

Blender Tools for GFS Documentation

Pherakki

v0.1

Contents

1	Getting Started	1
1.1	A Note on Expectations	1
2	Import	2
2.1	Import Restrictions	2
2.2	Importing Model (GMD/GFS) Files	3
2.3	Importing Animation (GAP) Files	4
2.4	Blender Settings	5
2.4.1	Previewing Materials	5
2.4.2	Importing Large Models	5
2.4.3	Hiding Unused Bones	6
3	Editing Models and Animations	8
3.1	GFS Models	8
3.1.1	Bones	8
3.1.2	Rest Pose	10
3.1.3	Meshes	10
3.1.4	Materials	14
3.1.5	Textures	19
3.1.6	Cameras	19
3.1.7	Lights	20
3.1.8	Physics	22
3.2	GFS Animations	23
3.2.1	Normal Animations	23
3.2.2	Blend Animations	25
3.2.3	LookAt Animations	26
3.2.4	Animation Packs	26
3.2.5	Tips for Editing Animations in Blender	27
3.3	GFS Properties	30
3.4	EPLs	31
4	Export	32
4.1	Exporting Model (GMD/GFS) Files	32
4.2	Exporting Animation (GAP) Files	32
5	BlenderToolsForGFS as a Python Library	34
5.1	Python Library Usage	34
5.1.1	GFS/GMD/GAP	34
5.1.2	EPL	37
5.2	Blender Script Usage	37
6	Future Development	39
6.1	Blender Interface	39
6.2	Python API	40

1 Getting Started

BlenderToolsForGFS is a **Blender 2.81+** plugin. It should work on all versions of Blender including and beyond 2.81 that are compatible with the 2.81 API. The plugin has been developed on versions 2.83 and 3.4.1. You should install and remove it like any other Blender plugin.

1.1 A Note on Expectations

This plugin should be viewed as a **supplement** to existing model-editing tools, and not as a replacement. You should, in particular, post-process any models exported from Blender in **GFD Studio**, following existing tutorials and knowledge to achieve your goals.

Please also note that although this document does offer a number of guides on bits of basic Blender usage required to use the plugin, this is not a guide on how to use Blender. If you are intending to work with models and animations, it is ultimately your responsibility to learn how to use Blender using the wealth of resources available on the internet.

This is also not a guide on how to mod games. Again, it is your responsibility to seek out or discover the information you require to attain any such goals.

What this guide *is* intended for is to assist you in successfully exporting data from Blender to the GFS file format. Suggestions for improvements on how to do that are very much welcomed, as long as they do not fall out-of-scope for the essential use of the plugin.

2 Import

2.1 Import Restrictions

Currently, a subset of features of the GFS format are not supported by the plugin and cannot be exported. These are:

- File versions outside of the range 0x01104920 - 0x01105100

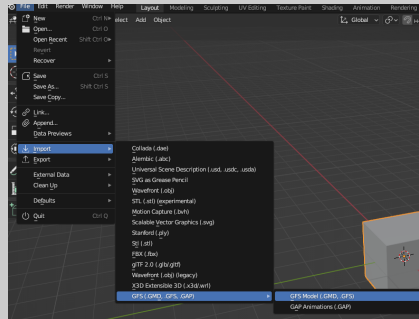
There are also some additional considerations to bear in mind.

- During import, meshes are re-parented from their positioning node to the root node, whilst preserving their world transform. This is because there does not seem to be a straightforward way to make a mesh follow the transform of a bone whilst also being deformed by an armature without Blender double-counting transformations. These positioning nodes are also removed from the import if they are not used for any other purpose.
- Animations of the Root Node and of mesh positioning nodes are not editable from Blender. Any mesh positioning node animations **will be messed up** by the Blender import if these nodes were not originally children of the root node. This must be fixed outside of Blender by re-parenting and repositioning these nodes.
- Only non-BC7 DDS texture slots are currently importable in models.
- There is a lot of extra data, such as EPL effects and physics, that is not explicitly expressed as Blender data but is still attached to the model for export. This data is non-interactable from Blender in the current version. If some chunk of data is not mentioned in the documentation, you can assume that it is probably stored in this fashion.

