Vehicle Types

Vehicle models in Transport Fever 2 are bound to one of the means of transportation road, rail, water or air. The distinction between the types of vehicles is achieved by specialized metadata entries in the `.mdl`-files for each type.

In the sections below, there are some identifiers that are used frequently:

- **mesh id** is a number that is equivalent to the index of the mesh in the model node tree. Use the MODEL button in the Model Editor to find out the correct number in the **ID** column.
- **.msh reference** is a path to a mesh file relative to `res/models/mesh/` including the `.msh` file ending. The correct reference is listed on the right side of the model node tree in the Model Editor too.

Road Vehicles

Road vehicles are bound to the street network built by the game or the player. They can drive along streets as well as on construction lanes of the right type in stations, depots and similar constructions.

They are indentified by the `roadVehicle` metadata block and the `.mdl`-files usually are located in the subfolders:

- `/vehicle/bus`
- `/vehicle/car`
- `/vehicle/truck`

For trams, see rail vehicles below.

Metadata

The `roadVehicle` metadata block consists of several block children as well as the properties for speed and weight:

```
roadVehicle = {
  configs = {
    -- light configuration
    -- axles configuration
    -- fakeBogies configuration
    -- wheels configuration
  },
```

```
  engine = {
    -- engine configuration
  },
  soundSet = {
    -- soundSet configuration
  },
  topSpeed = 13.8,       -- [m/s] maximum speed the vehicle can reach
  weight = 1.6,          -- [t (metric)] empty weight of the vehicle
  blinkInterval = 500,   -- [ms] interval for blinker lights, optional
},
```

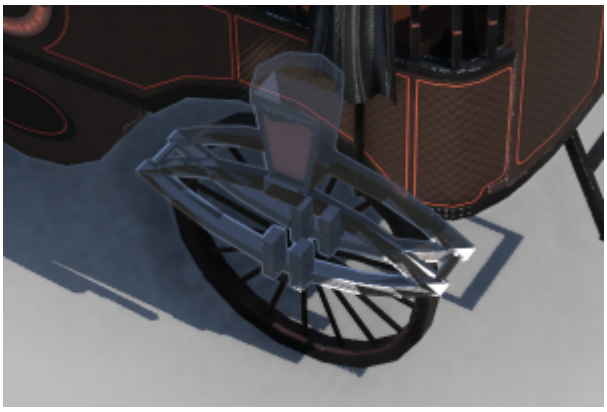
The `configs` block covers several properties regarding the 3D model. For each level of detail, there is one block with several properties. Some properties are used to define the lights of road vehicles:


headlight, blinker and turning wheel of a bus

- `blinkLightsLeft` is a list of mesh ids which should only be used when the road vehicle turns left or a passenger vehicle leaves a bus/tram stop. Trams do not blink at intersections.
- `blinkLightsRight` is a list of mesh ids which should only be used when the road vehicle turns right or a passenger vehicle stops at a bus/tram stop. Trams do not blink at intersections.
- `brakeLights` is a list of mesh ids which should only be used when the vehicle reduces speed.
- `headLights` is a list of mesh ids which should be used for the normal headlights.

Be aware that a mesh can't be used for multiple lighting purposes. Each mesh id may at most appear in one of the config lists.

Beside the lights, there are further config properties that are used for steering and wheels:


steering part of L'Obéissante

- `axles` is a list of `.msh` references pointing to the meshes which are axles that only rotate around the y axis while the vehicle is moving. This is usually used for rear axles.
- `fakeBogies` is a list of fake bogie definitions. Usually road vehicles have at least one positioned approximately in the middle between the steering axis and the first back axis.
- `steeringParts` is a list of mesh ids that are rotating around z axis when the vehicle turns around curves.
- `wheels` is a list of `.msh` references pointing to the meshes that represent wheels which rotate around z axis in curves and around y axis while the vehicle is rolling. This is usually used for front wheels.

Keep in mind to set the correct mesh origins for the vehicle parts to ensure they rotate in the right way. For axles, the origin should be aligned in the center in terms of the x and z axis. Otherwise the axle will wobble ingame.

