Creating Custom Cargo

Project: **SnowRunner™**

Version 0.9.9 of Feb 2, 2022

Contents

Revision History

Version	Changes
0.9.8	The following sections were added or modified: • 7.4. Update of Long Logs Mechanics in DLC 6 (“Haul & Hustle”) was added (with subsections). • 7.5. Note On Slots in the Case of Logs as Cargo was added. • 8.2. Adding Cargo Items to the Level was modified.

1. Introduction

This guide provides brief instructions on creating a **custom cargo** as a part of the truck mod or the map mod.

2. Concept and Main Entities

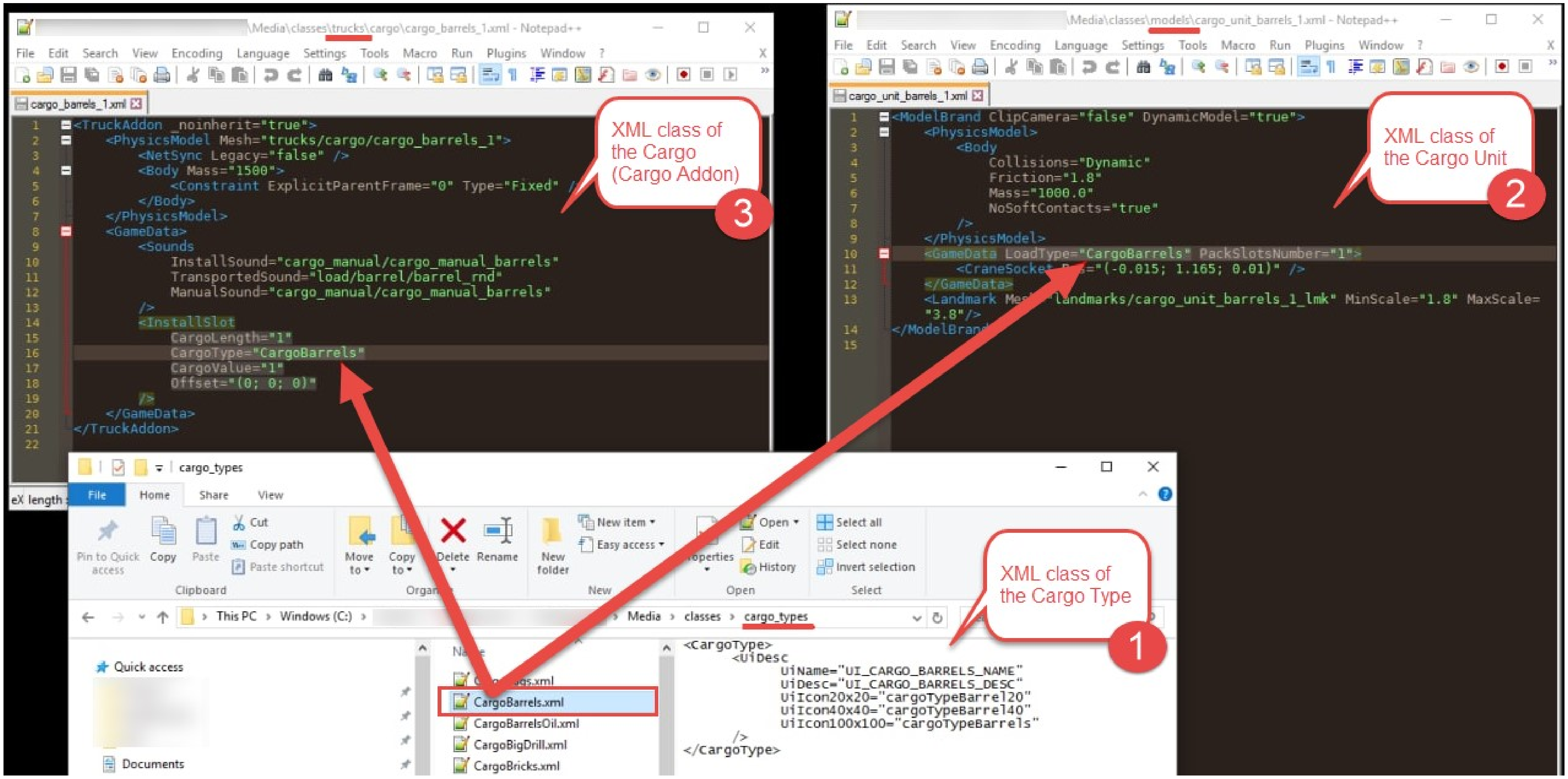
The concept of Cargo in SnowRunner™ is based on the following main entities:

- **Cargo Type** – the type of Cargo. The main purpose of this entity is to link the Cargo Unit and the Cargo Addon to each other. I.e., both the Cargo Unit and the Cargo Addon must have the same value of the Cargo Type within their XML classes since this allows the game to identify them as linked entities. Along with that, the XML description of the Cargo Type contains the name and description of this type of Cargo for UI (in the form of standard `UiName` and `UiDesc` fields) and names of icons used for this Cargo Type in the UI.
  Names of Cargo Types typically start with the "**Cargo**" prefix and are in the UpperCamelCase, e.g. **CargoBarrels**.

- **Cargo Unit** – the cargo unit of Cargo when it is outside of a truck (more precisely, when it is not packed within a truck or a trailer). I.e., this entity corresponds to a unit of cargo in the game environment, when it is not attached to a truck or a trailer and behaves similarly to a regular dynamic model. However, in addition to the properties of a regular dynamic model, the XML description of this entity must contain properties specific to Cargo Unit. Particularly, its XML class must contain the name of the Cargo Type specified in the `LoadType` attribute of the `GameData` tag and other specific properties.
  Names of cargo unit files typically start with the "**cargo_unit_**" prefix and are in the snake_case, e.g. **cargo_unit_barrels_1.xml**

- **Cargo** (so-called "**Cargo Addon**" to distinguish from Cargo in general) – the piece of Cargo when it is packed within a truck or a trailer. I.e., this entity corresponds to a Cargo when it is attached to a truck or a trailer and behaves similarly to a regular addon of the truck (moves along with the truck, and so on). However, in addition to the properties of a regular addon, the XML description of this entity must contain properties specific to Cargo Addon. Particularly, its XML class must contain the name of the Cargo Type specified in the `CargoType` attribute of the `InstallSlot` tag (within `GameData`) and other specific properties.
  Names of cargo addon files typically start with the "**cargo_**" prefix and are in the snake_case, e.g. **cargo_barrels_1.xml**

This scheme can be illustrated by the 3 opened XML files:


\Media\classes\trucks\cargo\cargo_barrels_1.xml - Notepad++
cargo_barrels_1.xml
XML class of the Cargo (Cargo Addon)
3
\Media\classes\models\cargo_unit_barrels_1.xml - Notepad++
cargo_unit_barrels_1.xml
XML class of the Cargo Unit
2
cargo_types
This PC › Windows (C:) › Media › classes › cargo_types
XML class of the Cargo Type
1
CargoBarrels.xml
CargoBarrelsOil.xml
CargoBigDrill.xml
CargoBricks.xml

```xml
<TruckAddon _noinherit="true">
    <PhysicsModel Mesh="trucks/cargo/cargo_barrels_1">
        <NetSync Legacy="false" />
        <Body Mass="1500">
            <Constraint ExplicitParentFrame="0" Type="Fixed" /
        </Body>
    </PhysicsModel>
    <GameData>
        <Sounds
            InstallSound="cargo_manual/cargo_manual_barrels"
            TransportedSound="load/barrel/barrel_rnd"
            ManualSound="cargo_manual/cargo_manual_barrels"
        />
        <InstallSlot
            CargoLength="1"
            CargoType="CargoBarrels"
            CargoValue="1"
            Offset="(0; 0; 0)"
        />
    </GameData>
</TruckAddon>
```

```xml
<ModelBrand ClipCamera="false" DynamicModel="true">
    <PhysicsModel>
        <Body
            Collisions="Dynamic"
            Friction="1.8"
            Mass="1000.0"
            NoSoftContacts="true"
        />
    </PhysicsModel>
    <GameData LoadType="CargoBarrels" PackSlotsNumber="1">
        <CraneSocket Pos="(-0.015; 1.165; 0.01)" />
    </GameData>
    <Landmark Mesh="landmarks/cargo_unit_barrels_1_lmk" MinScale="1.8" MaxScale=
    "3.8"/>
</ModelBrand>
```

```xml
<CargoType>
    <UiDesc
        UiName="UI_CARGO_BARRELS_NAME"
        UiDesc="UI_CARGO_BARRELS_DESC"
        UiIcon20x20="cargoTypeBarrel20"
        UiIcon40x40="cargoTypeBarrel40"
        UiIcon100x100="cargoTypeBarrels"
    />
</CargoType>
```

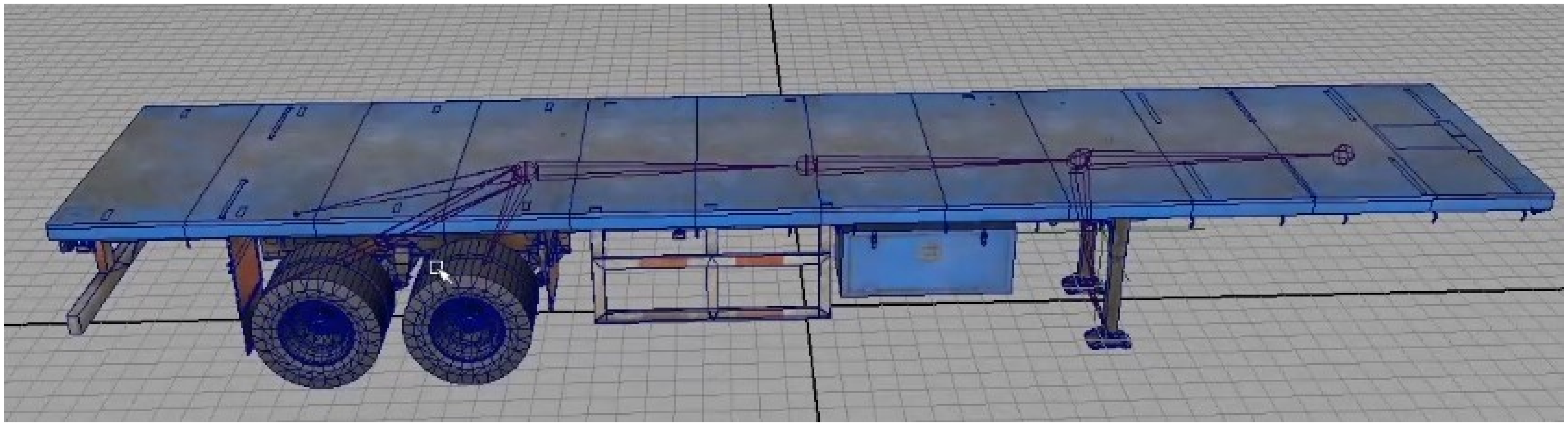
NOTE: However, along with the 3 main files of these entities displayed in the illustration above, there are other files of custom cargo required for its correct operation, see below.

2.1. Slots: Loading and Placement of Cargo

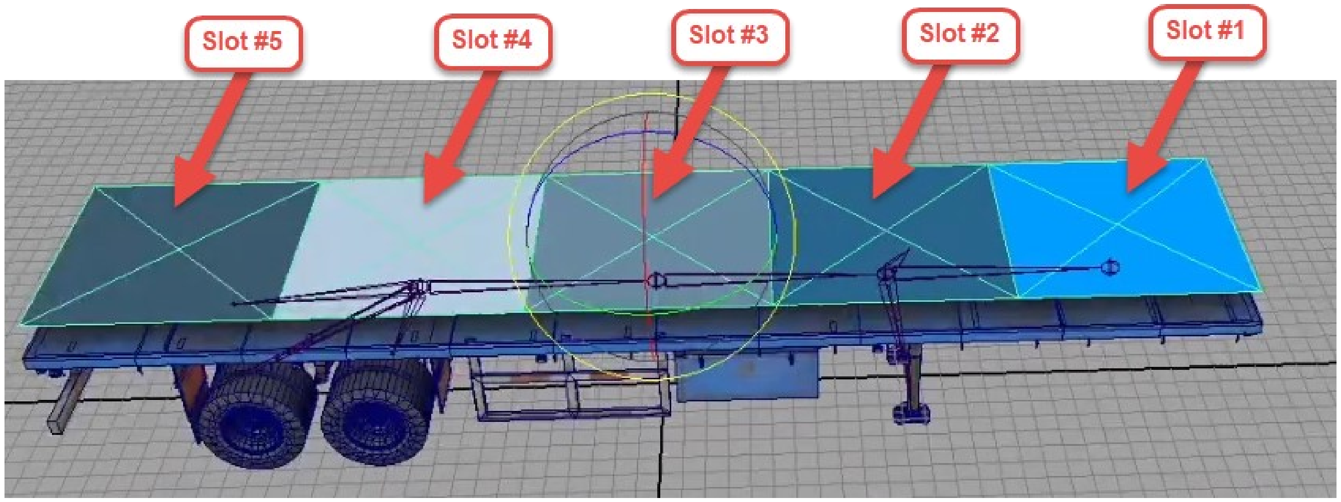
The concept of loading and placement of cargo is based on **Slots** (so-called **Addon Slots**).