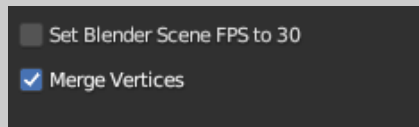
2.2 Importing Model (GMD/GFS) Files

Step-by-step: Accessing the Model Import Menu

1. Open the File menu and navigate to the **Import > GFS > GFS Model** submenu item.



2. From the import options on the right, choose whether to set the Blender render speed to that which is appropriate for GFS models and whether to merge model vertices such that the mesh is smooth (recommended).

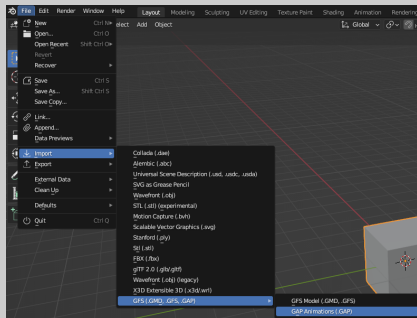


3. Select the GMD or GFS file to import using the file browser.

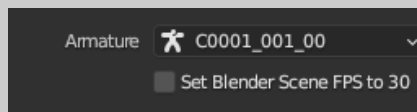
2.3 Importing Animation (GAP) Files

Step-by-step: Accessing the Animation Import Menu

1. Ensure that you have first imported the model that the animation is modelled for.
2. Open the File menu and navigate to the **Import > GFS > GAP Animation** submenu item.



3. Select the armature for your imported model using the drop-down. Also choose whether to set the Blender render speed to that which is appropriate for GFS models.



4. Select the GAP file to import using the file browser.

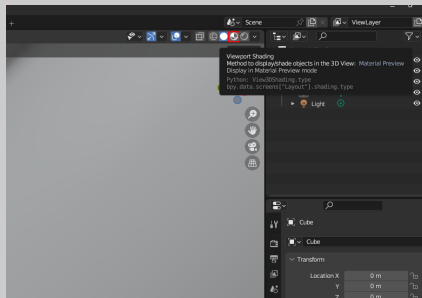
2.4 Blender Settings

2.4.1 Previewing Materials

Many beginners to Blender are confused by the fact that models do not display their materials when it is first opened. Blender by default renders models in a “Solid” representation that is useful for modellers editing meshes. You can preview materials instead by changing this render setting.

Step-by-step: Previewing Materials

1. Locate the **Viewport Shading** widget and select **Material Preview** mode.

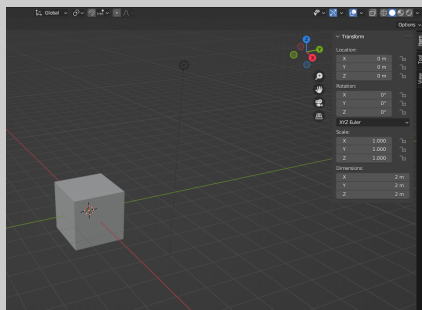


2.4.2 Importing Large Models

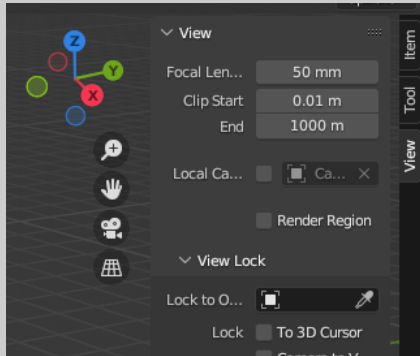
Some models, such as Field Models, are typically in such large units that they exceed the default render distance of Blender. In this instance, you may find it useful to increase Blender’s render distance.