The `engine` block describes the type and power of the vehicles motorization:

- `type` describes the motor type. Available are “`HORSE`”, “`STEAM`”, “`DIESEL`”, “`ELECTRIC`”. This property has an influence on the emission calculation, e.g. electric has lower emissions than diesel.
- `power` is the maximum power the engine can produce in *kW*.
- `tractiveEffort` is the maximum tractive force the vehicle can develop in *kN*.

The `soundSet` block contains the sound related properties:

- `name` is a reference to a soundset file. The path is relative to `res/config/sound_set/`.
- `horn` is a reference to a custom horn sound. If this is set, it overrides the `horn` event reference in the soundset above. The path is relative to `res/audio/effects/`.
- `openDoors` is a reference to a custom door opening sound. If this is set, it overrides the `openDoors` event reference in the soundset above. The path is relative to `res/audio/effects/`.
- `closeDoors` is a reference to a custom door closing sound. If this is set, it overrides the `closeDoors` event reference in the soundset above. The path is relative to `res/audio/effects/`.

The other three properties are not contained in a block:

- `topSpeed` is the maximum speed the vehicle can reach in meter per second.
- `weight` is the empty weight of the vehicle in metric tons.
- `blinkInterval` is an optional parameter to adjust the frequency of the blink lights by specifying the interval in milliseconds. If unset, the default value of 500 ms is used.

Events

Road vehicles support some animation events. They can be used in mesh nodes to support model aspects like opening doors:

- `forever` animation is used for roof ventilators or other elements that turn independently from the speed of the vehicle. It is looped forever and has no fixed length.
- `drive` animation is used for vehicle parts that move according to the current speed of the vehicle.
- `open_all_doors` is triggered after a vehicle stops at a station or bus/tram stop. It is triggered for stops on both sides of the vehicle.

Door animation of BK 670 bus

- `close_all_doors` is triggered before the vehicle leaves a station or bus/tram stop. It is triggered for stops on both sides of the vehicle.
- `open_doors_right` is triggered after a vehicle stops at a station or bus/tram stop. It is only triggered for stops on the right side.
- `close_doors_right` is triggered before the vehicle leaves a station or bus/tram stop. It is only triggered for stops on the right side.
- `open_doors_left` is triggered after a vehicle stops at a station or bus/tram stop. It is only triggered for stops on the left side.
- `close_doors_left` is triggered before the vehicle leaves a station or bus/tram stop. It is only triggered for stops on the left side.
- `empty_load` animation is used to empty the payload of tipper trucks. This animation is currently not used in game.

Rail Vehicles

Rail vehicles run on the track network built by the player and can drive along built tracks as well as on track lanes in stations, depots and similar constructions.

They are indentified by the `railVehicle` metadata block and the `.mdl`-files usually are located in the subfolders:

- `/vehicle/train` for motorized rail vehicles
- `/vehicle/wagon` for unmotorized rail vehicles
- `/vehicle/tram` for trams.

Although trams are defined with a `railVehicle` metadata block, they run on streets (with tram tracks) instead of train tracks.

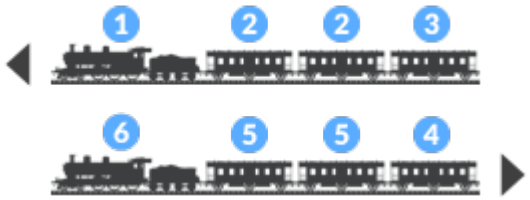
Metadata

The `railVehicle` metadata block consists of several block children as well as the properties for speed and weight:

```
railVehicle= {
  configs = {
    -- axles configuration
    -- fakeBogies configuration
    -- lights configuration
  },
  engines = {
    -- engines configuration
  },
```

```
  soundSet = {
    -- soundSet configuration
  },
  topSpeed = 27.4,          -- [m/s] maximum speed the vehicle can reach
  weight = 107,             -- [t (metric)] empty weight of the vehicle
  blinkInterval = 500,      -- [ms] interval of blinking lights, optional,
only for trams
},
```

The `configs` block covers several properties regarding the 3D model. For each level of detail, there is one block with several properties. Some properties are used to define the lights of rail vehicles:



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)


second locomotive without headlights

Be aware that a mesh can't be used for multiple lighting purposes. Each mesh id may at most appear in one of the config lists. 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.

For trams, there are additional metadata properties for the blinking lights.