Below we will illustrate this concept in the example.

Let's assume that we have a semi-trailer:



To be able to load cargo to it, we need to identify **Slots** on it – virtual areas on this semi-trailer where cargo items can be placed.

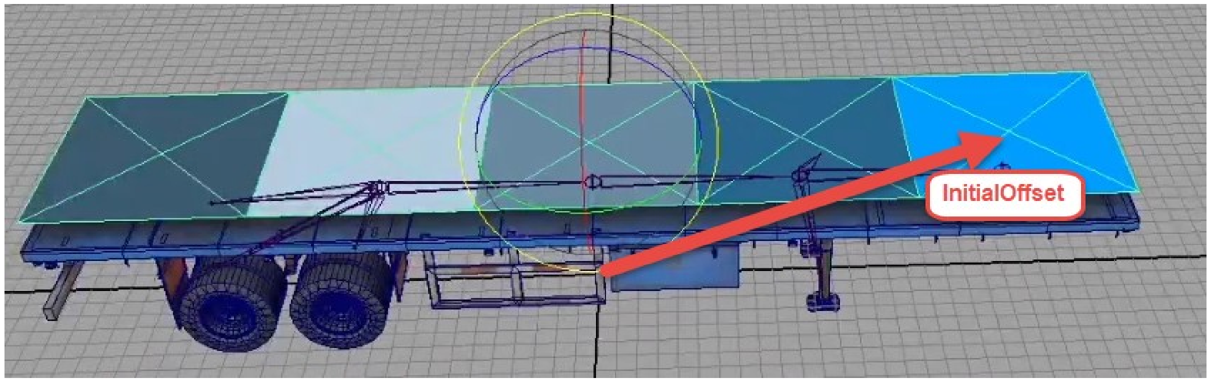


(The colored plane with slots is created for illustration purposes only, it does not exist in the geometry.)

After we have identified the necessary Slots, we need to specify the locations of these slots on the trailer, their quantity, and the bones of the trailer they are attached to in the description of the trailer.

So, in the XML file of the class of the trailer, we will specify these settings using the `<AddonSlots>` tag with the following attributes:

- `InitialOffset` – the coordinate of the center of the 1st slot, in the coordinates of the truck/frame addon/trailer. Using this value and the `OffsetStep` (see below), the system will be able to identify the coordinates of all slots.



- `OffsetStep` – sets the distance between centers of two adjacent slots, used to identify coordinates of all other slots by the coordinate of the 1st slot. However, please note that the negative values (e.g. `"(-2.559; 0; 0)"`) can also be specified in this field and the sign of these values specifies the *direction* in which the next slot is located (if there are multiple slots).
- `ParentFrames` – bones of the truck/frame addon/trailer these slots are attached to (the order corresponds to the order of the slots). When you specify `ParentFrames`, the number of bones listed for slots should be the same as the value of the `Quantity` field (see below). If multiple slots are attached to the same bone, it should appear in the list multiple times (like in the sample below).

  NOTE: If necessary, you can attach all slots to the same bone (e.g. to the root bone, by omitting the `ParentFrames` attribute).

- `Quantity` – the number of slots. Must correspond to the number of bones listed in `ParentFrames`, see above.

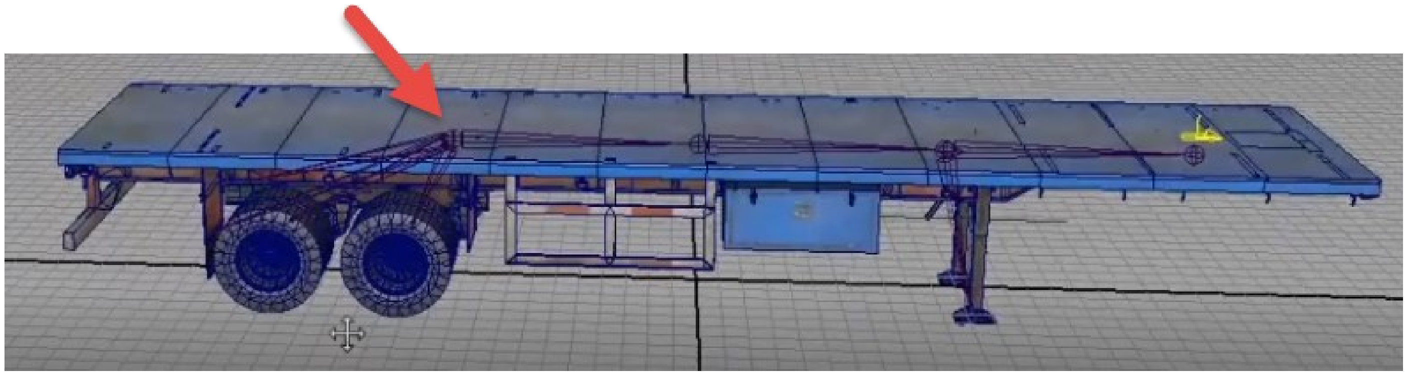
As you can see, the `<AddonSlots>` tag describes *all* slots of the trailer.

For example, the `<AddonSlots>` tag for the semi-trailer above will look like the following (a piece of **semitrailer_flatbed_5.xml**):

```
...
```

```
            <AddonSlots
                  InitialOffset="(5.118; 1.705; 0)"
                  OffsetStep="(-2.559; 0; 0)"
                  ParentFrames="BoneCargo_4_cdt,BoneCargo_3_cdt,BoneCargo_2_cdt, BoneCargo_1_cdt,
BoneCargo_1_cdt"
                  Quantity="5"
            />
...
```

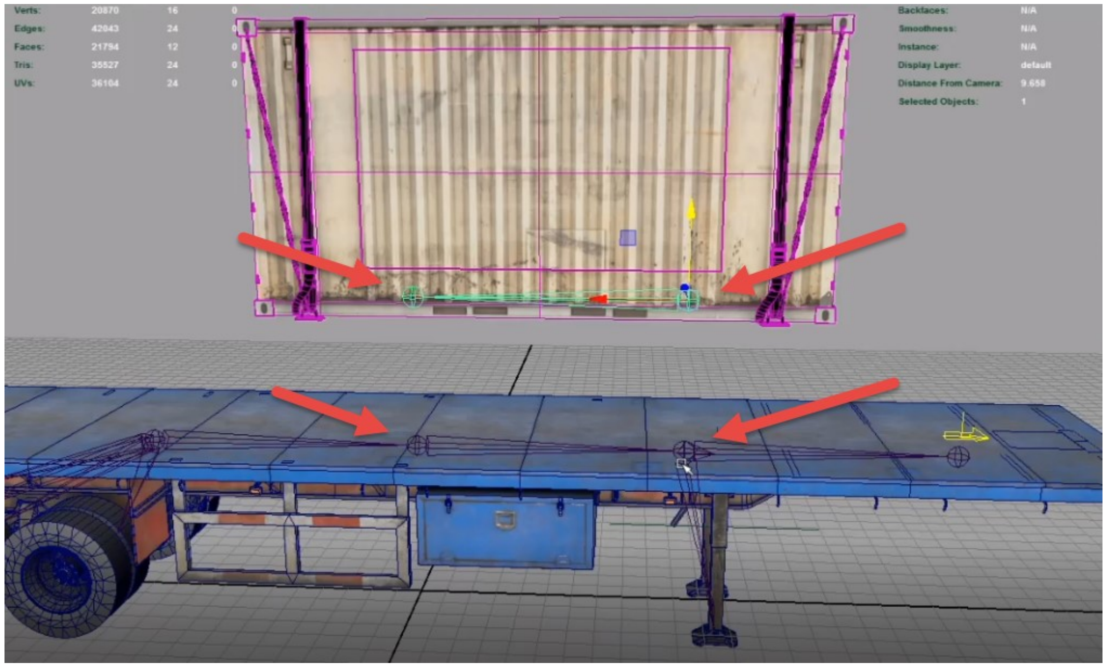
You can also see that our semi-trailer has only 4 bones (the end of it should not bend, it will be not realistic). So, both 4th and 5th slots will be attached to the same bone (`BoneCargo_1_cdt`):



In general, the approach of attaching slots to separate bones will allow us to slightly deform the cargo item attached to slots with the deformation of the truck/frame addon/trailer.

Now we have described slots for the cargo. However, we need some mechanics to attach a cargo item (Cargo Addon) to these slots. Moreover, our cargo item may occupy multiple slots, may have multiple bones too, and we want it to deform (slightly) with the truck/frame addon/trailer. Moreover, the cargo item can be attached to different slots depending on their occupancy by other cargo.

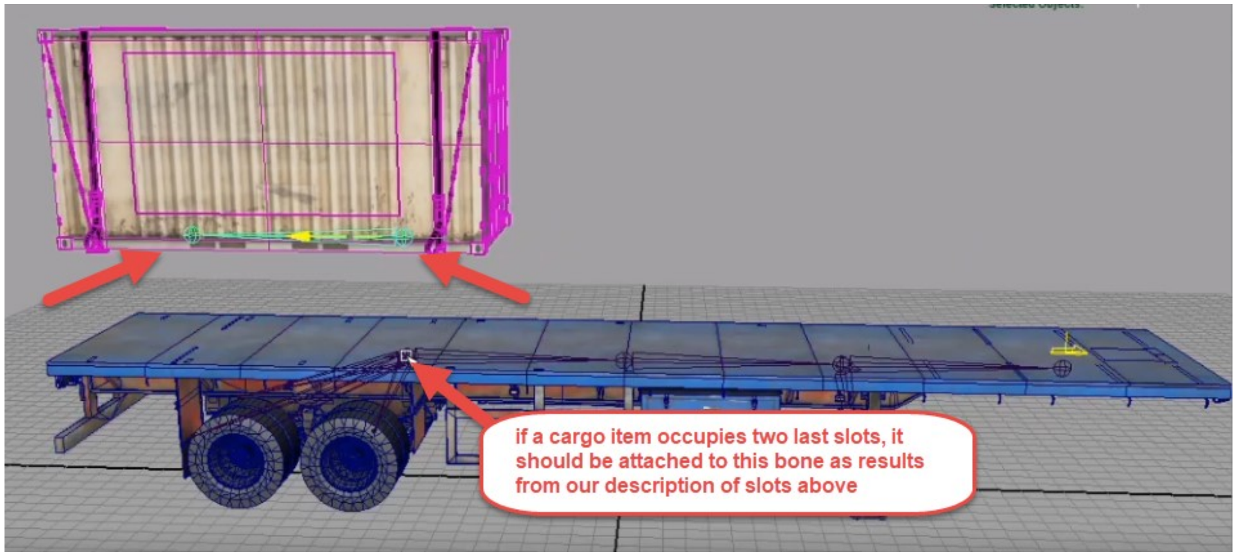
For example, we want a 2-slot container to bend a little along with our trailer:



```
Verts:      20870   16   0
Edges:      42043   24   0
Faces:      21794   12   0
Tris:       35527   24   0
UVs:        36104   24   0
```

```
Backfaces:                N/A
Smoothness:               N/A
Instance:                 N/A
Display Layer:            default
Distance From Camera:     9.658
Selected Objects:         1
```

And it may occupy different slots on the trailer (and should be attached to different bones depending on that):



if a cargo item occupies two last slots, it should be attached to this bone as results from our description of slots above

So, we need a system to describe all that.

We will do that by parenting bones of the cargo item (Cargo Addon) to slots, or, more precisely, to bones that we have specified for these slots the `ParentFrames` (see above).

