Vehicle Advanced Topics

This page contains different topics relevant for some vehicle modding projects.

Reversible Trains

Rail vehicles can be configured to not flip when turning around in a station, but instead to reverse direction. The lights and the driver position can be configured accordingly.

transportVehicle

In order for a train to be able to reverse direction, the first, the last vehicle as well as every motorized vehicle in the consist needs the reversible parameter set to true.

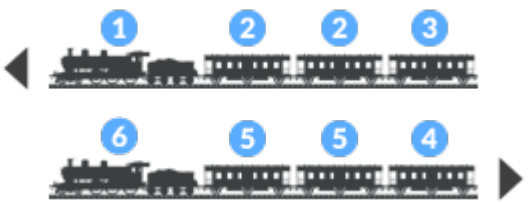
```
transportVehicle = {
  -- other configurations
  reversible = true,
}
```

railVehicle

The visibility of models can be controlled depending on the travel direction. Note that forward facing models only apply to the first, and backward facing models only the last vehicle in the composition.

```
railVehicle = {
  -- other configurations
  configs = {
    {
      -- other configurations
      frontForwardParts = { 10 },
      innerForwardParts = { },
      backForwardParts = { },
      frontBackwardParts = { 12 },
      innerBackwardParts = { },
      backBackwardParts = { },
    }
  }
}
```

1 `frontForwardParts` is a list of mesh ids that should be only visible if the vehicle is at the front of the whole train consist and driving forward
2 `innerForwardParts` is a list of mesh ids that should be only visible if the vehicle is not at the front



or back of the whole train consist and driving forward
3 `backForwardParts` is a list of mesh ids that should be only visible if the vehicle is at the back of the whole train consist and driving forward
4 `frontBackwardParts` is a list of mesh ids that should be only visible if the vehicle is at the front of the whole train consist and driving backward (only if consist is reversible)
5 `innerBackwardParts` is a list of mesh ids that should be only visible if the vehicle is not at the front or back of the whole train consist and driving backward (only if consist is reversible)
6 `backBackwardParts` is a list of mesh ids that should be only visible if the vehicle is at the back of the whole train consist and driving backward (only if consist is reversible)

Be aware that an instance of a mesh can't be used for multiple of these purposes. Each mesh id may at most appear in one of the config lists. To use the same mesh in several cases, add another instance of it. It is possible to use these configurations for other purposes like end of train devices, switching pantographs depending on the location in train and interconnecting gangways between coaches.

seatProvider

It is possible to restrict crew seats to only be used in forward or backward direction by using the `forward` property.

```
seatProvider = {
  drivingLicense = "RAIL",
  crewModels = {},
  seats = {
    {
      animation = "driving_upright",
      crew = true,
      forward = true,
      group = 10,
      transf = { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 8.5, 0.15, 1.35, 1, },
    },
    {
      animation = "driving_upright",
      crew = true,
      forward = false,
      group = 12,
      transf = { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, -8.5, 0.15, 1.35, 1, },
    },
    ...
  },
```

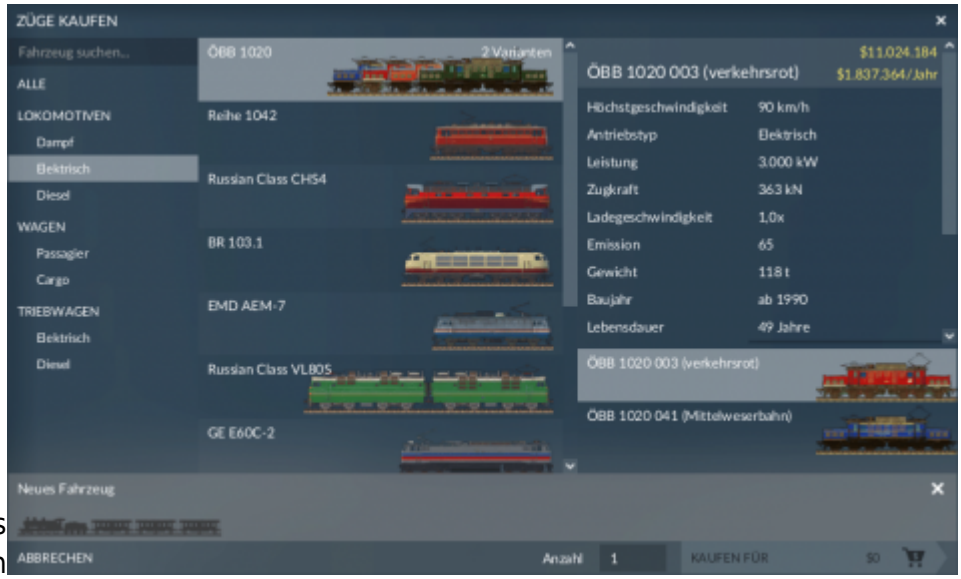
To display some seats only when the vehicle is at the front of the train, use one of the meshes that are set in the config properties described above as anchor meshes. In the code block above these would be mesh id 10 for forward front lights and mesh id 12 for backward front lights.

