Vehicle Basics

Vehicles are the models that are moving around the game world to transport passengers or cargo. Currently there are two types of vehicles:

- Vehicles that can be bought and managed by the player (buses, trucks, trams, trains, ships and planes)
- Vehicles that are controlled by the AI to represent the private transportation (cars)

The technical specialities for the different means of transport are listed in the sections linked above. Common Properties for player and AI controlled vehicles are explained below as well as other relevant metadata elements for vehicles. To get an overview over the gameplay relevant aspects of the different vehicle types, have a look at the game manual section for vehicles.

To find out more about 3D models in general, have a look at the file type section. The general properties of `.mdl` files are described in the model definition section. Below are the specific things that are added to models to use them as vehicles.

Emissions

Vehicles have some emissions depending on their age and maintenance level. The emissions are configured in the `emissions` metadata block:

```
emission = {
  idleEmission = -1,
  powerEmission = -1,
  speedEmission = -1,
},
```

The `idleEmission` property is used when the vehicle stands still. `powerEmission` defines the emission when the thrust of the engine is at maximum and `speedEmission` is the emission when the vehicle is at full speed. For situations between, the values are interpolated. Setting a property to `-1`, the actual value is automatically computed by using the other provided metadata like speed, availability and power.

The allowed ranges for the three properties are:

- idleEmission: [0.0, 100.0]
- speedEmission: [0.0, 2.0]
- powerEmission: [0.0, 0.0002]

Player Controlled Vehicles

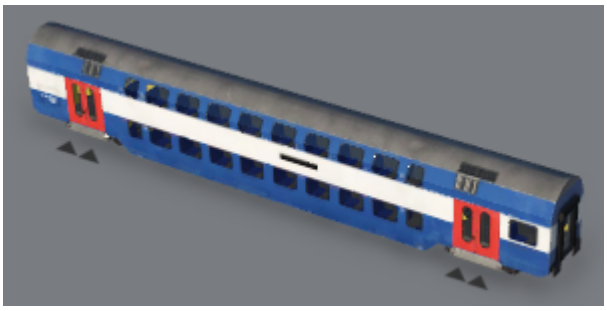
Vehicles that can be bought and controlled by the player all have a general metadata block called `transport Vehicle`:

```
transportVehicle = {
  carrier = "RAIL",          -- or "ROAD", "TRAM", "WATER", "AIR"
  loadSpeed = 2,             -- specifies how fast passengers or cargo items
are loaded/unloaded
  groupFileName = "",        -- for grouping in the buy menu

  -- Capacities
  compartmentsList = {
   ...                       -- see below
  },

  -- RAIL only
  multipleUnitOnly = false,  -- display only as part of multiple units
  reversible = false,        -- optional, if vehicle is reversible - for
RAIL vehicles only
  departureDelay = 2500      -- optional, adjusts the timing for the
departure of the vehicle, see below
},
```

Most of the properties are common for all types of vehicles:


passenger wagon with 4 doors

- `carrier` distinguishes between the different means of transportation. Possible values are “ROAD” for buses and trucks, “TRAM” for trams, “RAIL” for locomotives and wagons, “WATER” for ships and “AIR” for planes.
- `loadSpeed` is a property that specifies how fast passengers and cargo items are loaded/unloaded. For passenger waggons, a rule of thumb is that each door lane where a passenger could step into the vehicle or out of it at the same time counts as one loading speed unit.
- `groupFileName` is used whenever vehicles are grouped as subvariants in the buy menus. See the buy menu group details for more information.
- `compartmentsList` is used to define the load of the vehicle. See below for further information.

Three properties are currently only relevant for rail vehicles:

- `multipleUnitOnly` is a value that hides the model itself from the buy menu when set to `true`. It is still usable in multiple units though.
- `reversible` specifies if this vehicle is capable of being used in push/pull trains. See the reversible trains details for more information.
- `departureDelay` is a custom value in milliseconds that describes the time span between the closing of doors until the vehicle departs. If unset, the longest door close animation duration is used.

