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.

Depending on what tool you use, Node names may be prepended by node type prefixes. See Node Name Conventions on an example for Milkshape 3D. 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. Others depend strictly on the contents of your model.

A Basic Hierarchy

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

```
root
 damage
  borg
  crew
   crew1
  engines
   plasmalrg
  life
   steamlrg
  sensors
   sensor
  shield
  target
 hardpoints
  hp01
 geometry
  bbattleglow1
  Lod0
```

```
   Bbattle_2
  Lod1
   Bbattle_1
  Lod2
   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.

Officers Quarters Nodes

For Starbases there is the possibility of an officer upgrade (increasing the supply of a player). The optical representation of such an upgrade are special meshs, that are only displayed once a certain level of upgrades has been reached. The *oq* node is child element of *geometry* and parent element of *oq**x*** nodes. ***x*** here is a serial number, starting with 1 and increasing with the number of elements available (usually up to 6). For each upgrade level another element is displayed. These child nodes are not hierarchical (as the crew damage nodes would be). They are a flat hierarchy below *oq*.

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. It is parent element for the nodes *borg*, *crew*, *engines*, *life*, *sensors*, *shields* and *target*. The *damage* node is a child element of *root*. It does not have any direction and its location has no specific effect by itself.

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). The location or direction of the *engines* node has no relevance by itself, but the location **and** direction of its child nodes does matter.

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 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. Officers quartes nodes (*oq*) are for the officers upgrades of starbases.

The *geometry* node should always house nodes with the names of all mesh parts to be visible either constantly or during animations. This specifically excludes *borg* node elements, that are only visible if the unit is occupied by a Borg faction.

In case of mesh elements being visible only for certain levels of details (lod), the mesh elements must be child nodes of the corresponding lod node (*lod1* to *lod**n***). In case of animation elements, they must be direct child elements to *geometry*.

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 of the mesh group to be shown for that level of detail. In the above example the sub model *Bbattle_2* if referenced by the most basic LOD, with the number 0. LODs are optional. They don't have any location or orientation.

Hardpoint Nodes

The *hardpoints* node is a grouping node with no direction and no effect of its location by itself. It

houses a number of different types of nodes, e.g. actual hardpoints, repair and build as well as pod and bot nodes.

Hardpoints

Actual hardpoints are the locations from which weapons fire. They are children of the *hardpoints* element. Each hardpoint is named hp*xx*, where *xx* is a unique, serialized number of two digits, beginning with `01`. Their location is directly relevant as point of origin (when pulses, torpedoes, phasers or other originating weapons come from, when they are fired, including special weapons), but also their direction may make a difference in case of directed pulse weapons.

Bot

The *bot**x*** nodes are child elements of the *hardpoints* node. They define the location where the worker bees of constructors come from. Their names can be defined by the ODF directive *workerBeeHardpoints*, which takes multiple arguments, one for each unique worker bee. Usually this is *bot1* to *bot6*.

Repair

Child element of the *hardpoints* node can also be the *repair* node. It is the spot units will move to for repairing. It is found in yards. It is a node with location and direction. The y-direction together with its location is the line in 3-dimensional space units will line up at for repair queuing. Its name might be set with ODF directive *repairHardpoint*.

Build

The *build* node almost the same as the *repair* node, except it is used for construction of units, not repairs. It is as such part of yards and starbases. The actual name of this hardpoint can be defined by the ODF directive *buildHardpoint*.

Docking

The *dock* is similar to the *repair* node, but it is used for defining the docking location of freighters. It is as such part of Mining Stations. The actual name of this hardpoint can be defined by the ODF directive *dockingHardpoint*.

Reseach/Pod

Research items are represented by their own objects at research stations. Each item has a so-called pod, which is it's own SOD model. The nodes representing the location and direction at a research station of such a pod are named *pod**xy***, with ***xy*** being a number.

Life Node

The *life* node is a child element of the *damage* hierarchy. It's child 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. While the location and direction of the *life* node has no direct relevance, the location and direction of its child nodes are relevant. They mark the origin of the emitter animation, and the direction the emitter animation will take.

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. The *lights* node itself has no direction and the location does not have an effect by itself. But the child elements locations and directions are both relevant.