Step-by-step: Increasing Blender’s Render Distance

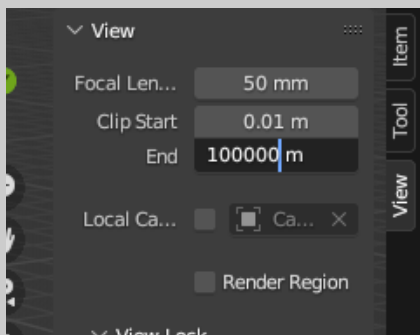
1. Press the **N** key to open the **Sidebar**.



2. Click the **View** tab in the **Sidebar**.



3. Change the value in the **End** box to a value large enough for you to comfortably navigate the model.



4. Press the **N** key to close the **Sidebar**.

2.4.3 Hiding Unused Bones

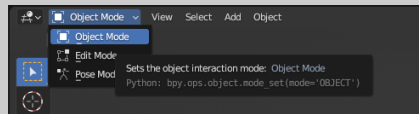
On import, three bone layers will be created in the armature:

- All Bones
- Bones used in Vertex Groups
- Bones not used in Vertex Groups

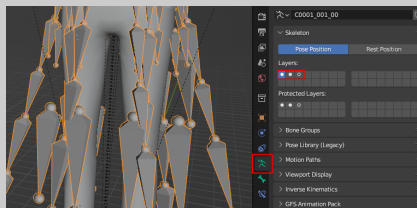
Selecting one of these layers will allow you to hide these subsets of bones. Note that these groups are created **by the plugin** on import, and if you add new bones to a model then it is up to you to add bones to whatever bone layers you want.

Step-by-step: Selecting Bone Layers

1. Switch to Object Mode.



2. Select the model armature.
3. Click the Armature icon in the Properties Panel. Select the Bone Layer that is active in the Viewport.



3 Editing Models and Animations

3.1 GFS Models

Models are imported as Armature objects with meshes, cameras, and lights parented below them. Most of the objects imported from GFS files can carry additional information beyond pure geometry and shading data, which is outlined in the proceeding sections.

GFS models are very large compared to typical Blender scales. If parts of the model disappear as you zoom out, you will need to change the render distance of Blender as described in section 2.4.2.

The data in the files are assumed to be oriented Y-up and with X-axis-oriented bones. During import, these are converted to the Z-up orientation and Y-axis-oriented bones to match the Blender conventions. On export, the Blender data is converted back to Y-up orientation and X-axis-oriented bones.

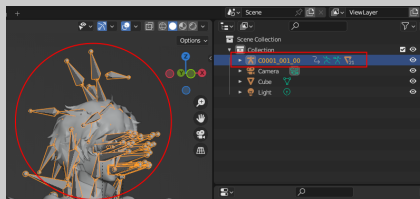
3.1.1 Bones

There are no special considerations for bones beyond how they behave in Blender. However, bones do also have some additional information that can be attached to them not representable in Blender. These can be accessed from the **GFS Node** panel in the **Bone Properties**.

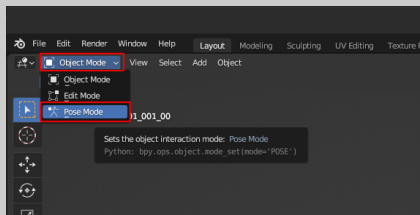
- **Unknown Float:** Unknown. Always seems to take a value of 1.
- **Properties:** Custom properties attached to the bone. See section 3.3 for further details.

Step-by-step: Accessing Extra Bone Properties

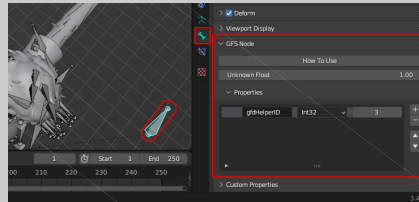
1. Select the armature of the model either in the Viewport or in the Outliner.



2. Switch to Pose Mode.



3. Select a bone and select the Bone icon in the Properties panel. You will find a panel called “GFS Node” containing the additional Bone Properties.