Buy Menu Groups

With Transport Fever 2 a new grouping feature has been added to the buy menu. This is especially useful for shape variants, repaints and multiple units with different length configurations. The following section is a detailed introduction to the necessary adjustments of the models.

Code adaptation in model files

Groups for the buy menu are passively defined in Transport Fever 2, i.e. a group is not explicitly defined with the contents contained in a separate file. Instead, the vehicles are optionally specified to be included in a group (and in which group). Only then will a purchase menu entry be displayed as a group.



In the game such a grouping is indicated by the reference to n variants in the upper right corner of the group entry. In the right column you will then find another list with the variants contained in these groups in the lower area. The technical data shows the values of the currently selected variant in the group.

The group is entered in the `transportVehicle` code block. The keyword to be used is `groupFileName`. The path to any mdl, which is to serve as a group entry in the purchase list, must be specified relative to the `models` folder. An example looks like this:

```
transportVehicle = {
  carrier = "RAIL",
  compartmentsList = {
    ...
  },
  loadSpeed = 1,
  multipleUnitOnly = false,
```

```
  reversible = false,
  groupFileName = "vehicle/train/menu_1020.mdl",
},
```

Code adaptation in *.lua files of multiple units

In the case of multiple-unit trains, the representation looks exactly the same as for individual vehicles. Here, such a grouping is indicated by the reference to n variants in the upper right corner of the group entry too. In the right column you will then find another list with the variants contained in these groups in the lower area. The technical data shows the values of the currently selected variant in the group.

The group is entered in the `return` code block. The keyword to use is also `groupFileName`. Behind it is the path:

- relative to the `/models` folder if a model is to serve as a group.
- relative to the `/config/multiple_unit` folder if another multiple unit is to serve as a group.

An example looks like this:

```
function data()
return {
  vehicles = {
    ...
  },
  groupFileName = "vehicle/train/menu_cityjet.mdl",
  name = _("oebb_4744_name"),
  desc = _("oebb_4746_desc")
}
end
```

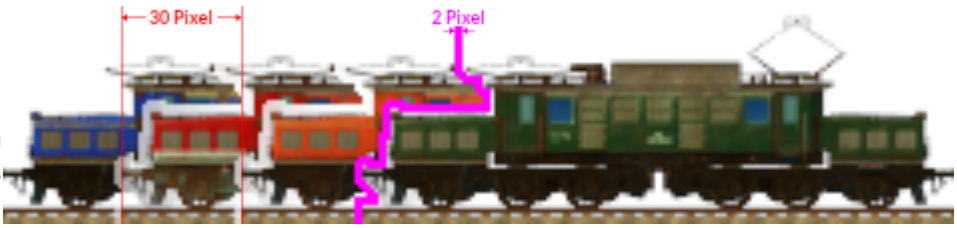
Individual group pictures and names

Especially for vehicles with many colour variations one would like to show in the group entry what one can expect in the group. Therefore it is necessary to create an individual menu model as a placeholder, which is never displayed in the game itself as a variant to buy. This model gets the name of the desired group and the preview image is saved as ui image in the `ui/models_small` folder. In the code it is important that the `multipleUnitOnly` entry is set to `true`. As long as no multiple unit

contains the model, this means that the entry itself is never offered for sale, but only appears if a subordinate train or model is available. It is important that the `groupFileName` is not set for this model:

```
transportVehicle = {
  carrier = "RAIL",
  compartmentsList = {
    ...
  },
  loadSpeed = 1,
  multipleUnitOnly = true,
  reversible = false,
  -- groupFileName not set!
},
```

The examples shown above each use an individualized menu item according to this scheme. To support a common style, it's recommended to use one of the two common styles for the menu icon.



The first style places the icons all on the same level. To create the image do the following steps:

1. Take up to 4 of the available variants
2. Place them 30 pixels apart of each other with the right one being at the front
3. Make an approximately 2 pixel wide gap between the variants to improve the contrast
4. For multiple units, use 50 - 100 % of the second coach, too
5. In the normal size of the purchase menu, the images can be a maximum of 327 pixels wide. Then the menu is stretched accordingly. Starting from 700 pixels the right edge is cut off.

Another common style uses diagonally offset images. There is a fanmade online generator tool available, though it is currently only available in german.