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. As such it has no orientation and its location is not relevant by itself.

Sensor Node

The *sensor* node is child node of the *damage* node. It has no orientation and its location is not relevant by itself. It serves as a grouping node for sprite nodes. These are in stock models animated sprites, that look like welding sparks. These sprites of course have an orientation and location.

Shield Nodes

The node *shield* is child of *root*. It represents shield system damage. Child elements of it are referencing sprites shown when the shield generator is failing (red) but not when it is simply deactivated, like in a Cerulean Nebula (yellow). As always, sprite nodes have not only their location but also their direction as relevant information.

Target Nodes

The node hierarchy *target* is child of *damage* and sprite nodes. By itself it has no orientation and its location has no effect by itself. Its child nodes are the names of the sprites shown when the weapons are down.

Node Structure Misconceptions

Westworlds Big Book of Modding is a widely regarded source on how to set up the node hierarchy. A lot of the above information basically comes from it. But some of it is actually not of a technical requirement but more like a best practice. So even the example in section A Basic Hierarchy is essentially only one way of going about a node hierarchy for a ship. Here are some of the things that the BBOM does not contain or set unjustified as a requirement:

- Node Name Conventions: There are actually no requirements to use specific prefixes, such as *h_*. These stem from the importer/exporter used. The nodes come by default without any of these prefixes as can be seen in the stock game SODs as well.
- The *root* node is not necessarily named *root*. It can also bear other names, such as *root_1* or *root1*. Example: stock game file *Bbase.SOD* has no *root* node, although it of course has a node that constitutes as the root of the hierarchy tree. Also, the name itself, even if it is used, does not necessarily name the *root* node, example would be *Kyard.SOD*, where there is a node *root* as the actual root node, as well as another child node with the name *root1*. This suggests, that the naming of the root is basically arbitrary.
- The *geometry* node can also be named differently, e.g. *geometry_1*. Again, example would be the *Bbase.SOD*.
- Same way *lights* can be named *lights_1*, example would be *Rbase.SOD*.
- *lights* does not seem to be necessary at all, as is indicated by *Rsuperbl.SOD*. It does not have a *lights* node at all but the strobe nodes, that usually are placed below *lights*. This suggests, that *lights* as a grouping node is merely a convention but no technical necessity. Light nodes can basically be placed anywhere in the hierarchy.
- Hardpoints do not need to belong to a node specifically named *hardpoints*. Example again is *Bbase.SOD*. Here the hardpoints are named by the usual *hp**xy*** scheme, but belong to the node *Bhardpts*. This includes the *pod* nodes (see *Kresear.SOD*).
- Nodes *build* and *repair* behave the same as other hardpoints, see *Bbase.SOD*.
- *hp**xy*** hardpoints do not need to be numbered consecutively. They may very well start with some other number than *00* or *01* and they may also have gaps in the numbering. Just make sure you do not use hardpoints in ODFs, that do not exist. Example: *Bturret2.SOD*.
- Hardpoints for weapons do not require to be attached to parent node *hardpoints*. You can just as well attach a hardpoint directly to the *root* node. Example: *Bbee.SOD*.
- Weapon hardpoints do not necessarily follow the name pattern *hp**xy***. Example would be *Romega.SOD*, where the tractor beam hardpoint is simply named *tractor*.
- The naming pattern *hp**xy*** is not a hard definition for (weapon) hardpoints specifically at all. In the *Fscout.sod* file there is no *hardpoints* node, but instead the hardpoint nodes are all child elements of a node named *hp20*. So neither the name of the parent node is a hard requirement, nor are *hp* nodes strictly weapon hardpoints.
- Although for many units only the pattern *hp**xy*** is used, with ***xy*** being a two digit number, more digits are indeed allowed, e.g. *hp200*, as seen in *FbaseHQ.SOD*.
- The mesh nodes do not need to be part of a *geometry* node. They can just as well be attached to the root node. Example: *Bbee.SOD*.
- Certain nodes with special roles can be nested, e.g. node *damage* is found as a child of *geometry* in the *Bsuperbl.SOD* file, or *geometry* houses *hardpoints* in the *Byard2.SOD* file.

Node Names for Multiple Occurrences of the Same Name

If you have to add a specific emitter or other node multiple times, the game will need you to specify different names for them, although of course e.g. an emitter like *plasmamed* still has only one name. The solution for this is adding a number at the end of such a node, e.g. like this: `plasmamed_1` and `plasmamed_2`. The game will recognize that both times *plasmamed* is meant. So you can make both of your nacelles each have a plasma leak at the same time.

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.

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*
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*

See Also

- SOD Files
- Node Names

[**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]