SOD Files

What is an SOD?

Star Trek: Armada is based on the Storm3D Engine which was introduced with Battlezone. As such it uses basically the same technology. *Storm3D Object Definition* files, or SODs for short, are Battlezone's/Armada's means of defining three dimensional objects, including their texture/material use, mesh definition and model hierarchy. In essence, they form the models of units, stations and other 3D objects of the game.

Location and Contents of SODs

They are placed in *SOD* folder of the game installation folder. They do not come with the texture files included in themselves. Those are still placed in folder *Textures\RGB* in form for TGA files. But their names are referenced by the SODs. Together with sprites and emitters they form the full model, that can then be used by ODF Files.

The 3D mesh part of the model must be created/modified with some sort of 3D modeling tool, such as Storm3D Tool, Milkshape 3D or 3DS Max. But usually they require some sort of importer and exporter to make SODs usable. The application of textures is also very often done with the same tools, while the creation of such textures is done with graphics programs such as GIMP.

The node hierarchy is also applied by the modeling tools, but depending on which you use, you might need another one to get it working properly. (E.g. *Milkshape 3D* is not particularly well suited for the directing of nodes, but at least you can create and place them.) The sprites and emitters are described somewhere else, but must be made referenced by the hierarchy by name.

Format of SODs

The following is taken from the document created by Steve Williams. You can find this original source in section Web Links.

Data Types Used

SODs are for the most part binary content files. This article describes version 1.8 of the SOD format. To understand how SODs work, you need to know the following data types:

Datatype Name	Format
UINT8	unsigned 8 bit integer
UINT16	unsigned 16 bit integer
UINT32	unsigned 32 bit integer
FLOAT	4 bit floating point number
VECTOR2	two FLOATs, u and v

Datatype Name	Format
VECTOR3	three FLOATs, x, y and z
MATRIX34	four VECTOR3s, Right, Up, Front, Position, note: Matrices must be orthogonal
COLOR	three FLOATs, red, green and blue, each with values between 0.0 and 1.0
TYPE ARRAY(*number*)	An array of *number* values of type *TYPE*
IDENTIFIER	one UINT16, making up the length of the following string, then UINT8 ARRAY(*length of the string*), the actual string. Note: The length of the string is mandatory, but for a string of length 0 the contents following ware of course of length zero (nothing). So the shortest IDENTIFIER you may encounter is of 2 bytes length.

File Structure

An SOD consists of five sections:

1. File header,
2. list of lighting materials
3. node hierarchy definition (recursively, starting with the *root* node).
4. animation channels and
5. animation references.

File Header

Every SOD starts with the ASCII string `Storm3D_SW`, followed by the version of the used format in form of one FLOAT value. So the header always has the length of 14 bytes. For *Star Trek: Armada* the version is 1.8, making the 4 version bytes always be 0x66 0x66 0xE6 0x3D.

Note: There are two stock game files, *Favenger.SOD* and *Ksensor.SOD*, that have a different header. They start out with `StarTrekDB`. It is likely that those two files are from an earlier stage of the Armada project, where the format was still different.

Lighting Materials

The second section of an SOD contains the characteristics of the vertex lighting materials. To understand this, we need some more definitions.

The surface lighting can use Phong reflectivity model or Lambertian reflectance. The model used can be one of the following three:

LIGHTING_MODEL:

Type	Number
Constant	0
Lambert	1
Phong	2

Each surface can use a lighting material, which is defined like this:

LIGHTING_MATERIAL:

Data Type	Meaning
IDENTIFIER	Name of the lighting material
COLOR	Ambient lighting component
COLOR	Diffuse lighting component
COLOR	Specular lighting component (using Phong reflection model]])
FLOAT	Specular power exponent, the »shinyness« of the Phong reflection material
UINT8	LIGHTING_MODEL

The full list of lighting materials consists of the following information:

1. UINT16: The number of materials defined in the SOD.
2. LIGHTING_MATERIAL ARRAY(<*number of materials*>).

Note: Lighting materials are usually not unique to a specific SOD but used by multiple SODs. As this is the case, loading can be expedited considerably by not loading lighting materials anew for each SOD, but looking for already loaded materials from previously loaded SODs. One implication of that is, that the materials are consistent across SODs. One special aspect about this is the file *Material.SOD*, which contains a number of materials being loaded prior to most SODs, defining the most common materials.

Note: While the above definitions work just fine with colors, one thing that is not immediately obvious is how team colors are created. Any material whose name starts with *team_* will be colored with the team color. This means, there can be multiple, different materials, that will exhibit the team color. Applying such a material to a mesh will make it glow in the color of the team.

Nodes

The third section in an SOD are the nodes, which have various implications on the behavior of a unit or station, including things like firing direction, damage indicators, mesh referencing or LODs used. They come in five different types.

NODE_TYPE:

Type	Number
null	0
mesh	1
sprite	3
LOD control	11
emitter	12

A node as such can be of the described type and may need additional information, depending on the type. But the general node definition looks somewhat like this:

NODE

Data Type	Meaning
UINT16	NODE_TYPE

Data Type	Meaning
IDENTIFIER	Name of the node
IDENTIFIER	Name of its parent node. In case of the *root* node, that has no parent, it's an empty string
MATRIX34	Local transform
TYPE_SPECIFIC_DATA<*Type*>	Data specific to nodes of the given type. See below specifications on this data.

Null Node Data

This particular type has no additional data. It is solely used to mark locations and directions, e.g. for hardpoints and to function as grouping points for other child nodes making up the hierarchy.