Particularly, in the XML class of the Cargo Addon, within the description of the physical model (`<PhysicsModel>`) of this Cargo Addon, for the necessary bones of this model (`<Body>` tags), we will use the `ExplicitParentFrame` attribute of the `<Constraint>` tag to do that. The values of the `ExplicitParentFrame` attribute (`"0"`, `"1"`, …) will set the order of attachment of these bones to slots.

For example:

Sample file: **cargo_container_small_2.xml**

```
<TruckAddon _noinherit="true">
      <PhysicsModel Mesh="trucks/cargo/cargo_container_small_2">
            <NetSync Legacy="false" />
            <Body Mass="1500" ModelFrame="BoneRoot_cdt">
                  <Constraint ExplicitParentFrame="0" Type="Fixed" />
                  <Body Mass="1500" ModelFrame="BoneCargo_1_cdt">
                        <Constraint ExplicitParentFrame="1" Type="Fixed" />
                  </Body>
            </Body>
      </PhysicsModel>
      <GameData>
            <Sounds
                  InstallSound="cargo_manual/cargo_manual_container_small"
                  TransportedSound="load/metal_small/metal_small_rnd"
                  ManualSound="cargo_manual/cargo_manual_container_small"
            />
            <InstallSlot
                  CargoLength="2"
                  CargoType="CargoContainerSmall"
                  CargoValue="1"
                  Offset="(1.24; 0; 0)"
            />
      </GameData>
</TruckAddon>
```

NOTE: For a description of other properties of this file, see [5.3.1. XML Class of the Cargo Addon]() below.

As you can see, the physical model described above has two bones (`<Body>` tags) within which the `ExplicitParentFrame` is used:

```
<Body Mass="1500" ModelFrame="BoneRoot_cdt">
      <Constraint ExplicitParentFrame="0" Type="Fixed" />
      <Body Mass="1500" ModelFrame="BoneCargo_1_cdt">
            <Constraint ExplicitParentFrame="1" Type="Fixed" />
      </Body>
</Body>
```

The code above will set that these bones will be parented to slots of our semi-trailer, depending on their occupancy, particularly, to the bones specified previously in the `ParentFrames`:

```
ParentFrames="BoneCargo_4_cdt,BoneCargo_3_cdt,BoneCargo_2_cdt, BoneCargo_1_cdt, BoneCargo_1_cdt"
```

The order of their attachment will correspond to the order of bones in `ParentFrames` and the occupancy of the corresponding slots of the semi-trailer.

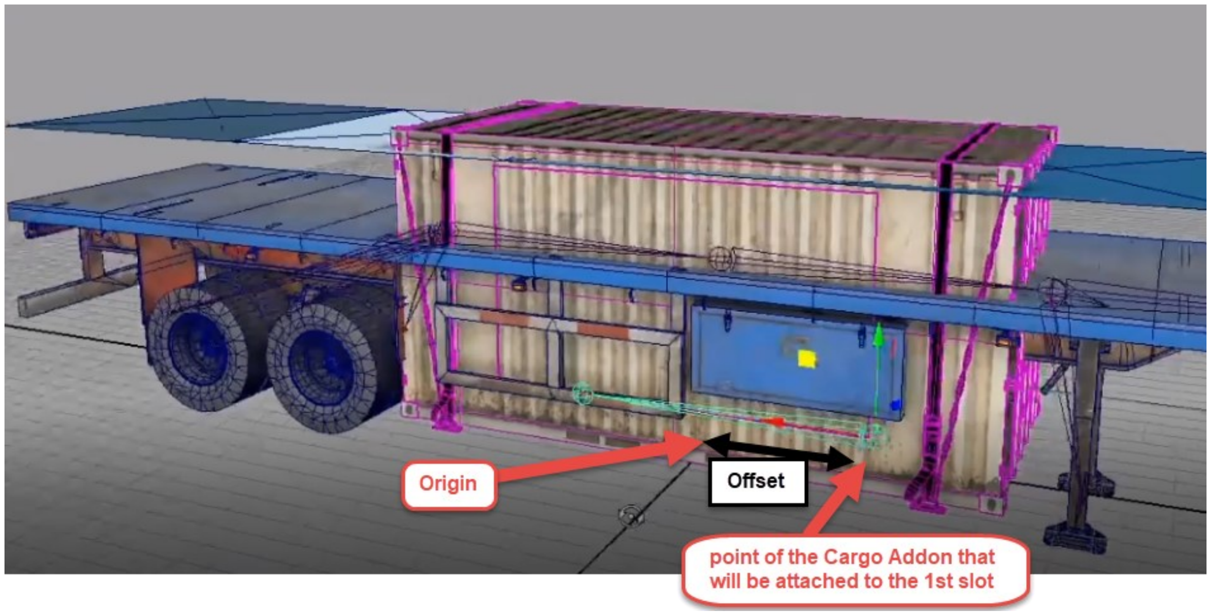
For example, if the semi-trailer has free slots that correspond to `BoneCargo_2_cdt` and `BoneCargo_1_cdt`, then the bone of the Cargo Addon that has `ExplicitParentFrame="0"` will be attached to `BoneCargo_2_cdt` and the bone with `ExplicitParentFrame="1"` will be attached to `BoneCargo_1_cdt`.

NOTE #1: If the `ParentFrames` were not specified at all, the system would attach all bones with `ExplicitParentFrame` of the Cargo Addon to the root bone of the trailer.

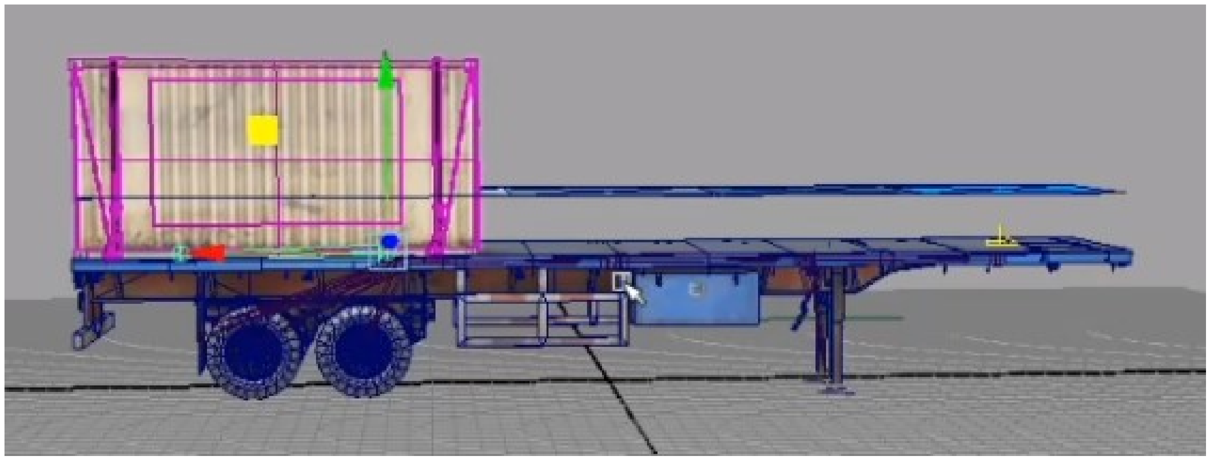
NOTE #2: If we had not specified the `ExplicitParentFrame` attributes *at all* within the Cargo Addon description, the root bone of this Cargo Addon (`BoneRoot_cdt` above) would be attached to the bone from the `ParentFrames` corresponding to the first free slot. However, this would be not realistic, the Cargo Addon would intersect with the geometry of the trailer on bumps, etc.

The game identifies the occupancy of the slot by the fact that there is a geometry above it. I.e., the position of the geometry of the corresponding Cargo Addon should match the position of the corresponding slot. If these positions do not match and there is no geometry above the slot, the system will treat this slot as free. I.e., it will not be able to identify that the Cargo Addon is loaded, and the player will be able to load an infinite number of cargo items using the Loading pop-up.

To avoid it, you should always model Cargo Addons so that the **position of *the "1st slot"* of this Cargo Addon** (i.e. the point of the model of this Cargo Addon that will be attached to the 1st slot) **is in the origin of coordinates**.
If you model it so that the *center* of the bottom surface of the Cargo Addon is in the origin, you will need to specify an `Offset` in the XML class of the Cargo Addon (see [5.3.1]()) to shift the position of the "1st slot" of this Cargo Addon to origin:


Origin
Offset
point of the Cargo Addon that will be attached to the 1st slot

However, if you model a Cargo Addon incorrectly, for example, like this:



Or, if you specify the incorrect value of the `InitialOffset` (see above) in the XML file of the class of the truck/frame addon/trailer – you may have infinite loading and other issues.

3. Files of Custom Cargo

To implement the custom cargo you need to create the files listed below.

WARNING #1: However, the location of these files depends on the type of the target mod, in which you want to include your custom cargo. Particularly:

- If you want to add your custom cargo to the **map mod**, you will need to put files to the locations specified below in the table.
- If you want to add your custom cargo to the **truck mod**, you will need to put all these files in the folder of the truck mod, at the same time keeping the folder hierarchy (but without the **Media** folder). For example, instead of putting the Cargo Type file to the **Media\classes\cargo_type**, you will need to put it into the **Media\Mods\<name_of_truck_mod>\classes\cargo_types**.

#	***The entity you need to add***		
#.#	***Folder (in case of the Map mod, see warning #1 above)***	***Files***	***Comments***
1.	***Entity: Cargo Type***		
1.1	**Media\classes\cargo_types**	**.xml** file of the class of the Cargo Type.	Contains information on the representation of the Cargo Type in the UI: • Fixed texts for UI or localization identifiers (UI_IDENTIFIERS) in case of localization. • identifiers of UI icons For example: **CargoBarrels.xml**

1.2	**Optionally, in the case of Localization:** in the folder of every map: **Media\prebuild\<map_name>\texts** **OR** in the folder of the truck mod: **Media\Mods\<truck_mod_name>\texts**	**.str** files with localization values for UI_IDENTIFIERS	Contain localization strings for each UI_IDENTIFIERS, see https://snowrunner.mod.io/guides/uiname-uidesc-names-descriptions-and-their-localization and the "*5.19. Localization of Maps*" section in the "**SnowRunner Editor Guide**" guide (**SnowRunner_Editor_Guide.pdf** in the same documentation package).
1.3	in the folder of every map: **Media\prebuild\<map_name>\ui\textures\specialIcons** **OR** in the folder of the truck mod: **Media\Mods\<truck_mod_name>\ui\textures\specialIcons**	**.png** files of icons of Cargo Type for UI	Icons of this Cargo for the UI. See 5.1.3 below.
2.	***Entity: Cargo Unit*** ***NOTE: The set of files is similar to the set of files of a regular custom model, see the "Custom Level Entities. Models, Overlays" guide for details, available as Custom_Level_Entities_Models_Overlays.pdf of the same documentation package)***		
2.1	**Media\meshes\models**	**.fbx** file of the mesh of the model of Cargo Unit **.xml** file of the mesh of the model of Cargo Unit	The **.fbx** file of the model contains its mesh and its collision mesh. The **.xml** file of the mesh, stored in the same folder, contains data on the materials (textures) of the model.
2.2	**Media\textures\models**	**.tga** files of textures of the model of Cargo Unit	This folder should contain all textures that were assigned to the mesh of the model in the **.xml** file of the mesh. **NOTE**: You can use the **Media\textures\trucks** folder for textures too (since some textures of Cargo Unit and Cargo Addon will probably be the same).

2.3	**Media\classes\models**	**.xml** file of the class of (the model) of the Cargo Unit	The **.xml** file of the class of the model defines its properties in the environment: instancing, physical model, collisions, etc. Along with that, for a Cargo Unit, this class must contain the name of the Cargo Type and other specific properties, see below.
	Optional (if you want to create a landmark model for your model):		
2.4	**Media\meshes\landmarks**	**.fbx** file of the landmark model of the Cargo Unit **.xml** file of the mesh of the landmark model of the Cargo Unit	If you want your model to appear on the navigation map, the **.fbx** file and **.xml** file of the mesh need to be created for the *landmark model* of the Cargo Unit. This is done the same way as for regular models. See the "*3.7. Optional: Landmark model*" section of the “**Custom Level Entities. Models, Overlays**” guide for details, available as **Custom_Level_Entities_Models_Overlays.pdf** of the same documentation package) for details.
3.	***Entity: Cargo (Cargo Addon)*** ***NOTE: The set of files for the Cargo Addon is almost similar to the set of files for a regular Addon to the truck (except the sound files).***		
3.1	**Media\meshes\trucks\cargo**	**.fbx** file of the mesh of the model of Cargo Addon **.xml** file of the mesh of the model of Cargo Addon	The **.fbx** file of the cargo addon contains its mesh and its collision mesh. The **.xml** file of the mesh of the cargo addon, stored in the same folder, contains data on the materials (textures) of the model.

