Modular Constructions

With Transport Fever 2, the concept of modular constructions was introduced. They are built like every other construction with parameters but can be edited afterwards by adding and removing some parts like pieces of a puzzle. These parts are called modules. The locations where they can be placed are indicated by slots.

To provide different start configurations, it is possible to define construction templates.

Construction Templates

A single `.con` construction file can provide several constructions to the player with different properties and icons. These can be defined in a `constructionTemplates` block in the shared construction file:

```
...
constructionTemplates = {
  {
    type = "DYNAMIC",
    constructionType = "STREET_STATION",
    availability = { ... },
    description = {
      name = _("Bus/tram station"),
      description = _("Modular bus/tram station."),
      icon = "ui/construction/station/street/passenger_era_a.tga",
    },
    data = {
      params = { ... },
    }
  },
  ...
}
```

For each of the blocks in the `constructionTemplates` list, there will be a new item in the construction menu. The block contains several parameters:

- `type` can either be “`DYNAMIC`” to return content in dependency of some parameters or “`STATIC`” to do not offer parameters.
- `constructionType` is set to define the actual type of this construction. See the construction basics for a list of all available types.
- `availability` restricts the availability of this dynamic construction analogue to the usual `availability` blocks.
- `description` is the block for the description. `name` and `description` work as usual, the `icon` property is used to reference the icon to display, it is located relative to `res/textures/`.

`data` is a block for additional data that is used depending on the type of the template. For “`STATIC`” templates, there is an `initialModules` block containing a mapping of modules to slotIDs like the one that is shown in the code sample below. For “`DYNAMIC`” construction templates, there is a `params` block like the ones known from conventional constructions.

In the main construction, there is a `createTemplateFn` function called for dynamic templates that is used to feed the initial module setup depending on the parameters from the `data` block. It receives several parameters:

- `paramX` and `paramY` are from the keyboard controls. See the basics to find out more about them.
- `seed` is a number that can be used for randomization purposes. It increases whenever a construction is set or proposed.
- `year` is the current year.
- `templateIndex` is the id of the template that was used for this function call.
- All custom parameters of the `data.params` block in the current asset construction template.

The function returns a mapping of modules to slotIDs. This may look like

```
{
  [10001000] = "station/air/airfield_main_building.module",
  [10002004] = "station/air/airfield_hangar.module",
  [10070002] = "station/air/airfield_passenger_terminal.module"
}
```

Method Cycle

The way the constructions are built is defined in the `game.config.ConstructWithModules` function that can be found in the `base_config.lua` in the `res/config` folder. This function is called whenever a construction should be built, either temporary as preview or permanent.

In the default implementation, it calls the `updateFn` function of the construction (line 41) and stores its result. If the parameters contain modules - both via template calls or during construction reconfiguration - a second part is executed (line 49 and below). For each module, sorted in ascending order of the `slotIds`, the following steps are executed:

1. Search for the `slotId` in the list of existing slots in the current result set from the construction and previous modules (line 62 - 67).
2. If the `slotId` is not in the list and invalid slots are not accepted (`result.callInvalidModules` unset or `false`) skip the current module (line 68).
3. Fetch the slot location (line 69) and tag name (line 70).
4. Prepare the `addModel` function (line 72 - 80).
5. Call the `updateFn` function of the module and provide the `addModel` function (line 93).

Finally if the construction added a `terminateConstructionHook`, that will be executed (line 101).

Construction UpdateFn

The `updateFn` function of the construction is used to assemble the core elements of the construction add put them in the result data struct. This may cover basic models, ground faces and terrain alignments as well as additional functions that can be called from modules later.

Parameters

The function receives a larger set of parameters:

- `paramX` and `paramY` are from the keyboard controls. See the basics to find out more about them.
- `seed` is a number that can be used for randomization purposes. It increases whenever a construction is set.
- `state` is a data struct containing several cached informations about assets and tracks.
- `year` is the current year.
- All custom parameters of the current asset construction.