- `blinkingLights0` / `blinkingLights1` are used for meshes that should blink when a tram leaves a station.
- `blinkLightsLeft0` / `blinkLightsLeft1` are used for meshes that should blink when a tram turns left at an intersection.
- `blinkLightsRight0` / `blinkLightsRight1` are used for meshes that should blink when a tram turns right at an intersection.

The meshes with 0 and 1 suffix alternate when blinking. Keep in mind that a mesh can't be used for multiple lighting purposes. Each mesh id may at most appear in one of the config lists.

Beside the lights, there are further config properties that are used for axles and the correct rotation of vehicle parts:

- `axles` is a list of `.msh` references pointing to the meshes which are axles that rotate around the y axis while the vehicle is moving. This is usually used for any physical axles.
- `fakeBogies` is a list of fake bogie definitions. Train vehicles with jacobs bogies and other nested constellations might need these.

Keep in mind to set the correct mesh origins for the vehicle parts to ensure they rotate in the right way. For axles, the origin should be aligned in the center in terms of the x and z axis. Otherwise the axle will wobble ingame.

The `engine` block describes the type and power of the vehicles motorization:

- `type` describes the motor type. Available are “`HORSE`”, “`STEAM`”, “`DIESEL`”, “`ELECTRIC`”. This property has an influence on the emission calculation, e.g. electric has lower emissions than diesel. Electric do only run on tracks with catenary.
- `power` is the maximum power the engine can produce in *kW*.
- `tractiveEffort` is the maximum tractive force the vehicle can develop in *kN*.

The `soundSet` block contains the sound related properties:

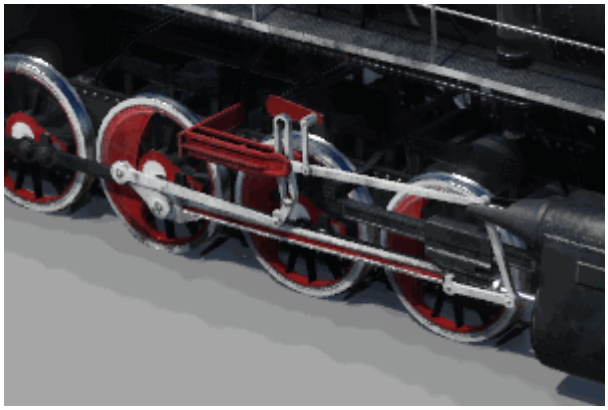
- `name` is a reference to a soundset file. The path is relative to `res/config/sound_set/`.
- `openDoors` is a reference to a custom door opening sound. If this is set, it overrides the `openDoors` event reference in the soundset above. The path is relative to `res/audio/effects/`.
- `closeDoors` is a reference to a custom door closing sound. If this is set, it overrides the `closeDoors` event reference in the soundset above. The path is relative to `res/audio/effects/`.
- `horn` is a reference to a custom horn sound. If this is set, it overrides the `horn` event reference in the soundset above. The path is relative to `res/audio/effects/`.
- `clacks` is a reference to a custom clack sound. If this is set, it overrides the `clacks` event reference in the soundset above. The path is relative to `res/audio/effects/`.
- `chuffs` is a reference to a custom chuffing sound. If this is set, it overrides the `chuffs` event reference in the soundset above. The path is relative to `res/audio/effects/`.

The other two properties are not contained in a block:

- `topSpeed` is the maximum speed the vehicle can reach in meter per second.
- `weight` is the empty weight of the vehicle in metric tons.
- `blinkInterval` is an optional parameter to adjust the frequency for the blink lights of trams by specifying the interval in milliseconds. If unset, the default value of 500 ms is used.

Events

Rail vehicles support some animation events. They can be used in mesh nodes to support model aspects like opening doors:


wheel animation of a steam locomotive

- `forever` animation is used for roof ventilators or other elements that turn independently from the speed of the vehicle. It is looped forever and has no fixed length.
- `drive` animation is used for vehicle parts that move according to the current speed of the vehicle.
- `wheels` animation is used for vehicle parts that move according to the current speed of the vehicle. The animation length is mapped to the wheel rotation by the game engine.
- `open_all_doors` is currently unused.
- `close_all_doors` is currently unused.
- `open_doors_right` is triggered after a vehicle stops at a station. It is only triggered for platforms on the right side.
- `close_doors_right` is triggered before the vehicle leaves a station. It is only triggered for platforms on the right side.
- `open_doors_left` is triggered after a vehicle stops at a station. It is only triggered for platforms on the left side.
- `close_doors_left` is triggered before the vehicle leaves a station. It is only triggered for platforms on the left side.