3.2	**Media\classes\trucks\cargo**	**.xml** file of the class of the Cargo Addon.	The **.xml** file of the class of the Cargo Addon defines its properties as the Addon to the truck: physical model, etc. Along with that, for a Cargo Addon, this class must contain the name of the Cargo Type and other specific properties, see below. Optionally, links to the sounds of the Cargo can be also provided: the sound of installation of the cargo to the truck (`InstallSound`), etc.
3.3	**Media\textures\trucks**	**.tga** files of textures of the model of Cargo Addon	This folder should contain all textures that were assigned to the mesh of the cargo addon in the **.xml** file of the mesh. **NOTE**: You can use the **Media\textures\models** folder for textures too (since some textures of Cargo Unit and Cargo Addon will probably be the same).
	Optional (if you want to specify sounds for your Cargo):		
3.4	Typically: • **Media\sounds\cargo_manual** – for `InstallSound` and `ManualSound` • The subfolder in the **Media\sounds\load** – for the `TransportedSound` set of sounds. However, you can use any folders within the **Media\sounds**, if necessary.	**.wav** files of sounds of Cargo	These folders should contain all sounds that were assigned to the Cargo in the **.xml** file of the class of the Cargo Addon.

WARNING #2: If you are adding your custom cargo to the particular **map mod** and, correspondingly, are putting files of the cargo to the **Media** folder, then, after adding these files, this custom cargo will be added to ALL mods of maps and regions that you will be packing with the SnowRunner™ Editor. To avoid it, you will need to remove these files after packing all necessary map mods with target custom cargo.

All folders listed above must be created in the **Media** folder, which is located in the **Documents\My Games\SnowRunner\** folder. The full path to the **Media** folder is typically similar to the following:

C:\Users\<name_of_user>\Documents\My Games\SnowRunner\Media\

4. Naming Convention

We recommend the following naming scheme (however, it is not mandatory):

- **Cargo Type** – Names of Cargo Types typically start with the “**Cargo**” prefix and are in the UpperCamelCase, e.g. **CargoBarrels**.
- **Cargo Unit** – Names of cargo unit files typically start with the “**cargo_unit_**” prefix and are in the snake_case, e.g. **cargo_unit_barrels_1.xml**
- **Cargo** ("**Cargo Addon**") – Names of cargo addon files typically start with the “**cargo_**” prefix and are in the snake_case, e.g. **cargo_barrels_1.xml**

The names of all *folders* (related to custom cargo) within the **Media** folder must be in lowercase, with the **specialIcons** folder as an exception (see 5.1.3 below).

Names of images of Cargo for UI are up to you; however, we recommend using the `20`, `40`, and `140` suffixes at the end of their names for easier identification (see 5.1.1 and 5.1.3 below).

5. Brief Notes on Creating Files

5.1. Cargo Type

5.1.1. XML Class of the Cargo Type

The XML class of the Cargo Type is very simple.

Its main purpose – to define a Cargo Type and provide a possibility to link the Cargo Unit to the Cargo Addon – is achieved by its file name.

For example, if you name this file as **CargoBarrels.xml**, the name of the new cargo type that you will need to use in other files will be **CargoBarrels**.

Particularly, you will need to specify the name of this type (e.g. **CargoBarrels**) in the following files:

- In the XML class of the Cargo Unit – in the `LoadType` attribute of the `GameData` tag.
- In the XML class of the Cargo Addon – in the `CargoType` attribute of the `InstallSlot` tag (which is also one of the child-tags of `GameData`).

Other properties specified within this file include:

- The name and description of this type of Cargo for UI – in the form of standard `UiName` and `UiDesc` fields, see 5.1.2 below.
- Names of icons used for this Cargo Type in the UI – see 5.1.3 below.

This XML class of the Cargo Type must be created:

- For a new cargo packed into a map mod – in the **Media\classes\cargo_types** folder.
- For a new cargo packed into a truck mod – in the **Media\Mods\<name_of_truck_mod>\classes\cargo_types** folder.

Sample File: **CargoBarrels.xml**

```
<CargoType>
	<UiDesc
		UiName="UI_CARGO_BARRELS_W_ORANGES_NAME"
		UiDesc="UI_CARGO_BARRELS_W_ORANGES_DESC"
		UiIcon20x20="cargoTypeBarrelWithOranges20"
```

```
            UiIcon40x40="cargoTypeBarrelWithOranges40"
            UiIcon100x100="cargoTypeBarrelWithOranges140"
      />
</CargoType>
```

5.1.2. Localization Files

The `UiName` and `UiDesc` fields of the XML class of the Cargo Type (see above) work as regular localizable text fields.

You can specify values for the name and description of the Cargo directly in this XML class of the Cargo Type. In this case, they will appear in the UI as is and will be not localizable. And, in this case, you will need no localization files, so it is a good option if you want to quickly create a draft version of your custom cargo.

Or, you can create the appropriate localization files (**.str** files) and specify the localization identifiers (UI_IDENTIFIERS) for them in the `UiName` and `UiDesc` fields. For details, see our short guide on localization: https://snowrunner.mod.io/guides/uiname-uidesc-names-descriptions-and-their-localization.

If you prefer to use the localization approach with UI_IDENTIFIERS, you will need to pack the **.str** files with the localizations for these UI_IDENTIFIERS with your mod.

Particularly, in the current version of the game, you will need to put these **.str** files in the following folders:

- For a new cargo packed into a map mod – in the **texts** folder within the folder of the map (the **Media\prebuild\<name_of_map>\texts** folder).

  NOTE: These **.str** files must be placed into (the **texts** subfolder within) folder of ***every map*** to which you want to attach the custom cargo. Moreover, if you want to use this custom cargo in a Region consisting of multiple maps, you will need to put these files into (the **texts** subfolder within) folder of every map of the Region, even if this type is used only on one of its maps.

- For a new cargo packed into a truck mod – in the **Media\Mods\<name_of_truck_mod>\texts** folder.

5.1.3. Icons of Cargo for UI

The `UiIcon...` fields of the XML class of the Cargo Type (see above in 5.1.1) allow you to specify the names of icons used for this Cargo Type in the UI.

Particularly:

- `UiIcon20x20` – corresponds to the small icon of Cargo displayed while tracking objectives and on the navigation map. Dimensions of the image: **20x20 px.**

- `UiIcon40x40` – corresponds to the medium icon of Cargo displayed in the Unloading pop-up and as a marker on the navigation map. Dimensions of the image: **40x40 px.**

- `UiIcon100x100` – corresponds to the large icon of Cargo displayed in the Loading pop-up. Dimensions of the image: **140x140 px (!).**

  WARNING: Please note that for the large icon of Cargo (the `UiIcon100x100` field) you must use ***not*** the **100x100px** image, but, instead, the **140x140px** image!

As usual, these fields must contain the names of the files of the images, without the file extension.

The files of the images themselves should be in **PNG** format.

They should be put in the following folders:

- For a new cargo packed into a map mod – in the **ui\textures\specialIcons** folder within the *folder of the map*, i.e. in the **Media\prebuild\**<name_of_map>**\ui\textures\specialIcons** folder.

  NOTE: These images must be placed into the folder specified above for ***every map*** to which you want to attach the custom cargo. Moreover, if you want to use this custom cargo in a Region consisting of multiple maps, you will need to put these images into corresponding folders of every map of the Region, even if this type is used only on one of its maps.

- For a new cargo packed into a truck mod – in the **ui\textures\specialIcons** folder within the *folder of the truck mod*, i.e. in the **Media\Mods\**<name_of_truck_mod>**\ui\textures\specialIcons** folder.

5.2. Cargo Unit

5.2.1. XML Class of the Cargo Unit

The XML file of the class of the Cargo Unit is very much similar to the XML class of the regular model.

It consists of two parts:

- **1st part – Description of the model** – this part is absolutely the same as the XML class of any custom model. It defines properties of the model of the Cargo Unit in the environment: physical model, collisions, etc.
  For details on creating such a description, see the “**Custom Level Entities. Models, Overlays”** guide for details, available as **Custom_Level_Entities_Models_Overlays.pdf** of the same documentation package).

  NOTE: For obvious reasons, the described custom model should be dynamic, not static. Otherwise, the player will not be able to move the Cargo Unit.

- **2nd part – Properties specific for a Cargo Unit** – this part must contain the name of the Cargo Type and other specific properties, see below.

The properties of the 2nd part are mostly specified as attributes of the `<GameData>` tag and its child tags (see the sample below).

Particularly, you need to specify the following attributes of the `<GameData>` tag:

- `LoadType` – the Cargo Type of this Cargo Unit. Here you need to specify the name of the necessary Cargo Type XML class file without the extension, e.g. `"CargoBarrels"` for **CargoBarrels.xml**. This field links the Cargo Unit to the Cargo Type and, through it, to the Cargo Addon.
- `PackSlotsNumber` – the number of slots that will be filled by this Cargo Unit when it will be loaded to a truck or a trailer.

Along with it, typically there is also one or more `<CraneSocket>` tags within the `<GameData>` tag:

- Every `<CraneSocket>` tag allows you to set a point on the Cargo Unit for grabbing it with the crane.
- The `Pos` attribute of this tag specifies the position of this attachment point.

NOTE: The XML class of the Cargo Unit can contain additional properties, if you want to assign a Cargo Unit to the particular trucks or trailers (see 6. below) or in the case of custom logs as cargo (see 7. below).

This XML class of the Cargo Unit must be created:

- For a new cargo packed into a map mod – in the **Media\classes\models** folder.
- For a new cargo packed into a truck mod – in the **Media\Mods\<name_of_truck_mod>\classes\models** folder.

Sample File: **cargo_unit_barrels_1.xml**

```
<ModelBrand ClipCamera="false" DynamicModel="true">
	<PhysicsModel>
		<Body
			Collisions="Dynamic"
			Friction="1.8"
			Mass="1000.0"
			NoSoftContacts="true"
		/>
	</PhysicsModel>
	<GameData LoadType="CargoBarrels" PackSlotsNumber="1">
		<CraneSocket Pos="(-0.015; 1.165; 0.01)" />
	</GameData>
	<Landmark Mesh="landmarks/cargo_unit_barrels_1_lmk" MinScale="1.8" MaxScale="3.8"/>
</ModelBrand>
```

5.2.2. Other files: Mesh files, Textures, and Landmark Model

All other files of the Cargo Unit – mesh files (.fbx and .xml files of the mesh), textures, and, optionally, a landmark model – are created absolutely the same as if you were creating a regular custom model.

For details, see the "**Custom Level Entities. Models, Overlays"** guide for details, available as **Custom_Level_Entities_Models_Overlays.pdf** of the same documentation package).

The Mesh files of the model (.fbx + .xml) of Cargo Unit must be created:

- For a new cargo packed into a map mod – in the **Media\meshes\models** folder.
- For a new cargo packed into a truck mod – in the **Media\Mods\<name_of_truck_mod>\meshes\models** folder.

The texture files of the model (.tga) of the Cargo Unit must be created:

- For a new cargo packed into a map mod – in the **Media\textures\models** folder.
- For a new cargo packed into a truck mod – in the **Media\Mods\<name_of_truck_mod>\textures\models** folder.

> **NOTE**: You can use the **Media\textures\trucks** and **Media\Mods\<name_of_truck_mod>\textures\trucks** folders for textures too (since some textures of Cargo Unit and Cargo Addon will probably be the same). Actually, you can use any folder within **Media\textures** or **Media\Mods\<name_of_truck_mod>\textures** if you specify it correctly in the paths to textures.

The landmark model files (.fbx + .xml) of Cargo Unit must be created:

- For a new cargo packed into a map mod – in the **Media\meshes\landmarks** folder.
- For a new cargo packed into a truck mod – in the **Media\Mods\<name_of_truck_mod>\meshes\landmarks** folder.

