input settings.

Press **P** to pause the simulation and press **L** while paused to step the simulation forward. Press **P** again to unpause.

Press **Ctrl + Space** to enter full screen mode. Press **Ctrl + Space** again or **Esc** to exit full screen.

On the graph view, press **Shift+A** to show the node creation popup. Press the item's initials to navigate. Press **Space** or **Enter** to to select an entry.

To find a node by name press **Ctrl+F**.

Navigation

Press **C** to change camera.

- Free look: allows you to fly around the scene freely.
- Chase: third person camera that follows the vehicle.
- Follow: similar to free look but it moves along with the vehicle.

- Hood: camera on top of vehicle's hood.
- Bumper: camera on the vehicle's bumper, low to the ground.

Hold middle mouse button and use the **WASD** keys to move around. Keys **E** and **Q** move up and down.

Hold **Alt** to move more slowly. Hold **Shift** to move faster.

Interaction

Click the *Grab* button to click and grab any part of the vehicle. Use this in case the car flips.

Keyboard

- **F** : Steer left
- **H** : Steer right
- **T** : Throttle
- **G** : Brakes
- **R** : Clutch
- **J** : Shift down
- **U** : Shift up
- **1**-**8** : Select n-th gear
- **9** : Reverse gear
- **0** : Neutral gear
- **C** : Change camera
- **V** : Look behind
- **Ctrl+R** : Reset vehicle

Gamepad

- **Left Joystick** : Steer left-right
- **Right Joystick Up** : Brakes
- **Right Joystick Down** : Throttle
- **Left Shoulder** : Shift down
- **Right Shoulder** : Shift up
- **X**: Change camera
- **A**: Reset vehicle
- **Y**: Look behind

Wheel

Expected defaults will be assigned. Will not correspond to all types of hardware. Manual setup is likely necessary. In the *World* tab, scroll down to the *Input* section and under *Method*, click on *Wheel*.

Press the gear button on the right of the axis button to configure range. Setting *Range Start* to 1 and *Range End* to -1 is equivalent to inverting the axis.

Decrease the overall range of the steering axis to increase steering sensitivity. A *Range Min* and *Range Max* of -0.45 and 0.45 (resp.) might be a good choice.