LOD Control Data

Similar to Null nodes, this node has no data. It serves as a grouping node for child LOD nodes. Depending on distance and size of a mesh used for LODs, the corresponding mesh is used for display or not.

Sprite Node Data

Those come in two flavors, actual sprites and particle emitters.

Actual sprite nodes do not have any actual data by themselves. The IDENTIFIER of the node itself contains the information. They serve as a localization means, including direction for sprites to face at. The actual sprite information comes from the sprite itself.

The emitters on the other hand need an additional IDENTIFIER for emitter sprites:

TYPE_SPECIFIC_DATA<PARTICLE_EMITTER>

Data Type	Meaning
IDENTIFIER	Name of the emitter, as defined by the sprite file, see Emitter Names on stock game emitters.

Polygon Mesh Data

These nodes tie the mesh to the hierarchy. Meshes consist of vertices, making up faces. Each face can have its own lighting properties.

The vertices in the SOD sense consist if two values, one for the actual mesh and one for the tied-to-it texture vertex. The coordinates are not stored directly but in form of the indices in the vertex arrays of mesh vertices and texture vertices. The combination of both makes it possible to map a texture onto a surface made up of faces very specifically.

FACE_VERTEX

Data Type	Meaning
UINT16	Index of a vertex position from the vertex positions array
UINT16	Index of a vertex position from the texture coordinate array

A face is made up by three vertices. As such each face references three vertices:

FACE

Data Type	Meaning
FACE_VERTEX	First vertex position
FACE_VERTEX	Second vertex position
FACE_VERTEX	third vertex position

The lighting of each face is basically a combination of all faces with the same lighting material tied with said material into one so-called lighting group.

VERTEX_LIGHTING_GROUP

Data Type	Meaning
UINT16	Number of faces/triangles
IDENTIFIER	Name of the lighting material, empty string if none
FACE ARRAY	Array of the faces using the same lighting material

A mesh is then made up of vertex positions as well as texture positions and the texture name itself, as well as the material. The number of said vertices and texture positions as well as the number of lighting groups is stored in a mesh information, along with a culling type (what happens if you look at a texture from the wrong side, is it removed when looking from the wrong side, or not).

TYPE_SPECIFIC_DATA<MESH>

Data Type	Meaning
IDENTIFIER	Texture material name, can be an empty string for default. See also section Texture Materials.
IDENTIFIER	Texture (can be an empty string, of none is applied)
UINT16	Number of vertices
UINT16	Number of texture coordinates
UINT16	Number of vertex lighting groups
VECTOR3 ARRAY	vertex positions
VECTOR2 ARRAY	Texture coordinates
VERTEX_LIGHTING_GROUP ARRAY	VERTEX_LIGHTING_GROUP entries, see definition below
UINT8	Cull type (0 = no culling, all faces' textures are drawn. 1 = back face culling, faces seen from the »wrong« side do not show any texture)
UINT16	Always 0

Animation Channels

The fourth section of an SOD contains the animation channels. An animation is basically the model altered in form and texture for each point in time, allowing for a series of meshes to be shown one after another, like in a movie, making up the animation/movement of said model. This gives a model

the means of moving, for example a fan rotating instead of standing still.

Animation channels are basically the animation of the mesh itself. If the textures on the faces are supposed to change as well, those would be the Animation References.

An animation channel consists of the following data:

ANIMATION_CHANNEL

Data Type	Meaning
IDENTIFIER	The node being referenced by the animation channel
UINT16	Number of key frames used by the channel
FLOAT	Duration this channel lasts
UINT16	Always zero (unused)
MATRIX34 ARRAY	Key frame data, the actual animation transformation, which will be distributed over time evenly during the duration this channel takes

The entirety of animations is stored like this:

Data Type	Meaning
UINT16	Number of animation channels following
ANIMATION_CHANNEL ARRAY	The actual animation channel data of each element.

Animation References

The fifth and last section contains the animation references. In contrast to the Animation Channels, animation references are used to animate the textures themselves. The textures used for this are basically sprites. So each step of the animation from a sprite is used during the animation sequence, but as part of a model.

The animation is made up as a list of animation references of this form:

ANIMATION_REFERENCE

Data Type	Meaning
UINT8	Type of the reference, always 4
IDENTIFIER	The node to which this animation belongs.
IDENTIFIER	Sprite animation used for this node.
FLOAT	Time offset (in seconds) used for this animation reference.

The animation references are then stored as a list:

Data Type	Meaning
UINT16	Number of animation references to follow
ANIMATION_REFERENCE ARRAY	The list of all animation references.

Texture Materials

A texture material defines characteristics of a polygon rasterizer used to render the polygons of a given mesh. In case of *Star Trek: Armada*, there is a fixed set (hard coded) of allowed values:

Material	Meaning
default	Standard material
additive	Utilize additive blending
translucent	Semi transparent
alphathreshold	Cut outs in objects can be realized with alpha channels. They have hard edges but are drawn faster.
alpha	Makes use of an alpha channel. In contrast to *alphathreshold* this requires sorting of layers, which has some performance drawbacks, but looks better.
wireframe	Uses only wireframe graphics

Web Links

- Storm3D Object Definition (SOD) File Format on Armada Files
- Storm3D Object Definition (SOD) File Format on DS-Servers
- Storm3D Object Definition (SOD) File Format on GameFront

See Also

- Model Hierarchy

[**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]