5.3. Cargo (Cargo Addon)

5.3.1. XML Class of the Cargo Addon

The XML file of the class of the Cargo Addon is very much similar to the XML class of the regular addon to the truck.

It consists of two parts:

- **1st part – Description of the addon** – this part is similar to the XML class of any regular addon. It defines properties of the model of Cargo Addon as a part of the truck, the physical model of the addon within the truck, etc.
  However, for Cargo Addons, the **physical model** may also contain:
  - Bones with the `ExplicitParentFrame` attribute of the `<Constraint>` tag that will be attached to free Slots on the truck/frame addon/trailer. See 2.1. Slots: Loading and Placement of Cargo above for details.
- **2nd part – Properties specific for a Cargo Addon** – Particularly, the following:

- **Links to sounds** – links to the sounds of this Cargo: sounds of installation of the cargo to the truck (manual or automatic) and set of sounds played during transportation of this Cargo.

- **InstallSlot description** – the Cargo Type of this Cargo and some other properties, see below.

The necessary properties to be set in the **1st part** are specified in 2.1. Slots: Loading and Placement of Cargo above.

The properties of the **2nd part** are specified within the child-tags of the `<GameData>` tag (see the sample below).

Particularly, you need to specify the following child-tags with the following attributes within the `<GameData>` tag:

- `<Sounds>` tag:
  - `InstallSound` – *optional*, the path to the sound that will be played during automatic loading of this Cargo. See 5.3.3 below for details.
  - `TransportedSound` – *optional*, the path to the random set of sounds that will be played during the transportation of this Cargo (when cargo bounces on road bumps). See 5.3.3 below for details.
  - `ManualSound` – *optional*, the path to the sound that will be played during automatic loading of this Cargo. See 5.3.3 below for details.

- `<InstallSlot>` tag:
  - `CargoLength` – the number of slots that are filled by this Cargo Addon when it is within a truck or a trailer. Similar to the `PackSlotsNumber` of the Cargo Unit.
  - `CargoType` – the Cargo Type of this Cargo Unit. Here you need to specify the name of the necessary Cargo Type XML class file without the extension, e.g. `"CargoBarrels"` for **CargoBarrels.xml**. This field links the Cargo Addon to the Cargo Type and, through it, to the Cargo Unit.
  - `CargoValue` – *currently not used*, can be set to "1".
  - `Offset` – the offset for the "1st slot" of this Cargo Addon (i.e. for the point of this Cargo Addon that will be attached to the 1st slot of the truck/frame addon/trailer). If you have modeled the Cargo Addon so that the position of this point is in the origin of the Fbx file of the Cargo Addon, you can set this offset to `(0; 0; 0)`.
    See 2.1. Slots: Loading and Placement of Cargo for details.

NOTE: The XML class of the Cargo Unit can contain additional properties, if you want to assign a Cargo Unit to the particular trucks or trailers (see 6. below) or in the case of custom logs as cargo (see 7. below).

This XML class of the Cargo Addon must be created:

- For a new cargo packed into a map mod – in the **Media\classes\trucks\cargo** folder.
- For a new cargo packed into a truck mod – in the **Media\Mods\<name_of_truck_mod>\classes\trucks\cargo** folder.

Sample File: **cargo_barrels_1.xml**

```
<TruckAddon _noinherit="true">
      <PhysicsModel Mesh="trucks/cargo/cargo_barrels_1">
            <NetSync Legacy="false" />
            <Body Mass="1500">
                  <Constraint ExplicitParentFrame="0" Type="Fixed" />
            </Body>
      </PhysicsModel>
      <GameData>
            <Sounds
                  InstallSound="cargo_manual/cargo_manual_barrels"
                  TransportedSound="load/barrel/barrel_rnd"
                  ManualSound="cargo_manual/cargo_manual_barrels"
            />
            <InstallSlot
                  CargoLength="1"
                  CargoType="CargoBarrels"
                  CargoValue="1"
                  Offset="(0; 0; 0)"
            />
      </GameData>
</TruckAddon>
```

5.3.2. Other files: Mesh files and Textures

Other mandatory files of the Cargo Addon – mesh files (.fbx and .xml files of the mesh) of the model of the Cargo Addon and its textures – are created absolutely the same as if you were creating a regular addon.

The Mesh files of the model (.fbx + .xml) of Cargo Addon must be created:

- For a new cargo packed into a map mod – in the **Media\meshes\trucks\cargo** folder.
- For a new cargo packed into a truck mod – in the **Media\Mods\**<name_of_truck_mod>**\meshes\trucks\cargo** folder.

The texture files of the model (.tga) of Cargo Addon must be created:

- For a new cargo packed into a map mod – in the **Media\textures\trucks** folder.
- For a new cargo packed into a truck mod – in the **Media\Mods\**<name_of_truck_mod>**\textures\trucks** folder.

> **NOTE**: You can use the **Media\textures\models** and **Media\Mods\**<name_of_truck_mod>**\textures\models** folders for textures too (since some textures of Cargo Unit and Cargo Addon will probably be the same). Actually, you can use any folder within **Media\textures** or **Media\Mods\**<name_of_truck_mod>**\textures** if you specify it correctly in the paths to textures.

5.3.3. Optional: Sound files

Optionally, you can add sounds to your custom cargo.

The Format of these sound files needs to be the following: **.wav** file, **44 kHz**, **16 bit**, **Mono**

Links to these sounds are specified in the attributes of the `<Sounds>` tag, within the XML class of the Cargo Addon (see above).

Paths in these fields need to be specified relative to the **Media\sounds**, or **Media\Mods\**<name_of_truck_mod>**\sounds** folder, see below.

Typically, the sound files themselves are put into:

- For `InstallSound` and `ManualSound` :
  - For a new cargo packed into a map mod – in the **Media\sounds\cargo_manual** folder.
  - For a new cargo packed into a truck mod – in the **Media\Mods\**<name_of_truck_mod>**\sounds\cargo_manual** folder.

- For the `TransportedSound` set of sounds:

- For a new cargo packed into a map mod – in the **Media\sounds\load** folder.
- For a new cargo packed into a truck mod – in the **Media\Mods\<name_of_truck_mod>\sounds\load** folder.

NOTE: However, you can use any folders within the **Media\sounds**, or **Media\Mods\<name_of_truck_mod>\sounds**, if necessary.

The sound from the `TransportedSound` set of sounds will be played during the transportation of this Cargo when cargo bounces on road bumps. At every such moment, the sound from the set will be selected randomly. For details on working with sets of sounds, see the "*5.12.1. Adding Series of Sounds*" section in the **SnowRunner™ Editor Guide** (***SnowRunner_Editor_Guide.pdf*** in the same documentation package).

6. Assigning Cargo Types to Addons. ExcludedCargoTypes

For some types of custom cargo, you may face a serious difficulty – they will require *particular* trucks and trailers for transportation since not all types of cargo will fit all frame addons (for example, due to the geometry of the cargo).

However, you can prohibit the loading of particular cargo types to the particular addon.

To do this, you need to list all cargo types prohibited for loading to the particular addon in the XML class of this addon, within the `ExcludedCargoTypes` attribute of the `<GameData>` tag.

All other cargo types that are not listed will be allowed for loading. For example, you can see that **CargoContainerSmall** cargo type is missing in this list in the sample below. Thus, it will be allowed for the specified addon.

> **WARNING**: This approach for linking cargo types and addons also means that all new custom cargo types will be allowed for loading since they are not explicitly prohibited for the addon. Moreover, this also means that all original in-game trucks and trailers will allow loading of all new custom cargo types since this approach is used for them too and you are not able to modify their initial XML classes within .pak files during modding.

Sample file: **cat_745c_container_carrier.xml** (located within the **[media]\classes\trucks\addons** folder of the **initial.pak**)

```
<TruckAddon>
	<PhysicsModel Mesh="trucks/addons/cat_745c_container_carrier">
		<Body Mass="400">
			<Constraint Type="Rigid" />
			<Body Collisions="None" Mass="5" ModelFrame="BoneAddonFront_cdt">
				<Constraint ExplicitParentFrame="0" Type="Fixed" />
			</Body>
		</Body>
	</PhysicsModel>
	<GameData
		CameraPreset="addon_1"
		Category="frame_addons"
		IsCustomizable="true"
		Price="5700"
		UnlockByExploration="false"
		UnlockByRank="12"
```

```
            ExcludedCargoTypes = "CargoBags2, CargoCellulose, CargoRailway, CargoForcklift,
CargoForkliftCaravanContainer2, CargoSequoia, CargoLogsShort, CargoLogsMedium, CargoLogsLong, CargoBarrels,
CargoMetalPlanks, CargoWoodenPlanks, CargoMetalRoll, CargoWing1, CargoWing2, CargoServiceSpareParts,
CargoPipesMedium, CargoBags, CargoServiceSparePartsSpecial, CargoPlane, CargoCrateLarge, CargoRadioactive,
CargoContainerLarge, CargoConcreteSlab, CargoContainerLargeDrilling, CargoBigDrill, CargoRocks, CargoBA20,
CargoBarrelsOil, CargoVehiclesSpareParts, CargoPipesSmall, CargoBA20Add, CargoConcreteBlocks,
CargoPipeLarge, CargoBricks"
      >
            <UiDesc
                  UiDesc="UI_ADDON_CAT_745C_CONTAINER_CARRIER_DESC"
                  UiIcon30x30=""
                  UiIcon40x40=""
                  UiName="UI_ADDON_CAT_745C_CONTAINER_CARRIER_NAME"
            />
            <InstallSocket Type="Cat745CContainerCarrier" />
            <AddonSlots InitialOffset="(-1.219; 2.076; -0)" OffsetStep="(-3.198; 0; 0)" Quantity="2" />
      </GameData>
</TruckAddon>
```

7. Custom Logs as Cargo. CargoAddonSubtype

Along with regular Cargo Types (Barrels, Bricks, etc.), the game supports different types of logs as cargo. The main difference of this type of cargo from the regular types is that, in the case of logs, multiple Cargo Units loaded into the truck/trailer are transformed into a single Cargo Addon within the truck/trailer after packing.

> **WARNING**: Logs are a very specific type of cargo and the implementation of the "logging" feature is tuned specifically for logs. We recommend you to use the information in this chapter for the creation of new custom logs only. If you use it to create other custom types of cargo that will utilize the same principle, these types may not work as intended.

Particularly, the "logging" feature works as follows:

> You can manually load 3 units of logs into a truck/frame addon/trailer, locating them there specifically (Load Points of all cargo units should be inside Load Areas of the truck/frame addon/trailer). After that, you will be able to pack them, and they will be transformed into a single Cargo Addon that will be displayed as the large pack of the logs in this truck/frame addon/trailer. This pack will be so large that the truck/frame addon/trailer will appear full of short logs.
>
> **NOTE:** Currently, the number of cargo units (separate logs) that can be transformed into a single Cargo Addon (pack of logs) is fixed and equal to **3**. I.e., you need to manually load and correctly place 3 logs within a truck to transform them into a pack of logs.

In this process there are some important nuances:

1. The resulting Cargo Addon should perfectly match the particular truck/frame addon/trailer by its dimensions. I.e., along with linking 3 Cargo Units to a Cargo Addon, you also need to **link this Cargo Addon to a particular truck/frame addon/trailer**.

2. Moreover, you also need to set up **Load Areas within the truck/frame addon/trailer** and **Load Points on the Cargo Unit**.

3. The loading of logs is based on the functionality of regular slots (of the truck, frame addon, or trailer). Therefore, **the position of the geometry of the corresponding Cargo Addon (pack of logs)** that will appear within the truck after packing **should match the position of the corresponding slot.** If these positions do not match and there is no geometry above the slot, the system will treat this slot as free. I.e., it will not be able to identify that the Cargo Addon has been already created, and the player will be able to load an infinite number of log packs using the Loading pop-up.