Multiple Units

Some rail vehicles are defined as multiple units. Then they are a fixed consist of more than one model. The configuration files for multiple units are located in the `res/config/multiple_unit/` folder.

```
function data()
return {
  vehicles = {
    { name = "vehicle/train/ice1.mdl", forward = true },
    { name = "vehicle/train/ice1_waggon_1.mdl", forward = true },
    ...
    { name = "vehicle/train/ice1.mdl", forward = false },
  },
  name = _("VEHICLE_MULTIPLEUNIT_ICE1_NAME"),
  desc = _("VEHICLE_MULTIPLEUNIT_ICE1_DESCRIPTION"),
  groupFileName = "",

}
```

```
end
```

The configuration has four properties:

- `vehicles` is a list of vehicle models used in the multiple unit consist. For each of the vehicles there are two properties:
  - `name` is a reference to the model file relative to `res/models/model`.
  - `forward` is a boolean value. If set to `true` the model is used as is, otherwise it is flipped by 180 degrees.
- `name` is the name of the multiple unit that is used in the buy menu. It can be translated in a strings.lua file.
- `desc` is the description of the multiple unit used in the buy menu. It can be translated too. The technical data is calculated based on the single vehicles used in the multiple unit.
- `groupFileName` is an optional reference to another multiple unit or a model to define this multiple unit as a variant under the other multiple unit / model as main group. See the buy menu group explanation for further details.

For multiple units to be available, all of their parts have to be in their respective availability timespan.

Water Vehicles

Ships are bound to the navigatable water on the map. The routes are dynamically calculated based on the reachable water regions. In harbors, they stop at designated points.

They are indentified by the `waterVehicle` metadata block and the `.mdl`-files usually are located in the `/vehicle/ship` subfolder.

Metadata

The `waterVehicle` metadata block consists of several block children as well as the properties for speed and weight:

```
waterVehicle = {
  configs = {
    -- paddles configuration
    -- rudder configuration
  },
  waterLine = {
    -- waterline configuration
  }
  area = 5,                         -- [m^2] cross sectional area, which
linearly correlates with the drag force
  availPower = 294000.0,            -- [W] maximum power the engines can
produce
```

```
  weight = 135000.0,                -- [kg] empty weight of the ship
  maxRpm = 55,                      -- [rpm] used to animate movable parts of
the engines
  topSpeed = 7.5,                   -- [m/s] top speed (movement speed is
clamped to this value)
  type = "BIG"                      -- is used to deny access for big ships at
small piers (alternative "SMALL")
},
```

The `configs` block covers several properties regarding the 3D model. For each level of detail, there is one block with several sub blocks.

- `paddles` is a small subblock with two properties for paddles and propellers. Paddle wheel steamer can keep the mesh orientation as usual, for modern propellers rotation around the x axis, simply rotate the mesh by 90° so the meshs y axis points in the direction of the intended rotation axis.
  - `ids` is a list of mesh ids that rotate around the y axis of the mesh while the vehicle is moving.
  - `maxAngle` is set to 0.
- `rudder` is another small subblock. The rudders are rotating around z axis when the ship curves.
  - `ids` is a list of mesh ids.
  - `maxAngle` is an angle in degree that limits the maximum deviation of the rudder meshes.

spinning propeller and turning rudder

Keep in mind to set the correct mesh origins for the vehicle parts to ensure they rotate in the right way. For paddles and propellers, the origin should be aligned in the center in terms of the x and z axis. Otherwise they will wobble ingame.

Other animated parts of the ship like waving flags and dangling anchors can be realized with `drive` and `forever` animations instead of the old config properties.

The `waterLine` block describes the form of the ship at water level that is used to generate the spume around the ships body. It is a list of two-value pairs each representing a point with x- and y-coordinate relative to the model origin.