AI Controlled Vehicles

Some residents are considered as car owners. If they decide to not use public transportation to travel to their destination, they might use a car. To set a road vehicle model to be used by the AI as a car, it requires an empty metadata block:

```
car = { },
```

As a reference for better balancing here are the common top speeds for different eras of vanilla cars:

1850 - 1900	1900 - 1950	1950 - 1980	from 1980
20 km/h 5,56 m/s	50 km/h 13,89 m/s	80 km/h 22,22 m/s	100 km/h 27,77 m/s

To extend variety in private transportation, cars can have a preset of colors that can be used to recolor the cars randomly. It is required to have a material with a color blending map. The colors then can be defined in a metadata block in the `.mdl` file of a car:

```
colorConfig = {
  configs = {
    { { 0.26554900407791, 0.31372499465942, 0.22268399596214, }, },
    { { 0.61393797397614, 0.61960798501968, 0.52241402864456, }, },
    ...
  },
},
```

The color each are three values, one for red, green and blue channel with a range from 0 to 1.

Compartment List

Most properties that are load related can be found in the `compartmentsList` block that is part of the `transportVehicle` block. It is a nested structure with various hierarchy levels.

```
compartmentsList = {
  {
    loadConfigs = {
      {
        cargoEntries = {
          {
            capacity = 650,
            cargoBay = {
              bbMax = { 32.318698883057, 4.9700999259949, 3.5, },
              bbMin = { 13.892600059509, -4.9700999259949, 0.5, },
              cargoFormat = "BIG",
              childId = 1,
              gridSize = { 3, 4, },
              sizePolicy = "STRETCH_HEIGHT_SCALEY",
              type = "DISCRETE",
```

```
            },
            seats = { },
            type = "LOGS",
          },
        },
        toHide = { "comp_a", "comp_b", },
      },
      ... -- other load configurations
    },
  },
  ... -- other compartments
},
```

The `compartmentsList` block is a list of compartments. A vehicle can have one or more of these compartments. Usually one compartment is enough, but for vehicles that can carry several cargos, this might be a relevant use of multiple compartments. For example a hypothetic double deck tram has an upper deck for passengers only and a lower deck for various cargos. Then the upper deck is represented with the first compartment and the lower deck is represented in an independent compartment.

Each compartment currently only contains one property, another block called `loadConfigs`. This `loadConfigs` block is a list with one or more load configurations. A compartment can be used with different cargo combinations, e.g. a ship could either be loaded with coal only or it could be split in two halves with coal in one and iron ore in the other one of them. As a rule of thumb, every different capacity distribution possibility needs a seperate load configuration. When a vehicle is empty and should be loaded, the first suitable configuration that results in the most transported cargo items is used with consideration of the cargo type filters set in the game.

A load configuration has two properties:

- `toHide` is a list of `.msh` references that are hidden when this load configuration is used.
- `cargoEntries` is a list of cargo entries one for each cargo type that is used in this load configuration.

Each cargo entry has several parameters:

- `capacity` is the amount of cargo items or passengers that fits in the compartment.
- `type` is the cargo type. See cargo types for information about the cargo types.
- `seats`, `cargoBay` and `customCargoModels` are relevant for the visual representation of the loaded cargo. They are optional and not exclusive, thus they could be used in combination.

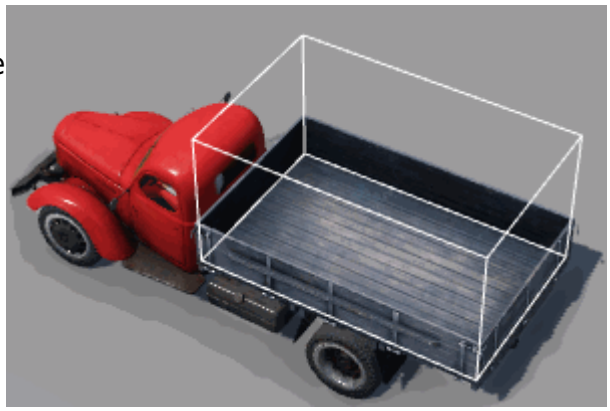
seats

The `seats` block is a list that contains all seat ids of seats from the seatProvider belonging to the configuration. Then passengers loaded in this compartment of the vehicle are only displayed on the respective seats.

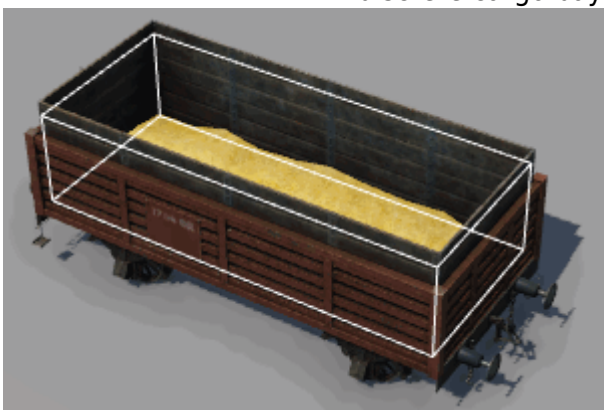
This property is ignored for all cargo types other than “PASSENGERS”.