The mechanics behind that is the following:

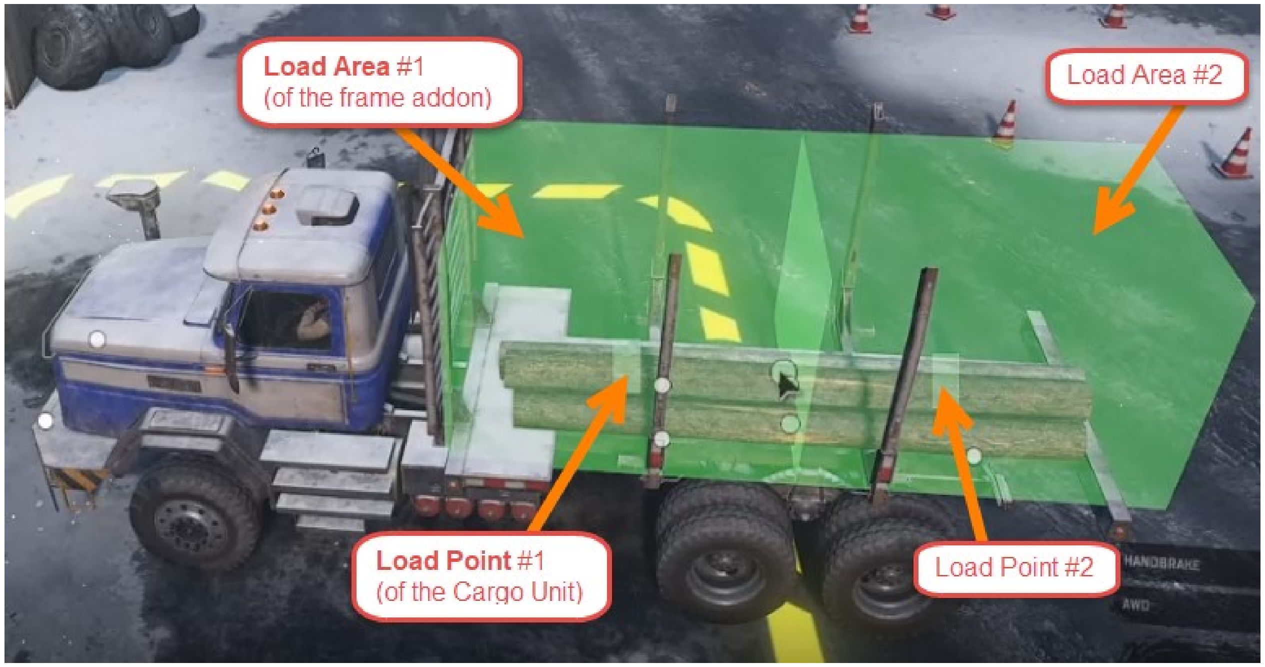
Main Terms:

- **Cargo Addon SubType** – this entity links the XML class of the Cargo Addon (pack of logs) to the XML class of the particular truck/frame addon/trailer where this Cargo Addon can appear. Particularly, the value of the `CargoAddonSubtype` attribute in the `<InstallSlot>` tag of the XML class of the Cargo Addon must have the same value as the `Subtype` attribute of the `<LoadArea>` tag of the XML class of the particular truck/frame addon/trailer to link these two files (and this value will be the name of the Cargo Addon SubType).

- **Load Area** – a volume within the truck/frame addon/trailer within which a load point of every Cargo Unit (every single log) must be located for the system to identify the fact that all cargo units are correctly placed within the truck/frame addon/trailer, which will enable packing of these cargo units into a Cargo Addon (pack of logs). See the picture below.

- **Load Point** – a point of the cargo unit that needs to be placed within the Load Area to enable packing. See the picture below.

  NOTE: All load points of all cargo units must be within Load Areas to enable packing.



The process of the setup:

To enable the “logging” mechanics for your custom logs, you need to modify 3 files:

1. XML class of the Cargo Unit.
2. XML class of the Cargo Addon (which will appear after packing of 3 Cargo Units of this type).
3. XML class of the truck/frame addon/trailer (where this Cargo Addon will appear).

See subsections below for more details.

7.1. Modifying XML Class of the Cargo Unit

In the XML class of the Cargo Unit, you need to specify Load Points and their positions on the Cargo Unit.

Load points are added by `<LoadPoint>` tags within the `<GameData>` tag. Their positions on the model of the Cargo Unit are specified within the `Pos` attribute of each `<LoadPoint>` tag.

Sample file: **cargo_unit_log_short.xml**

```
<_templates Include="models" />
<ModelBrand
      ClipCamera="false"
      DynamicModel="true"
>
      <PhysicsModel>
            <Body _template="MediumWeight" ModelFrame="BoneRoot"/>
      </PhysicsModel>
      <GameData PackSlotsNumber="1" LoadType="CargoLogsShort" ManualLoadSpawnOffsetY="2">
            <LoadPoint Pos="(1.365; 0; 0.000)" />
            <LoadPoint Pos="(-1.365; 0; 0.000)" />
      </GameData>
      <Landmark Mesh="landmarks/cargo_unit_bricks_lmk" MinScale="1.8" MaxScale="3.8"/>
</ModelBrand>
```

7.2. Modifying XML Class of the Cargo Addon

In the XML class of the Cargo Addon, in addition to the regular fields of the Cargo Addon, you need to specify the following additional attributes of the `<InstallSlot>` tag:

- `CargoAddonSubtype` – the name of the Cargo Addon SubType. The value of this field must have the same value as the `Subtype` attribute of the `<LoadArea>` tag of the XML class of the particular truck/frame addon/trailer (to link the Cargo Addon to this truck/frame addon/trailer).

- `ManualLoads` – the number of correctly placed Cargo Units required for transformation into this Cargo Addon (after packing). Please note that currently this value is fixed and must be equal to **3**.

Along with that, you can specify the position of the Cargo Units (separate logs) that are spawned instead of a Cargo Addon (pack of logs) when you unpack the Cargo Addon. This position can be specified in the `Position` attribute of the `<SpawnLoadOrigin>` tag. The value needs to be specified in the coordinates of the Cargo Addon.

Sample file: **cargo_logs_frame_addon_log_short.xml**

```
<TruckAddon>
      <PhysicsModel Mesh="trucks/cargo/cargo_logs_frame_addon_log_short">
            <NetSync Legacy="false" />
            <Body
                  CenterOfMassOffset="(0; -0.3; 0)"
                  Friction="0.1"
                  ImpactType="Foliage"
                  Mass="2000"
                  ModelFrame="BoneRoot_cdt"
            >
                  <Constraint Type="Rigid" />
                  <Body ImpactType="Truck" Mass="2000" ModelFrame="BonePrismaticShift_cdt">
                        <Constraint ExplicitParentFrame="0" Type="Fixed" />
                  </Body>
            </Body>
      </PhysicsModel>
```

```
    <GameData>
        <Sounds
            InstallSound="cargo_manual/cargo_manual_logs_medium"
            TransportedSound=""
            ManualSound="cargo_manual/cargo_manual_logs_medium"
        />
        <SpawnLoadOrigin Position="(-3; 1; 0.000)" />
        <InstallSlot CargoLength="1" CargoType="CargoLogsShort" CargoAddonSubtype="FrameAddonLog"
ManualLoads="3" />
        </GameData>
</TruckAddon>
```

7.3. Modifying XML Class of the Truck / Frame Addon / Trailer

In the XML class of the particular truck/frame addon/trailer, you need to specify the following:

- In the `<LoadArea>` tag within `<GameData>`, you need to specify the following attributes:
  - `Subtype` – the name of the Cargo Addon SubType. The value of this field must have the same value as the `CargoAddonSubtype` attribute in the `<InstallSlot>` tag of the XML class of the Cargo Addon (to link the truck/frame addon/trailer to this Cargo Addon).
  - `TrailerLoad` – the optional Boolean field that defines whether the player can load cargo to this entity. For example, a trailer for loading logs can consist of two parts: one addon at the end of the truck and one trailer with the opened front part. To avoid loading logs to both of these parts (and duplicating cargo), one of these parts should have `TrailerLoad="true"` and the other one should have `TrailerLoad="false"`.
  - `Type` – the Cargo Type of this cargo, the same value as in the XML class of the Cargo Addon and in the XML class of the Cargo Unit. For details, see the sections on regular cargo (2. and 5.) above.
  - `Min` and `Max` – position of two points on the edges of the cube that define the area (volume) of the Load Area.
  - `ParentFrame` – the bone (frame) to which this Load Area is attached.

- Since logging is still based on the Slots functionality, you need to use the `<AddonSlots>` tag within `<GameData>` too. For details, see 2.1. Slots: Loading and Placement of Cargo above.
  However, frequently, only 1 slot is used in the case of logging (as in the sample below).

WARNING: If multiple slots are used, the number of the slots must exactly correspond to the number of bones with `ExplicitParentFrame` in the physical model of Cargo Addon. If these numbers do not match, this can result in issues during packing and unpacking.

Sample file: **frame_addon_log_short.xml**

```
<_templates Include="trucks" />
<TruckAddon IsChassisFullOcclusion="true">
      <PhysicsModel Mesh="trucks/addons/frame_addon_log_short">
            <Body ImpactType="Truck" Mass="700">
                  <Constraint Type="Rigid" />
            </Body>
      </PhysicsModel>
      <ModelAttachments/>
      <GameData
            CameraPreset="addon_1"
            Category="frame_addons"
            Price="6000"
            UnlockByExploration="false"
            UnlockByRank="1"
      >
            <UiDesc
                  UiDesc="UI_ADDON_FRAME_LOG_SHORT_ADDON_DESC"
                  UiIcon30x30=""
                  UiIcon40x40=""
                  UiName="UI_ADDON_FRAME_LOG_SHORT_ADDON_NAME"
            />
            <LoadArea

                  Subtype="FrameAddonLog"
                  TrailerLoad="true"
                  Type="CargoLogsShort"
                  Min="(-3.038; 0.182; -1.198)"
                  Max="(-0.178; 2.145; 1.198)"
```

```
			ParentFrame="BoneRoot_cdt"
		/>
		<LoadArea
			Subtype="FrameAddonLog"
			TrailerLoad="true"
			Type="CargoLogsShort"
			Min="(-5.897; 0.182; -1.198)"
			Max="(-3.038; 2.145; 1.198)"
			ParentFrame="BoneRoot_cdt"
		/>
		<InstallSocket Offset="(-2.351; 0; 0)" Type="FrameAddonLogShort" ParentFrame="BoneRoot_cdt" />
		<AddonSlots
			InitialOffset="(0; 0; 0)"
			ParentFrames="BoneRoot_cdt"
			Quantity="1"
		/>
	</GameData>
</TruckAddon>
```

7.4. Update of Long Logs Mechanics in DLC 6 (“Haul & Hustle”)

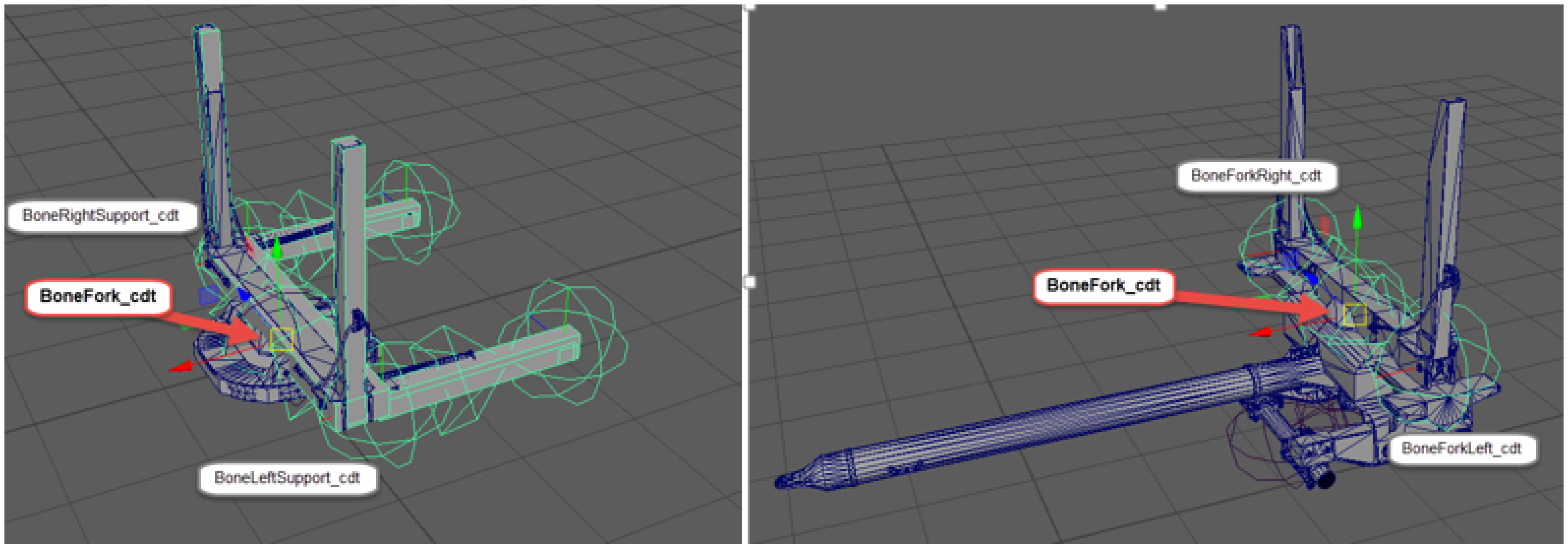
NOTE: Long Logs are in some way a unique type of cargo since they are loaded *not* into a single truck addon/trailer (like Short Logs or Medium Logs) but *both* into a truck addon and a trailer at the same time. That’s why the mechanics behind them are more complicated.