There are several other properties:

spume around the water line of the ship

- `area` is the maximum under water cross sectional area of the ships body in m², which linearly correlates with the drag force
- `availPower` is the maximum available power the engines can produce in W (not in kW!)
- `weight` is the empty weight of the ship in kg (not in tons!)
- `maxRpm` is the maximum rotation per minute of the main paddle wheel. This value is used for animation calculation.
- `topSpeed` is the maximum speed the vehicle can reach in meter per second.
- `type` is used to distinct between small and large ships. Only "`SMALL`" ships can stop at the small piers. "`BIG`" ships need the large pier.

In contrast to the road and rail vehicles, the sounds are not in the `waterVehicle` block but in a special `soundConfig` block:

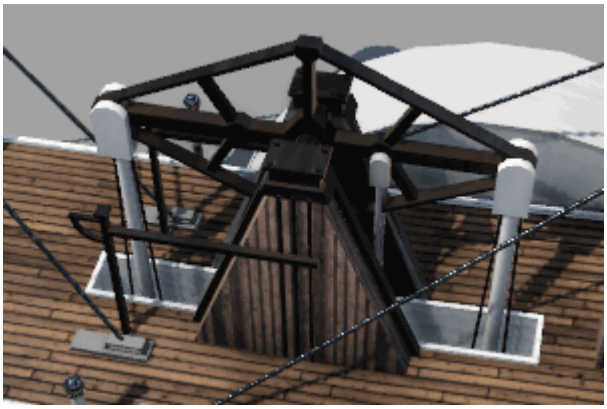
```
soundConfig = {
  effects = { },
  soundSet = {
    horn = "",
    name = "ship_diesel_modern",
  },
},
```

It contains two parts:

- The `effects` block is left empty for now.
- The `soundSet` block contains the known soundset related properties:
  - `name` is a reference to a soundset file. The path is relative to `res/config/sound_set/`.
  - `horn` is a reference to a custom horn sound. If this is set, it overrides the `horn` event reference in the soundset above. The path is relative to `res/audio/effects/`.

Events

Ships support some animation events. They can be used in mesh nodes to support model aspects like opening doors:


Drive Animation of Frontenac Steam Ship

- `forever` animation is used for continous animations like the radar beacons on ships.
- `drive` animation is used for vehicle parts that move according to the current speed of the vehicle.
- `open_all_doors` is triggered after a vehicle stops at a harbor. It is triggered for stops on both sides of the vehicle.
- `close_all_doors` is triggered before the vehicle

leaves a harbor. It is triggered for stops on both sides of the vehicle.
- `open_doors_right` is triggered after a vehicle stops at a harbor. It is only triggered for stops on the right side.
- `close_doors_right` is triggered before the vehicle leaves a harbor. It is only triggered for stops on the right side.
- `open_doors_left` is triggered after a vehicle stops at a harbor. It is only triggered for stops on the left side.
- `close_doors_left` is triggered before the vehicle leaves a harbor. It is only triggered for stops on the left side.

Air Vehicles

Airplanes are not bound to any terrain or infrastructure except the airports where they land, stop and start.

They are indentified by the `airVehicle` metadata block and the `.mdl`-files usually are located in the `/vehicle/plane` subfolder.