Further it receives a list of all modules that should be built for the construction in `modules`:

```
modules = {
  [10001000] = {
    metadata = {},
    name = "station/air/airfield_main_building.module",
    updateScript = {
      fileName = "",
      params = {}
    },
    variant = 0
  },
  ...
},
```

The index is the slotID where the module should be placed. The properties are:

- `metadata` is a block containing all the metadata that was set in the `metadata` block of the module.
- name is the reference to the module relative to `res/construction/`.
- `updateScript` is a block containing information about an alternative update script that should replace the modules `updateFn` function. This is especially used for modules that are generated on the fly.
  - `fileName` is a path relative to `res/` that contains the path to a `.scrip` file, the filename without the file ending and the name of the function in the file as a suffix behind a `.`. An example would be “`construction/station/rail/modular_station/trackmodule.updateFn`”.
  - `params` is a list of parameters that are provided to this remote `updateFn` as a table in `closureParams`.
- `variant` is an integer that is increased and decreased by pressing M and N while placing a module. It can be used to provide rotated variants of models in a module or cycle through different assets. The value can be positive and negative, default is 0.

Return Data Struct

The result data struct contains several blocks. These cover the usual return properties of constructions as well as specific ones for modular constructions:

- `result.collider` for collider information.
- `result.cost` for the cost of the main construction. Modules may add their cost later.
- `result.edgeLists` for streets, tracks and taxiways.
- `result.edgeObjects` for waypoints and signals along the edges, e.g. along taxiways.
- `result.groundFaces` for the terrain painting.
- `result.models` for the 3D models.
- `result.runways` for airports and harbors.
- `result.slotConfig` for slot type restrictions.
- `result.slots` for a list of slots currently available in the construction.
- `result.stations` for an assignment of terminal groups to different stations in the construction.
- `result.terminalGroups` for the terminals of stations.
- `result.terrainAlignmentLists` for the terrain alignment.

It is possible that construction extend the `result` by custom properties, e.g. helper functions that can then be used by modules at a later point.

The description below only covers those that are specific for the modular constructions. See the construction basics and construction types for a more in detail explanation on the other parts.

In the `result.slotConfig` list, there can be restrictions for some slot types. The list uses the slot type as a key and contains two properties for each entry:

- `maxModules` is the maximum amount of modules with this slot type that can be built in the whole construction. If `maxModules` is set to `-2`, it depends on the value of `message` if the module button is enabled or disabled. No messages results in the button being enabled, a set message disables the button.
- `message` is a message that is displayed on modules of this type in the menu, when they can't be build due to the slot configuration restrictions.
- `skipCollisionCheck` is an optional boolean value that is used to disable the ui collision checks. With skipped checks, the slot markers are not painted red when they are obstructed by something else, e.g. free tracks. The potential collider collisions of the module models still apply.

The `result.slots` block is a list of all currently existing slots in the construction. Each entry in the list has the following properties:

- `id` is the unique slot id that is used as reference for the placed modules. It is a non-negative number.
- `type` is the key that is used to restrict the compatible modules on this position.
- `transf` is the position relative to the construction origin as a transformation matrix.
- `spacing` is a four value vector that is used to set the size of the slot around the `transf` position. It is `{-x, x, -y, y }`.
- `shape` is a value that is used to select the style of the slot marker symbol. It is either `0` for a square, 1 for a triangle, 2 for a transverse rectangle or 3 for a longitudinal rectangle.
- `height` is the height of the slot counted from the position of `transf`. Together with the `spacing` this is used for selection and bulldozing purposes.

Construction UpgradeFn

The `upgradeFn` function currently is only used by rail stations. It is used to react on the use of some modification tools like the electrification and track type refit tool.

The function receives several parameters to compute and returns a list of mappings from slotIds to modules. These are replacing previous modules at these slots.