The DLC 6 (Season 6, "Haul & Hustle") brings some changes in the internal mechanics of Long Logs and their setup.

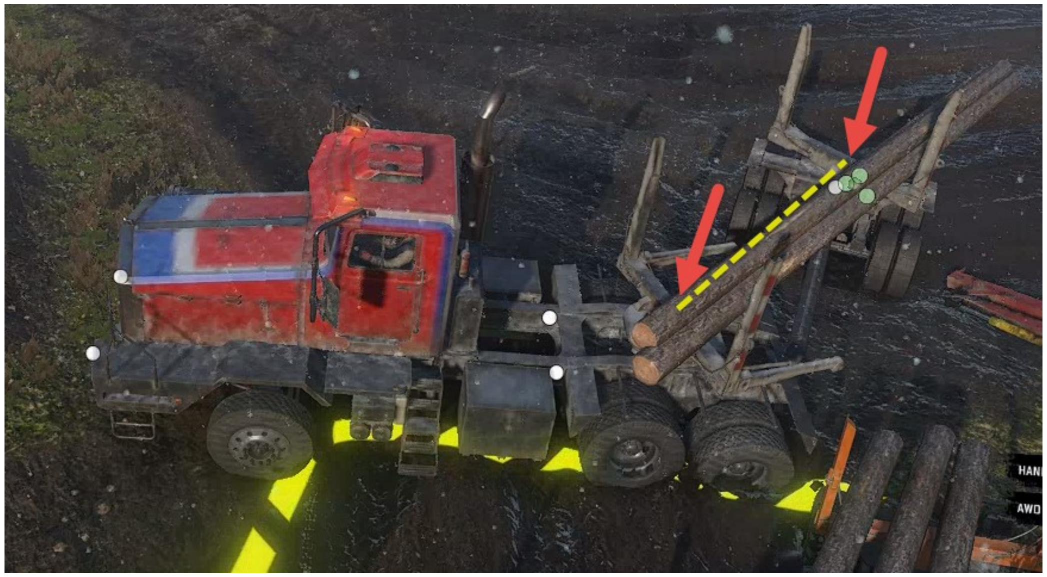
Before this update, when the long logs were packed into a Cargo Addon, the game was taking into account only the orientation of the fork of the log addon and was not taking into account the orientation of the fork of the trailer. This caused issues when the trailer was located at a large angle to the truck (packed logs were spawned not within the fork of the trailer, they were falling to the ground, and so on).

After the update, the game will consider the orientation of both forks and modify them, which will eliminate these issues. The particular mechanics behind this can be illustrated as follows:

1. The game will determine the central bones of the fork of the addon and the fork of the trailer (see below).

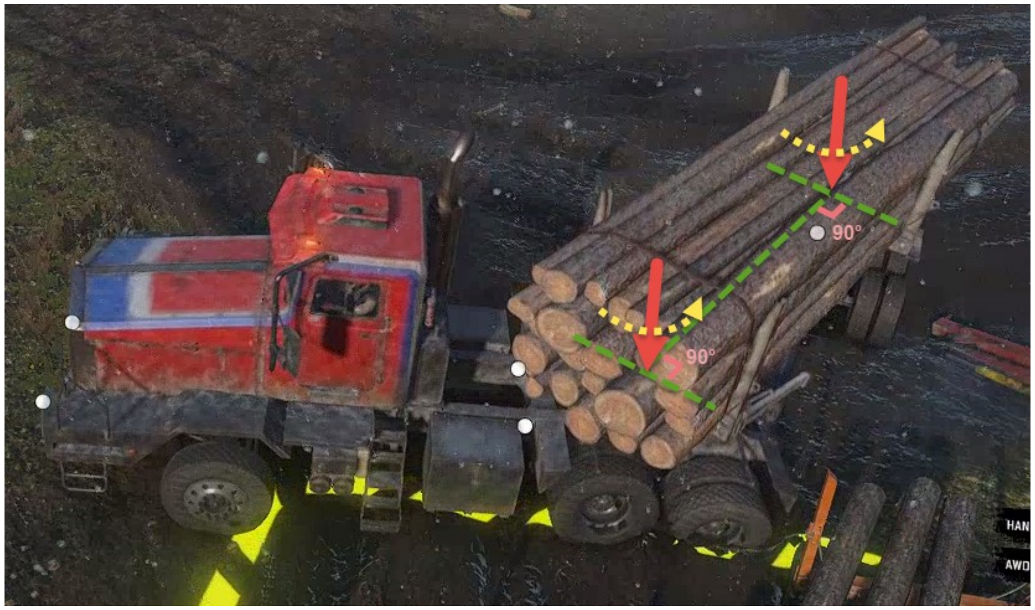


2. The game will draw an invisible line (vector) between these points of the forks.



3. At the moment of packing, the game will rotate the fork of the addon and the fork of the trailer to put them in the position that is perpendicular to this invisible line from step #2. As a result, the packed long logs will be located within both forks and this

cargo addon will be parallel to the invisible line between forks.



However, for the game to be able to do all of that correctly, the setup of the long logs in the game must be updated. And, if you have custom long logs or custom long log trailers, you should update them too (otherwise, they will cause issues).

Particularly, you will need to update the XML classes of the following entities:

- **Cargo Addon**
- **Addon**
- **Trailer**

As the reference files for these changes, you can use:

- *Cargo Addon*: **cargo_logs_long_trailer_log_pole.xml**, located at **[media]_dlc\dlc_3\classes\trucks\cargo\** in the **initial.pak** archive.
- *Addon*: **bunk_log_addon.xml**, located at **[media]_dlc\dlc_3\classes\trucks\addons\** in the same archive.
- *Trailer*: **trailer_log_pole.xml**, located at **[media]_dlc\dlc_3\classes\trucks\trailers\** in the same archive.

For details on the changes that need to be done, please refer to the subsections below.

7.4.1. Long Logs: New Fields in the Cargo Addon

If you open the **cargo_logs_long_trailer_log_pole.xml**, located at **[media]_dlc\dlc_3\classes\trucks\cargo\** in the **initial.pak** archive, you will see the following new tag and attributes within the `<GameData>` tag:

```
<GameData ... RecreateOnZoneChange="True">
      ...
      <LongLogsAlignData
            TargetOffset="(0.0, 0.0, 0.0)"
            CargoPivotBody="BoneRoot_cdt"
            CargoBodiesToAlign="BoneBottom_cdt,BoneTop_cdt,BoneRoot_cdt"
            TruckAddonBodiesToAlign="BoneFork_cdt"
            TrailerBodiesToAlign="BoneFork_cdt"
      />
</GameData>
```

This new `<LongLogsAlignData>` tag will allow the system to load the long logs (or a similar cargo addon) to the trailer located at an angle to the truck, without the necessity for the player to align the trailer. Please note that it works only in the combination with the new fields that appeared in the addon of the truck and the trailer.

You need to specify the following attributes in the `<LongLogsAlignData>` tag:

- `CargoPivotBody` – Typically, the long logs (as a model) consist of multiple bones, which makes their movement within a truck more natural. As the value of `CargoPivotBody`, you need to specify the name of the root bone of your long logs Cargo Addon.
- `CargoBodiesToAlign` – In this field, you need to specify all bones of your long logs Cargo Addon, in random order. This is necessary for the system to be able to align them.
- `TruckAddonBodiesToAlign` – The name of the central bone of the fork *of the truck addon*. This value will help the system to identify the center of the fork of the truck addon and be able to rotate it. As you can see in the illustration at 7.4 above, this bone is named `"BoneFork_cdt"` in our setup of the *truck addon*. If you look into `<PhysicsModel>` containing this bone (in the XML class of the truck addon – **bunk_log_addon.xml**), you will see that this bone can be rotated (it is attached to the parent bone with the constraint of the `"Hinge"` type).
- `TrailerBodiesToAlign` – The name of the central bone of the fork *of the trailer*. This value will help the system to identify the center of the fork of the trailer and be able to rotate it. As you can see in the illustration at 7.4 above, this bone is also named `"BoneFork_cdt"` in our setup of the *trailer*. If you look into `<PhysicsModel>` containing this bone (in the XML class of the trailer – **trailer_log_pole.xml**), you will see that this bone can also be rotated (it is also attached to the parent bone with the constraint of the `"Hinge"` type).

NOTE: This particular long logs Cargo Addon may be used *only* in a combination with a particular truck addon and particular trailer. That's why we can specify the information on the bones of the truck addon and trailer in the Cargo Addon class itself.

Other attributes that were added (`RecreateOnZoneChange`, `TargetOffset`) are currently not used and will be deleted.

7.4.2. Long Logs: New Fields in the Addon

If you open **bunk_log_addon.xml**, located at **[media]_dlc\dlc_3\classes\trucks\addons\** in the **initial.pak** archive, you will see the `UseTrailerFrame` attribute added to the `<AddonSlots>` tag:

```
<GameData ...>
      ...
      <AddonSlots
            InitialOffset="(0; 0; 0)"
            OffsetStep="(-4.0; 0; 0)"
            ParentFrames="BoneFront_cdt"
            UseTrailerFrame="true"
            Quantity="1"
      />
</GameData>
```

As we stated before, the mechanics of logs is based on the functionality of regular **Slots**, just like any other cargo (see 2.1. Slots: Loading and Placement of Cargo for details on slots). In addition, this mechanics uses **Load Areas** and **Load Points** on the Cargo Unit (see 7. Custom Logs as Cargo. CargoAddonSubtype and its subsections for details).

Thus, for your logs, you need to set up slots correctly. To orientate these slots correctly (taking into account the trailer's position) when the long logs trailer is attached to the truck, we added the `UseTrailerFrame` attribute. When this attribute is equal to `"true"` and the trailer is attached to the truck, then the system will use the values specified in the `InitialOffset` and `OffsetStep` attributes **from the `<TractorCargoSlotsOverride>` tag** (specified in the XML class of the *trailer*) instead of the initial `InitialOffset` and `OffsetStep` values of the `<AddonSlots>` tag.

NOTE #1: The values of the `InitialOffset` and `OffsetStep` attributes are used as was described above (see 2.1. Slots: Loading and Placement of Cargo).

NOTE #2: When the overrides of these attributes from the `<TractorCargoSlotsOverride>` tag are applied, this switches the slots from being configured relative to the truck addon to being configured relative to the trailer. In other words, the initial values of these attributes (in the `<AddonSlots>` tag) are configured in the coordinate system of the truck addon, but their *overrides* in the `<TractorCargoSlotsOverride>` tag need to be specified in *the local coordinate system of the trailer.*

7.4.3. Long Logs: New Fields in the Trailer

If you open **trailer_log_pole.xml**, located at **[media]_dlc\dlc_3\classes\trucks\trailers\** in the **initial.pak** archive, you will see two new tags (`<TractorCargoSlotsOverride>` and `<LongLogsAlignTarget>`) within the `<GameData>` tag:

```
<GameData FrameAlign="" ...>
      <TractorCargoSlotsOverride InitialOffset="(0.0, 2.0, 0.0)" OffsetStep="(-5.0; 0; 0)" />

      ...

      <LongLogsAlignTarget Pos="(0.0, 1.050, 0.0)"/>
</GameData>
```

The `InitialOffset` and `OffsetStep` attributes of the `<TractorCargoSlotsOverride>` tag allow you to specify overrides for the corresponding attributes of `<AddonSlots>` tag (of the truck addon). These overrides will be used when this long logs trailer is attached to the truck, see 7.4.2 above for details.