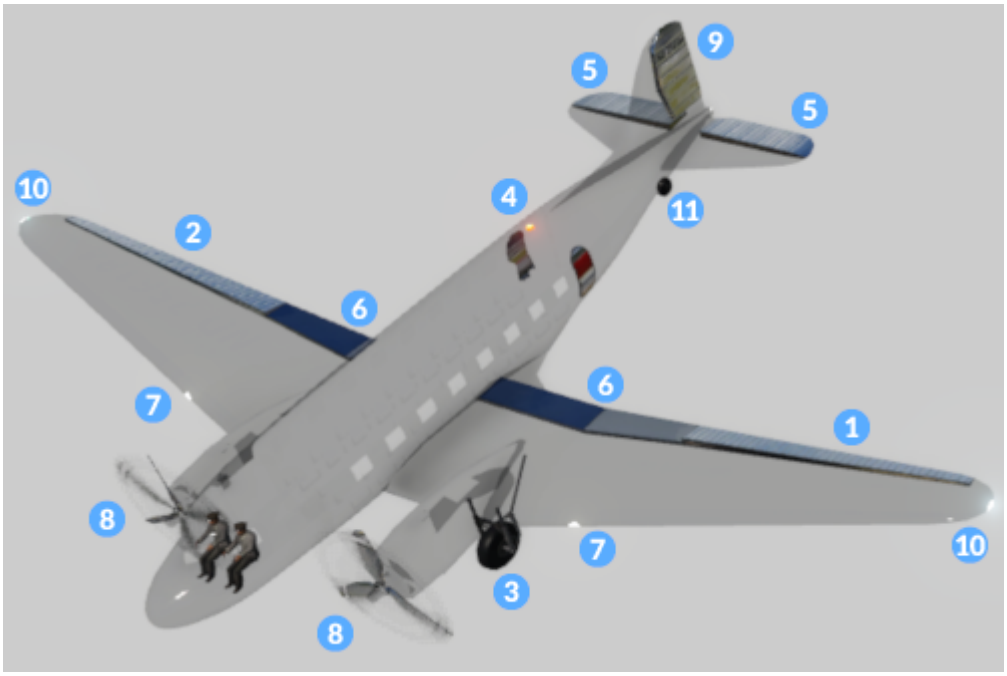
Metadata

The `airVehicle` metadata block consists of several block children as well as the properties for speed and weight:

```
airVehicle = {
    configs = {
       -- axles configuration
       -- wheels configuration
       -- elevator configuration
       -- aileronLeft configuration
       -- aileronRight configuration
       -- flaps configuration
       -- rudder configuration
    },
    maxPayload = 0,              -- [kg] (currently not in use)
    maxTakeOffWeight = 78000.0, -- [kg] (currently not in use)
    maxThrust = 236000.0,        -- [N] maximum total thrust the engines can
generate
    idleThrust = 11800.0,        -- [N] thrust generated by the engines in
idle state
    timeToFullThrust = 3,        -- [s] time the engines need from idle state
to generate full thrust
    topSpeed = 230.0,            -- [m/s]
    weight = 44000.0,            -- [kg] empty weight of the aircraft
```

```
    wingArea = 122.6,           -- [m^2] wing area, which correlates
linearly with the lift force
    type = "BIG",               -- is used to deny access for big planes at
small airfields (only "SMALL")
},
```

The `configs` block covers several properties regarding the 3D model. For each level of detail, there is one block with several sub blocks.



plane config elements

1 `aileronLeft` is a small subblock with two properties for the left aileron used when the aircraft is rolling.

- `ids` is a list of mesh ids that rotate around the x axis of the mesh during rolling maneuvers.
- `maxAngle` is an angle in degree that limits the maximum deviation of the aileron meshes.

2 `aileronRight` is a small subblock with two properties for the right aileron used when the aircraft is rolling.

- `ids` is a list of mesh ids that rotate around the x axis of the mesh during rolling maneuvers.
- `maxAngle` is an angle in degree that limits the maximum deviation of the aileron meshes.

3 `axles` is a list of `.msh` references pointing to the meshes which are axles that only rotate around the y axis while the vehicle is moving. This is usually used for axles that can't turn in curves on taxiway. `axleRadii` is a list of half diameters of the axles in meter for every axle mesh listed above. It is used to calculate the plane tilting on ground when the axles and wheels are not at the same height level and size.

4 `beaconLights` is a list of mesh ids which are flashing lights.

5 `elevator` is a small subblock with two properties for the elevator used when the aircraft is pitching.

- `ids` is a list of mesh ids that rotate around the x axis of the mesh during pitching maneuvers.
- `maxAngle` is an angle in degree that limits the maximum deviation of the elevator meshes.

6 `flaps` is a small subblock with two properties for the flaps that are used especially during take off and landing.

- `ids` is a list of mesh ids that rotate around the x axis of the mesh during flaps adjustments.
- `maxAngle` is an angle in degree that limits the maximum deviation of the flaps meshes.

7 `landingLight` is a list of mesh ids which are visible during landing approach and on ground.

8 `props` is a list of mesh ids which are rotating around the x axis depending on the thrust of the vehicle. `propsB` is a list of mesh ids which are used for blurred propellers at higher rotation speeds.

9 `rudder` is a small subblock with two properties for the rudders that are used in yaw maneuvers (turning).

- `ids` is a list of mesh ids that rotate around the z axis of the mesh during yaw maneuvers.
- `maxAngle` is an angle in degree that limits the maximum deviation of the rudder meshes.

10 `strobeLights` is a list of mesh ids which are double flashing lights.