cargoBay

A cargo bay is a box that is filled with cargo models depending on the amount of cargo that is loaded into the compartment. The `cargoBay`-Block has several properties:



discrete cargo bay



level cargo bay

- `bbMax` and `bbMin` are the parameter for the cargo bay bounding box that encloses the space of the model in which the cargo should be displayed. The values are the distances from anchor mesh origin in positive and negative direction on all three axis.
- `cargoFormat` specifies which generic model should be used. Possible values are “SMALL” and “BIG” for all cargo types and additionally “MEDIUM4x1” or “MEDIUM2x1” for bulk cargos. If left blank the best fitting of “SMALL” or “BIG” is used.
- `gridSize` is the pattern in which the cargo models are layed. The values are 1 or above. Other values or unset are considered as 1. The first value is the number of models in x direction, the second value is the number in y direction. If there is a third value, several layers are used in z direction. If no gridSize is specified, the content is filled with the least scale deviation to the original model sizes.
- `sizePolicy` is the property that defines how the models are scaled to fit in the bounding box of the cargo bay.:
  - “STRETCH” scales the models in x and y direction to fit the bounding box
  - “STRETCH_HEIGHT_SCALEY” scales the models in z direction too. All scaling factors are independend from each other..
  - “BEST_FIT” scales the models with the same scaling factor in all direction so that they fit in the bounding box.
  - unset or other contents do not scale the models at all, they are used in their original size.

- `type` is used to distinguish between the type of cargo bay. There are two types available:
  - "`DISCRETE`" uses the discrete cargo formats ("`SMALL`" or "`BIG`") and puts them into the box according to the `gridSize` pattern and the `sizePolicy`
  - "`LEVEL`" uses the bulk cargo formats ("`MEDIUM4x1`" or "`MEDIUM2x1`") to show a surface model at different heights depending on the loaded amount of cargo. `gridSize` is ignored and `sizePolicy` should be set to "`STRETCH`".

This property is ignored for cargo type "`PASSENGERS`".

customCargoModels

Custom cargo models can be used to define own cargo models independend of the generic ones defined by the cargo type configuration. The generic cargo models can be arranged in a custom way. It is possible to use this feature to get randomized cargo visualisation too. See custom cargo models description for further details.

This property can be used with every cargo type including "`PASSENGERS`".

Seats

Seats are the positions, where passenger and crew models are located. The positions are defined in another metadata block called `seatProvider`:

```
seatProvider = {
  drivingLicense = "TRAM",
  crewModels = { "characters/era_c_driver_rail.mdl", },
  seats = {
    {
      animation = "idle",
      crew = true,
      forward = true,
      group = 1,
      transf = { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1.5, 0.15, 0.85, 1, },
    },
    ...
  },
```

The `drivingLicense` property is used to locate the right default crew models. Available

are “BUS”, “TRUCK”, “TRAM”, “RAIL”, “WATER”, “AIR” and “AIR_OUTDOOR” (with pilot helmet). To use custom crew models, it is possible to list model references in `crewModels`.

The `seats` block contains a list of zero or more seat subblocks. Every block has the following properties:

- `animation` specifies which posture and animation should be used for this seat. Available are `driving`, `driving_upright`, `idle`, `sitting` and `walk`.
- `crew` defines if this person is a crew member. Crew members are visible even if there is no passenger on board. Non-crew member seats are used for passengers. This parameter is optional, if omitted default `false` is used.
- `forward` is used to show crew members only in the right direction when reversible trains are used. See there for more details. This parameter is optional.
- `group` is a node id. This mesh is used as an anchor for the seat, e.g. if it rotates or scales, the seat rotates or scales as well. If the anchor mesh is used as a part that is only visible under certain conditions, the seat is only visible then too. This can be used to hide crew members in secondary locomotives.
- `transf` is used to scale, rotate and position the seat relative to the anchor mesh. It requires a 16 number transformation matrix. The `transf.lua` and `vec3.lua` scripts can be used if these operations should be parameterized with seperate values for scaling, rotation and positioning.

Vehicle Types

From:
https://www.transportfever2.com/wiki/ - **Transport Fever 2 Wiki**

Permanent link:
https://www.transportfever2.com/wiki/doku.php?id=modding:vehiclebasics

Last update: **2022/05/09 19:41**