Different availability, other modes of transport and engine types

For vehicles with a long development time, it is not necessary for the group model to cover the availability range of all variants. If only one variant is available at a time, it is displayed individually in the purchase list. As soon as two or more variants are available, the group entry with the variant list is displayed.

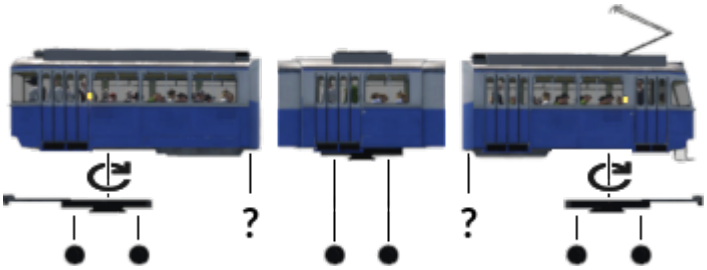
Grouping is not only possible for rail vehicles. It can also be used for other vehicle types (aircraft, ships, road and tram vehicles).

For rail vehicles, there is also the special case that different variants of a group have defined diesel and other electrical equipment as drive types, for example. If in the purchase menu the tab is set to “ALL”, all are displayed under the group. If e.g. diesel vehicles have been explicitly selected, only the corresponding subset of the vehicles is displayed.

Fake Bogies

Transport Fever 2 uses the axle configs to determine which part of the vehicle need to be aligned along the tracks or street lane:

- The origin of an axle always sits centered above the the lane.
- Nodes that contain axle meshes as children are considered as bogies. Bogies have a pivot point at their mesh origin, which should sit in the center between the axles.
- Nodes that contain two bogies have their pivot point in the center between the bogie pivot points.



vehicle parts with their pivot points at normal bogies

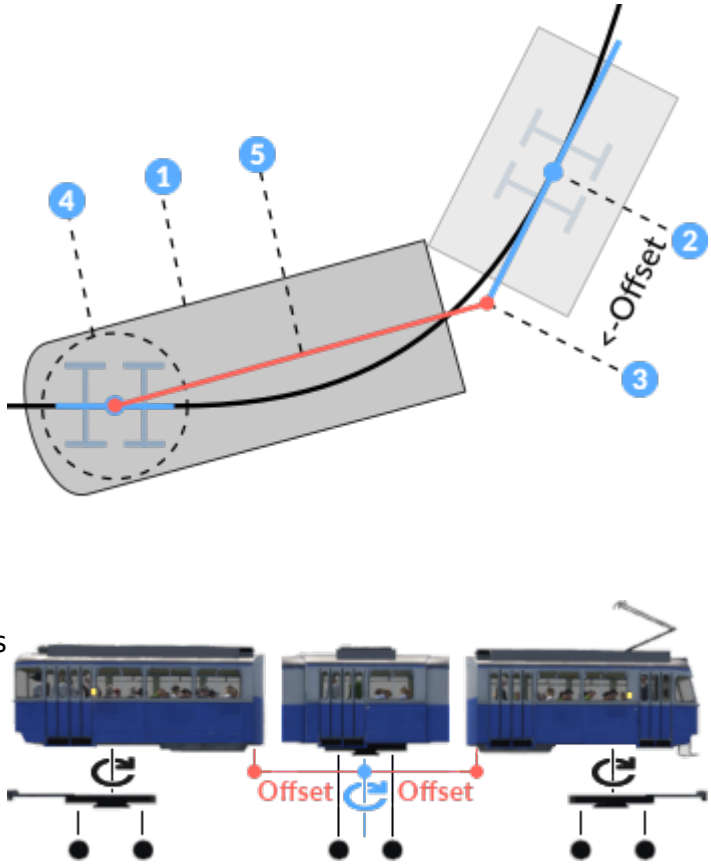
The picture shows the Be 4/6 tram that has three articulated parts. The middle frame sits on two axles and is considered as a bogie on its own. The front and back parts have one bogie seperated from the main frame. As they each do not have a second bogie, the articulation would not look good in turns. That is why there might be the need to imitate a bogie, especially on articulated vehicles or road vehicles with steering wheels.

