Star Trek: Armada Model Hierarchy

Summary

Star Trek: Armada utilizes SOD files, that basically contain each units' structural definition, the mesh, as well as references to the textures to be applied to the faces defined by it.

Along with that come also the so-called node definitions. Each node has a specific role, like functioning as a point of origin from which weapons fire or from where damage-flares are emitted. Every node has a vertex position and some types also have a direction in which they point (e.g. crew damage indicator sprites).

The totality of the nodes is part of the node hierarchy of the model. Each node is tied to a parent node, except for the *root* node, which stands for itself. While the textures on the faces of the mesh are clearly visible, nodes serve as location points of other elements, so are by themselves not visible. The names of nodes are in part fixed, in part depend on your models mesh groups or the number of hardpoints you give them.

Node names prepended by *e_*, *s_*, *m_* or *hp* are reserved nodes. Some names are very specific, serving a unique purpose: *borg*, *crew*, *damage*, *engines*, *geometry*, *hardpoints*, *life*, *lights*, *root*, *sensors*, *shield* and *target*. They must not be used for anything else but their standard function.

A Basic Hierarchy

Here is an example structure of nodes derived from the Cube unit:

```
root
 damage
  borg
  crew
   s_crew1
  engines
   e_plasmalrg
  life
   e_steamlrg
  sensors
   s_sensor
  shield
  target
 hardpoints
  hp01
 geometry
  m_bbattleglow1
  Lod0
   m_Bbattle_2
  Lod1
```

```
   m_Bbattle_1
  Lod2
   m_Bbattle
```

The indentation represents the hierarchical structure. There are different types of nodes, some of which are mandatory.

Node Types

Borg Node

Nodes of this type are simply named *borg*. *borg* nodes are serving the task of parts of the model to be displayed, once it has been built or assimilated by a Borg faction. For all other factions the referenced model is set invisible. The *borg* node is child node of *damage* and it is optional. By itself the *borg* node and its child elements do not have any direction. The direction of the child elements is implicitly clear by the mesh models alignment as part of the entire model. So they don't need any alignment via node direction. A *borg* node without any child nodes will have no visible effect. So you can simply omit it, of you do not intend to give your model any borg indicators. (E.g. when meant for the Borg faction to begin with, the unit most likely will be recognizable as Borg anyway.)

The child elements of the *borg* node have to have the name(s) of the mesh group(s), that are only displayed for as long as the occupying crew is of faction Borg. These child nodes may also have an additional child node each, that has the name of another mesh group making up the borg glow (green, **not** the team color). These also only appear, if the currently occupying crew is of faction Borg.

Crew Nodes

The *crew* node is child element of *damage* and the parent of further nodes that are displayed when the unit suffers crew losses. It is optional (e.g. crew-less units won't need it). Usually it contains at least one child node in form of sprite nodes taken from the *damage.spr* file. See also Damage Sprite Node Names. By itself it does not have any direction. But its child nodes will need to be directed properly, so that sprite textures align with the surface the node is close to.

Note that when adding multiple damage nodes for crew, the order in which they will start to appear is given by the tailing number in the sprite name, after *crew*. The higher the number, the later the sprite is being displayed (indicating more and more amount of crew damage taken). Higher order damage nodes are added as child node of lower level nodes (making the default sprite *crew16* always be the last and *crew1* always be the first in stock game models).

Damage Node

The node *damage* is mandatory and has strictly a grouping function. If is parent element for the nodes *borg*, *crew*, *engines*, *life*, *sensors*, *shields* and *target*. The *damage* node is a child element of *root*. By itself it does not have any direction.

Engine Nodes

The node named *engines* is a child of the *damage* element. Child elements of *engines* are used as damage indicators, when the engines are down. They are optional (e.g. for stationary models it makes little sense to have them).

Geometry Nodes

The node with the name *geometry* is mandatory. It represents the actual unit's optical manifestation. Without it the model will not be visible. It is the child element of the *root* node and the parent of sub-parts of the geometry definition. Some are special in their function, such as the *glow* element. It makes the unit get the team color. LODs on the other hand are meant for representations of different details. See Level of Display on the concept. The general geometry defining node is the name of the mesh (or mesh group) prepended by `m_`.

Hardpoints

Hardpoints are the locations from which weapons fire. They are children of the *hardpoints* element, which in turn is child of *root*. Each hardpoint is named `hp`*xx*, where *xx* is a unique, serialized number of two digits, beginning with `01`.

Life Node

The *life* node is a child element of the *damage* hierarchy. It's children nodes are usually emitters, a special kind of sprite. The are named in the same way as animated SOD models, usually representing flares. See also Steam and Fog Emitters on some commonly used ones for the *life* node. They are indicating the life support system being down. For ships without life support system (e.g. automated stations) it is not needed.