11 `wheels` is a list of `.msh` references pointing to the meshes which are wheels that rotate around the y axis while the vehicle is moving on ground and around z axis while the vehicle is turning around curves. `wheelRadii` is a list of half diameters of the wheels in meter for every wheel mesh listed above. It is used to calculate the plane tilting on ground when the axles and wheels are not at the same height level and size.

tilted plane on ground

For some purposes, other properties in the `config` block are available:

- `fakeBogies` is a list of fake bogie definitions. Air vehicles with complicated axle and wheel constellations might need these.
- `steeringParts` is a list of mesh ids that are rotating around z axis when the vehicle turns around curves.

Keep in mind to set the correct mesh origins and rotations for the vehicle parts to ensure they rotate in the right way. There are different axis relevant for different purposes!

Beside the `config` block, there are several other properties:

- `maxThrust` is the maximum total thrust the engines can generate in N.
- `idleThrust` is the thrust generated by the engines in idle state in N.
- `timeToFullThrust` is the time the engines need from idle state to generate full thrust in seconds.
- `topSpeed` is the maximum speed the vehicle can reach in meter per second.
- `weight` is the empty weight of the plane in kg (not in tons!)
- `wingArea` is the wing area in m², which correlates linearly with the lift force.
- `type` is used to distinct between small and large planes. Only "`SMALL`" planes can land at the small airfields. "`BIG`" planes need the large airports.

In contrast to the road and rail vehicles, the sounds are not in the `airVehicle` block but in a special `soundConfig` block:

```
soundConfig = {
  effects = { },
  soundSet = {
    land = "",
    name = "aircraft_prop_modern",
  },
},
```

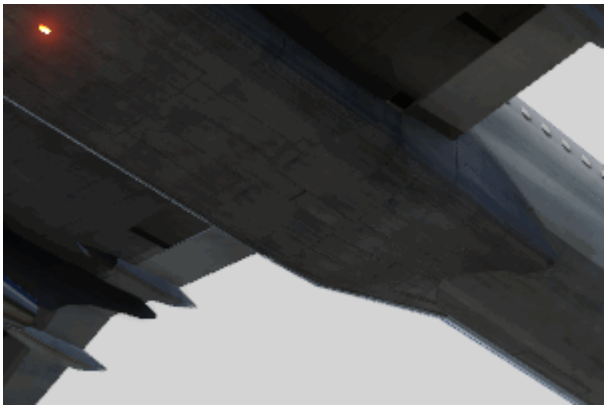
It contains two parts:

- The `effects` block is left empty for now.
- The `soundSet` block contains the known soundset related properties:
  - `name` is a reference to a soundset file. The path is relative to `res/config/sound_set/`.
  - `sonicBoom` is a reference to a custom sonic boom sound. If this is set, it overrides the `horn` event reference in the soundset above. The path is relative to `res/audio/effects/`.
  - `land` is a reference to a custom landing sound. If this is set, it overrides the `horn` event reference in the soundset above. The path is relative to `res/audio/effects/`.

Events

Planes support some animation events. They can be used in mesh nodes to support model aspects like opening doors:

- `forever` animation is used for elements that turn independently from the speed of the vehicle. It is looped forever and has no fixed length.
- `drive` animation is used for vehicle parts that move according to the current speed of the vehicle.
- `open_all_doors` is triggered after a vehicle stops at a gateway. It is triggered for gateways on both sides of the vehicle.
- `close_all_doors` is triggered before the vehicle



Extension and retraction of the landing gear

leaves a gateway. It is triggered for gateways on both sides of the vehicle.
- `open_doors_right` is triggered after a vehicle stops at a gateway. It is only triggered for gateways on the right side.
- `close_doors_right` is triggered before the vehicle leaves a gateway. It is only triggered for gateways on the right side.
- `open_doors_left` is triggered after a vehicle stops at a gateway. It is only triggered for gateways on the left side.
- `close_doors_left` is triggered before the vehicle leaves a gateway. It is only triggered for gateways on the left side.
- `open_doors_cargo` animation is used to open cargo payload doors.
- `close_doors_cargo` animation is used to close cargo payload doors.
- `open_wheels` is triggered before the landing approach to extend the landing gear.
- `close_wheels` is triggered after take off to retract the landing gear.

Vehicle Basics

Vehicle Advanced Topics