These so called fake bogies are manually scripted pivot points that are attached to a node as replacement for non existant real bogies. They are listed per LOD in a `fakeBogies` list block in the vehicle `config` block:

```
...
configs = {
  {
    axles = { "vehicle/tram/be4_6mirage/w1_lod0.msh", ... },
    -- lights
    -- front/backwar parts
    fakeBogies = {
      {
        group = 1,
        offset = 2.0299999713898,
        position = 0,
      },
      {
        group = 33,
        offset = -2.0299999713898,
        position = 0,
        upright = false,
      },
    },
  },
  ...
},
```

Each fakeBogie has three properties:

- `group` is the id of the node to which the fake bogie should be attached. In the picture, this would be the one larger grey end body mesh 1.
- `position` is the position along the lane from the root node of the model. Positive values go to the front of the vehicle, negative values to the end. In the picture, the blue point on the right is positioned where the `position` property points 2. At this point, a tangent to the lane is calculated. This is visualized by the blue line.
- `offset` is the offset from the place where `position` points to the actual point on this tangent that is used as the pivot point. In the picture, this is the red point 3.
- `upright` is an optional parameter that can be set to true for models that should always stand or hang upright.



vehicle parts with additional fake bogies

It is possible to define up to two fakeBogies per node. If the number of real bogies and fakeBogies exceeds two, the regular bogies are overridden. In the example there is a regular bogies 4 beside the fakeBogie explained above. The node will then align along the connection of the two pivot points 5.

The Be 4/6 tram that with its three articulated parts received two fake bogies both positioned in the center and with an offset from there.

Road vehicles usually have their fake bogies either between the two axles or in case of articulated busses on the back axle of the front part but with an offset to the hinge between front and back part.

It is recommended to use the node tree with numbers in the model editor to get the correct ids.

Custom Cargo Models

In Transport Fever 1, there was the possibility to define own cargo models for display on loaded vehicles with the `loadIndicators`. Due to a new implementation of the cargo loads in Transport Fever 2, this method is not supported anymore. Instead, the new `customCargoModels` concept is introduced which provides even more functionality. This feature has several possible use cases, e.g. it is possible to:

- scale the cargo models to different sizes depending on the position (e.g. for funnel-shaped bulk freight cars)
- randomize cargo models of different and same sizes (e.g. container wagons, car transport

wagons, ...)
- add additional models in combination with the common `seats` and `cargoBays` (e.g. for luggage in passenger coaches)

cargoSlotProvider block

In the `.mdl` files there are two relevant blocks concerning cargo known so far. For passengers this is the `seatProvider` and for all cargoTypes it is the `transportVehicle` block with the `compartmentsList`. To define the position of custom cargo models, there is a new block called `cargoSlotProvider`.

```
cargoSlotProvider = {
  slots = {
    {
      models = {
        "animal/cow.mdl",
        "#MEDIUM4x1",
        "#SMALL",
        "animal/sheep.mdl",
      },
      group = 0,
      transf = { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, -3, -0.5, 1.2, 1, }, --
offset, rotate and scale
      randomId = 1,
    },
    ...
  },
},
```

The `cargoSlotProvider` contains a list of slots inside the property `slots`. Each of the slots has several properties:

- `models` is a list of models which can be displayed at this position. They are randomly chosen.
- `group` is the mesh id of the mesh that should serve as an anchor for the cargo models. When this mesh rotates, the cargo models will rotate too.
- `transf` is used to scale, rotate and position the slot 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.
- `randomID` is a seed for randomization. Slots that use the same randomID value will pick the same index from the `models` list.

Be aware that the cargo models are actual `mdl` files and not only `mesh` files anymore. You can either reference a model relative to `res/models/model/` or set a dynamic reference to a generic cargo model depending on the actual cargo type by using "#SMALL" where SMALL is a key of the discreteModels or levelModels in the cargo type definition.

customCargoModels block

Inside the cargoEntry block, it is possible to use a new block called `customCargoModels`. It can be used in addition or instead of `cargoBay` and `seats` there.

```
cargoEntries = {
  {
    capacity = 20,
    customCargoModels = {
      configurations = {
        {
          slotLevels = {
            { 0 },
            { 0, 1 },
            { 0, 1, 2 },
            { 0, 1, 2, 3 },
          },
        },
        ...
      },
    },
    type = "GOODS",
  },
},
```

The `customCargoModels` block has another property called `configurations`. This property contains a list of slot configurations. Each configuration has a list of `slotLevels`. Each level can have zero or more cargo slot ids from the `cargoSlotProvider` and they are used top down. The first level is used when the vehicle is empty, the last level is used when the vehicle is almost full. The more levels are defined, the smaller are the intervalls between the levels.

Example

As an example, the modern stake car that is available from year 2000 was used and modified so that it transports cargo items of type `MACHINES`. There are 4 slots defined in the cargoSlotProvider, one for the bus that uses the whole cargo waggon and three for the smaller cars that are placed next to each other. The car slots have more than one possible model that can be displayed. It is randomly selected which one should be displayed.

The customCaroModels block contains two configurations from which one is randomly selected. One configuration only uses the slot with the bus, the other configuration uses the three car slots.

The example `.mdl` file with cars as models for the “MACHINE” cargo type described above is available for download. Please be aware that this file alone is not usable in the

game, it needs some other files for a working mod.

Vehicle Types

Repaint Mods