> **NOTE**: The overrides of the `InitialOffset` and `OffsetStep` attributes in the `<TractorCargoSlotsOverride>` tag work with the coordinates of slot(s) in *the local coordinate system of the trailer.* Please note that this differs for the *initial* values of these attributes in the `<AddonSlots>` tag (these initial values are specified in the coordinate system of the truck addon).

The value of the `Pos` attribute of the `<LongLogsAlignTarget>` tag is very important for the cases when the trailer is located at an angle to the truck. This value should be set in such a way that the `Pos` point will be located exactly in the middle of the fork of the trailer. If this value is set incorrectly, then the larger is the angle between the trailer and the truck, the more probable is the case that packed logs will intersect the fork. However, if this value is set correctly, then the logs will be correctly packed even when the angle between the truck and the trailer is close to 90°.

> **NOTE #1**: Typically, the `Pos` value must be specified in the local coordinate system of the trailer. We have added a specific debugging attribute to help you identify it, see Debugging, Logging, and Identifying the "Pos" Value More Easily below.
>
> **NOTE #2**: However, there are some special cases, when the `Pos` value is specified relative to the particular bone of the trailer *that corresponds to its orientation*. See Special Case: the "ParentFrame" and "FrameAlign" Attributes below for details.

Debugging, Logging, and Identifying the "Pos" Value More Easily

To allow you to identify the `Pos` value more easily, we have added one more specific attribute of the `<LongLogsAlignTarget>` tag – the `TestFrame` attribute. It allows you to identify the position of the particular bone. So, you can specify the bone that is in the middle of the fork of the trailer there, and, by that, identify the `Pos` value.

The usage of this attribute is very simple: you specify the name of the target bone of the trailer in it.

For example:

```
<LongLogsAlignTarget Pos="(-10, 1.050, 0.0)" TestFrame="BoneLeftFork2_cdt"/>
```

The `TestFrame` attribute is used for debugging only. Particularly, when you specify the name of the particular bone in it and perform packing of the logs in the game, two new entries are added to the log of the game.

> **NOTE**: The log of the game is located in the **...\Documents\My Games\SnowRunner\base\logs\log.txt** file (e.g. "C:\Users\<*username*>\Documents\My Games\SnowRunner\base\logs\log.txt").

For example:

```
Game| posTestLocal -8.124664;0.547661;1.180298
Game| posCargoRootLocal 2.200737;0.781914;-0.017334
```

Where:

- posTestLocal entry – shows the coordinates of the bone that is set as `TestFrame`. Thus, by specifying the bone in the center of the fork in this field (or the bone that is close to it) you can identify the correct value of the `Pos` attribute (or the value that is close to it).
- posCargoRootLocal entry – shows the coordinates of the center of the fork of the truck addon.

NOTE #1: For best results, Y coordinates of `Pos` (identified from posTestLocal) and posCargoRootLocal should be the same (if the truck is standing on the flat surface at this moment).

NOTE #2: This simple debugging method works with the `ParentFrame` attribute too (see below). I.e., you can specify:

```
<LongLogsAlignTarget Pos="(-10, 1.050, 0.0)" ParentFrame="Bone10_cdt" TestFrame="BoneLeftFork2_cdt"/>
```

and the logged coordinates of the `"BoneLeftFork2_cdt"` bone (posTestLocal) will be relative the `"Bone10_cdt"` bone.

Special Case: the "ParentFrame" and "FrameAlign" Attributes

> **WARNING**: Please use this section only if you know what you are doing.

However, there are nuances in specifying the `Pos` value for some special cases. Particularly, we know that some of your log trailers may be rather unique too. They can have a rather complex bone structure inside. And, in some rare cases, using the regular local coordinate system of the trailer as the origin will not work correctly to align the slot(s) along the trailer.

So, in this case, you need to specify the `Pos` value *not* in the coordinate system of the trailer but relative to the particular bone *that corresponds to the orientation of the trailer*. This bone can be identified by the system if you specify its name in the specific hidden

`ParentFrame` attribute of the `<LongLogsAlignTarget>` tag. Please note that this approach and the `ParentFrame` attribute are unnecessary in most of the regular cases.

> **NOTE**: Typically, the `ParentFrame` attribute is necessary in rare cases when there is a complex hierarchy of bones of the trailer and there is a particular bone that corresponds to the orientation of the trailer (i.e. it is *not* the root bone of the trailer). In this case, this bone can be set as a `ParentFrame` to achieve the correct behavior of the system.

For example, if the bone named `"Bone10_cdt"` corresponds to the *the orientation of the trailer*, you can specify `Pos` relative to it:

```
<LongLogsAlignTarget Pos="(-10, 1.050, 0.0)" ParentFrame="Bone10_cdt"/>
```

Moreover, in this case (if you are using the `ParentFrame` approach), you will also need to specify the `FrameAlign` attribute of the `<GameData>` tag. In this attribute, you will need to specify the name of the same bone that you have put in `ParentFrame`.

```
<GameData FrameAlign="Bone10_cdt" ...>
```

This value of the `FrameAlign` attribute is necessary for the correct operation of slots (in the case of the `ParentFrame` approach only). Particularly, it tells the system to work with the coordinates of the slot(s) *not* in the coordinate system of the trailer but, as with `Pos`, relative to the particular bone that is specified in `FrameAlign`.

> **WARNING**: This `ParentFrame` approach (if used in the wrong case) can lead to issues. In most cases, it is not necessary.

7.5. Logs as Cargo: Note On Slots

As was stated above (in 7. and 7.3), the loading of logs is based on the functionality of regular slots (of the truck, frame addon, or trailer). For the description of **Slots** mechanics, please see 2.1. Slots: Loading and Placement of Cargo.

And, the setup of the Slot(s) is still important when you configure logs as cargo (along with the setup of Load Area, Load Points, and so on). If the slot is configured incorrectly (e.g. is incorrectly located), this may result in issues during loading and packing.

To ensure that the slot(s) created for logs will work for them, you may use the following method:

You can create slot(s) as if you are creating a regular custom cargo (e.g. a container, *not* logs). Then, in the most simple way, create the regular custom cargo for this slot and make its size the same as the size of your logs (in their Cargo Addon form). Then, check how this cargo is installed to the configured slot(s) during the loading of the truck. If this cargo is loaded correctly, then, with high chances, the logs will be loaded to this slot correctly too. If there are issues with loading a simple regular cargo to this slot, then the logs loading will not work correctly too.

> **NOTE**: In the case of the long logs, do not forget to configure the overrides of the `InitialOffset` and `OffsetStep` attributes in the `<TractorCargoSlotsOverride>` tag (in the XML class of the trailer). These overrides will be used when this long logs trailer is attached to the truck. See 7.4.2 above for details.

8. Custom Cargo in the Editor

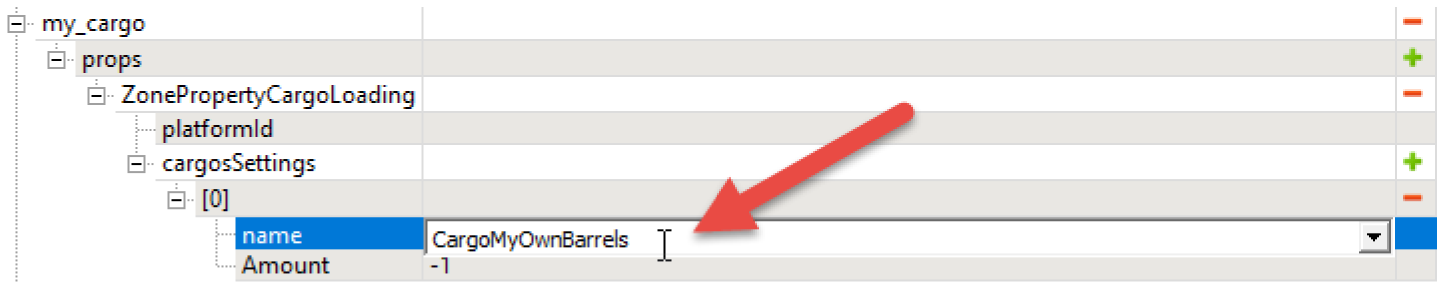
8.1. Selecting Custom Cargo for the Loading Zone

In the current version of the game, you will *not* find your freshly created custom Cargo Types in the typical drop-down lists of cargo types.

E.g., if you browse the list of cargo types within the **ZonePropertyCargoLoading** zone (see "*5.15.1.1. Loading Cargo: ...CargoLoading and ...ManualLoading zones*" in the **SnowRunner™ Editor Guide**), you will not find them there.

But, you can still add your new custom Cargo Type to the properties of your Loading zone, by doing the following:

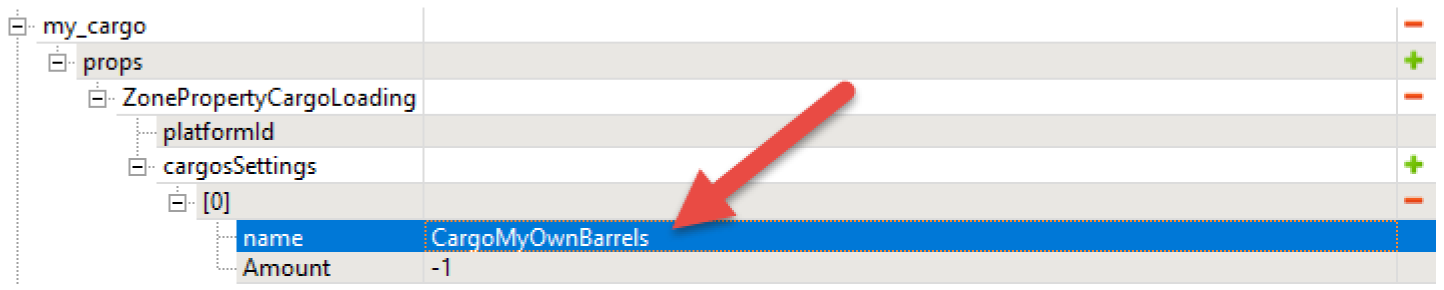
1. Create the **ZonePropertyCargoLoading** zone, in a regular way.
2. In the Zone Settings plugin, expand its properties.
3. In the **cargosSettings** list, add a new entry.
4. Manually specify one of the new cargo types listed above as the cargo type for this entry:
    a. In the list entry, double-click the value next to the **name** field.
    b. Do not use the drop-down list, but manually delete the selected value in this field and enter the name of your custom Cargo Type instead.



```
my_cargo
  props
    ZonePropertyCargoLoading
      platformId
      cargosSettings
        [0]
          name      CargoMyOwnBarrels
          Amount    -1
```

    c. Press ENTER to save the value. The system will remember it and it will be displayed as the value of this field.

    NOTE: If you do not press ENTER (e.g. if you simply click outside the field), the entered value will ***not*** be saved and will be changed back to value from the drop-down list.



```
my_cargo
  props
    ZonePropertyCargoLoading
      platformId
      cargosSettings
        [0]
          name      CargoMyOwnBarrels
          Amount    -1
```

5. Save your zone settings.

Using this technique, you can create the Loading zone with your custom cargo.

Similarly, your custom cargo type can be added to all other drop-down lists of cargo types.

8.2. Adding Cargo Items to the Level

Previously, models of custom Cargo Units were displayed in the SnowRunner™ Editor (in the **Select Asset** window, along with all other models that are displayed when you are adding a model to the level). They were displayed there with the "**\Media**" suffixes (to show that they are custom models with source files in the Media folder).

However, now, they are intentionally not displayed there. In fact, all dynamic models of cargo units that previously were displayed there with the "**cargo_unit**" prefixes are now removed from there. These models were mixed with the "legitimate" models of cargo units that the player can receive with the help of zones and objectives (see below), which resulted in various issues. So, they were removed from there.

> **TIP**: However, if you still need your model of a cargo unit for decorative purposes, you can add it to the level as a regular custom model, which will not be identified as cargo by the game. The process of creating a regular custom model is even more simple than the process of creating a custom cargo. For details, see the "**Custom Level Entities. Models, Overlays"** guide for details, available as **Custom_Level_Entities_Models_Overlays.pdf** of the same documentation package).

So, now, there are only the following "legitimate" ways to add a custom cargo item to the level:

- you can use the automatic or manual Loading zones. For details, see 8.1 above.
- or, you can use the **spawnCargoOnStageActive** stage of an objective (in a similar way). For details, see "*5.16.1.7. Spawning Cargo on a Map …*" section in the **SnowRunner™ Editor Guide**.