Light Nodes

The *root* hierarchy may also contain the *lights* node. If used it holds elements, that are used for lighting, such as positioning lights. Child nodes of *lights* are named by sprite names that can be found in the file *lights.spr* in the *Sprites* directory of your Armada main directory or in the Weapons Sprite Names article.

LOD Nodes

LOD nodes reference parts of the model by naming convention. *Lod**x*** with ***x*** being a number, defines the level of detail. Each LOD node has a child node, that references the part of the model by name by being named in the same fashion, prepended with an *m_*. In the above example the sub model *Bbattle_2* if referenced by the most basic LOD, with the number 0. LODs are optional.

Root Node

The node named *root* is mandatory. It is parent of the nodes *damage*, *hardpoints*, *geometry* and *lights*. It serves strictly a grouping purpose.

Sensor Node

This node is child of the *damage* node and has its own children.

Shield Nodes

The node *shield* is child of *root*. It represents shield damage. Child elements of it are named by sprites shown when certain levels of damage are reached. The names are prepended by s_.

Target Nodes

The node hierarchy *target* is child of *damage* and contains nodes, that are shown when the weapons are down. They are named by sprites to be shown and a prepended s_.

Stock Game Node Names

The names of nodes used in the stock game can be found in article Node Names.

Level of Display

The further away the view point of the player is from the unit, the less details a unit needs, in order to still look good. To facilitate this concept of dynamically shown models or model parts, *Levels of Display* (LOD). Parts of the mesh are named specifically, to be referenced by the hierarchy on form of LOD nodes.

Particle Emitters

Emitters are usually kind of flares (e.g. the fountain displayed when engines are down). Node names referencing them are prepended with e_.

Hierarchy Creation With Milkshape

Modelling

Milkshape is not particularly well suited when it comes to creating the node **hierarchy**. It does not know of the concept, somewhat. Instead of it, the joint concept usually used for animations, takes its place. This has the implication, that every joint does still have a direction (just like nodes) but the location of one joint is always depending on the location of its parent element. The parent element concept is basically the same as meant for nodes. But positioning a node is not as free as it is usually required. So creating the nodes in *Milkshape* is **not** advised. 3DS Max and Storm3D Tool do a far better job.

Node Name Conventions

Apparently *Milkshape* in conjunction with an SOD exporter requires the normally none-prefixed node names to be prefixed with indicators such as h_. These prefixes indicate what type the node has. The different types are as follows:

Type	Prefix	Description
hardpoints	h_	Some technical elements being part of the hierarchy, not necessarily having an artwork-like function (like applying a texture or giving a direction of something). This may include structuring nodes, like *borg*, but also nodes with special tasks, such as the bee-nodes *bot**x***. Structuring nodes have a location that does not really matter by itself. The location of the child elements on the other hand may very much be relevant, e.g. for *hp* nodes (actual hardpoint nodes). Depending on the situation, some of the child elements may even have a direction, such as hard points (directed pulse weapons will only fire in the general direction the node points to).
sprite	s_	A location **and** direction of a sprite texture being applied (e.g. crew damage indicators).
emitter	e_	A 3D animated object with location **and** direction, usually damage indicators for life support and engines being down.
mesh	m_	An actual 3D object, e.g. Borg modification indicators, that only appear, when a ship is commandeered by a Borg faction. These nodes don't have a direction and their location is arbitrary. The vertizes/faces of the mesh itself define where the mesh will appear and how it is oriented.

So for example the *damage* node will not be named just *damage* but has to be named h_damage in order to work properly. SODs imported into Milkshape will already fit this naming convention. When you have a look at the interior of SOD files exported from *Milkshape*, the names will be set to normal (for our example, still be named *damage*), too. But when creating nodes yourself, you have to prepend the names of nodes with the correct prefixes. Otherwise your model may not work (e.g. not show the meshes used or not exhibit certain behaviors, such as damage indicators).

Here are some examples of what that may look like:

Node Name	Milkshape Node Name
Bconst	*m_Bconst*
borg	*h_borg*
bot1	*h_bot1*
crew	*h_crew*

Node Name	Milkshape Node Name
crew1	*s_crew1*
damage	*h_damage*
engines	*h_engines*
geometry	*h_geometry*
hardpoints	*h_hardpoints*
hp01	*h_hp01*
life	*h_life*
lod1	*h_lod1*
lod2	*h_lod2*
plasmamed	*e_plasmamed*
poly1	*m_poly*
root	*h_root*
sensor	*s_sensor*
sensors	*h_sensors*
shield	*h_shield*
steamlrg	*e_steamlrg*
target	*h_target*

[**Modding**] [Tools] [ODF Files] [ODF Directives] [Class Labels] [Tech Tree Files] [SOD Files] [Buttons] [Wire Frames] [Sprites] [AI Scripts] [Model Hierarchy] [Node Names] [Emitter Names] [Texture Animation Names] [Sprite Names]

[Back to Modding]