The parameters are:

- `modules` is the list of modules that are used in the construction. It is equal to the `module` list in the section above.
- `slotId` is the slotId where the cursor pointed on.
- `catenaryToggle` set to `true` if the catenary modification tool is used.
- `trackTypeToggle` set to `true` if the track type modification tool is used.

The return structure is an indexed list like:

```
return {
  [10001000] =
"station/rail/modular_station/platform_high_speed_track.module",
  [10001010] =
"station/rail/modular_station/platform_high_speed_track.module",
}
```

Modules

A module is a construct of one or mode models that can be placed in certain positions in modular constructions. The basic structure of a module is:

```
function data()
return {
  type = "hangar",

  -- other meta information

  -- update function, which adjusts the module according the parameters
  updateFn = function(result, transform, tag)
    ...
  end,

  -- function that provides the models for floating modules attached to the
cursor while placing.
  getModelsFn = function()

  end
}
end
```

Each modules has a `type`, a set of metadata properties, an `updateFn` function and a `getModelsFn`.

The `type` is a key that is used to restrict modules to a certain slot type.

The modules have individual metadata properties:

- `availability` restricts the availability of this module analogue to the usual `availability` blocks.
- `description` is the block for the description. `name` and `description` work as usual, the `icon` property is used to reference the icon to display, it is located relative to `res/textures/`.
- `cost` and `maintenanceCost` are used to set the cost of this module.
- `category` is a block that contains the `categories` list with subcategory keys. These are the tabs in the module menu.
- `order.value` is used for the sort order in the module menu tabs.

To provide additional static metadata, the modules can provide more information in a `metadata` block. This will be sent to the `updateFn` of the construction.

Module UpdateFn

The `updateFn` function of modules is called after the construction `updateFn` to add the models of the module to the construction as well as do all the stuff that is needed for the module, e.g. adding terminals. It receives several parameters:

1. the `result` of the `updateFn` from the construction or the module that was called before containing all previously added models, data etc.
2. the location of the slot relative to the construction origin as a transformation matrix. This should be applied to all models of the module.
3. a `tag` that should be added to the models in the result to assign them to the current module at that slot.
4. the `slotId` that is a unique numeric identifier of this slot.
5. the `addModuleFn` function is a function that adds models to the construction and ensures that the correct `tag` is set. It is defined in the `baseConfig.lua` in `ConstructWithModules`.
6. the `params` block containing all params that the construction got for its `updateFn` function too.

Note that not all modules need every parameter. It is possible to omit parameters from the end of the function header. A function that uses only some of the parameters could look like:

```
...
updateFn = function(result, transform, tag)
  result.models[#result.models + 1] = {
    id = "station/air/airfield/hangar.mdl",
    transf = transform,
    tag = tag
  }

  local faces = { {-25.0, -25.0, 0.0, 1.0}, {25.0, -25.0, 0.0, 1.0}, ... }
  modulesutil.TransformFaces(transform, faces)
  result.groundFaces[#result.groundFaces + 1] = {
    face =  faces,
    modes = {
      {
        type = "FILL",
        key = "airfield_hangar.lua",
    texCoords = { {.0, .0}, {1.0, .0}, {1.0, 1.0}, {.0, 1.0} }
      },
    }
  }

  -- add further stuff
end,
...
```

Make sure to apply the transformation matrix received as `transform` parameter to all models, ground faces, terrain alignments, … of the module before adding them to the `result`.

The `updateFn` should return the `result` that was received where the content of the module was inserted.

Module GetModelsFn

The `getModelsFn` is a function without parameters. It returns a list of models that should be used for the floating preview that follows the mouse cursor while not pointing on a slot. The function looks like:

```
...
getModelsFn = function()
  local result = {
    {
      id = "station/air/airfield/hangar_preview.mdl",
      transf = transf.scale(vec3.new(1, 1, 1)),
    }
  }
  return result
end,
...
```

Construction Types

Ground Textures