3.1.2 Rest Pose

GFS models have a **rest pose** in addition to the bind pose. The bones are imported in the **bind pose**, and the **rest pose** is added as a single-frame animation. The **rest pose** is used as a “default transform” for any animations, such that any bones without animation data uses the **rest pose** instead. If you want the **rest pose** to differ from the **bind pose**, create a single-frame action and push it to an NLA track called **Rest Pose**. This should always be at the **bottom** of the NLA editor, since it should have the lowest priority.

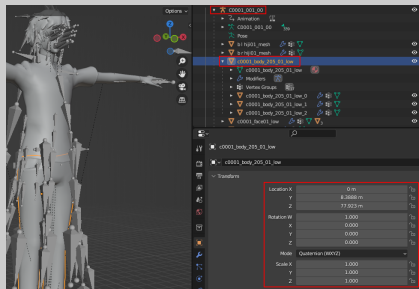
3.1.3 Meshes

Meshes that belong to a GFS model are parented under an armature. Meshes can use an armature deform with vertex groups as usual inside Blender. However, each vertex is only allowed to be part of a maximum of 4 vertex groups, and a maximum of 256 vertex groups are permitted across the entire model. If two meshes are weighted to the same bone, this counts as two vertex groups, since vertex groups are tied to the transform of a mesh in the GFS file format. The two vertex groups can be counted as one if meshes are parented to each other, as will be described momentarily.

The transform of the mesh inside Blender is exported as the mesh transform within the GFS files. Note however that meshes can be parented under other meshes to ensure that they share a transform. In this situation, the transform of the child-mesh is ignored and should always be a unit transform if you want to accurately see what will be exported in the viewport. In addition, if a mesh and its parent-mesh both use the same bone as a vertex group, it will be counted once instead of double-counted since both meshes have the same transform. Only one level of mesh-to-mesh parenting will be detected on export.

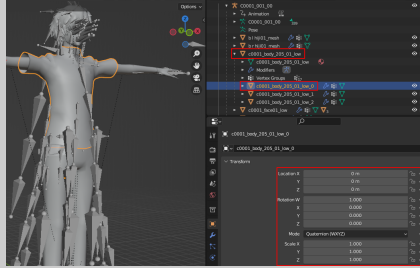
Step-by-step: Mesh Parenting for GFS

1. Meshes must be parented under an armature. The transform of a mesh dictates the transform of the mesh in the exported file.



2. Meshes can also be parented under other meshes. The transform of these child meshes will be ignored. Parenting a mesh under another

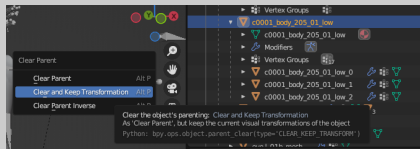
mesh ensures that they will share a transform in the exported file and vertex groups used by both meshes will not be double-counted by the file format.



Step-by-step: Unparenting Objects within Blender

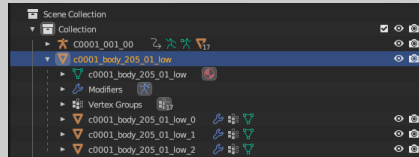
1. Select an object in the outliner or in the viewport.
2. Press **Alt** + **P** with the mouse in the Viewport.
3. Select either:
 - (a) **Clear Parent** if you want to unparent the object from its parent **and** move the mesh so that its transform is relative to the origin.
 - (b) **Clear and Keep Transform** if you want to unparent the object from its parent **and** edit the mesh's transform so that it stays in the same place in the viewport.

Ignore **Clear Parent Inverse**.

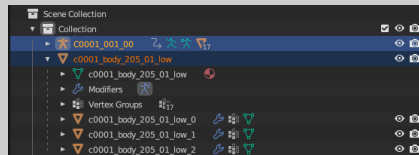


Step-by-step: Parenting Objects within Blender

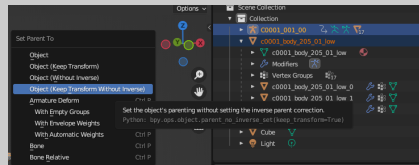
1. Select the child object in the outliner or in the viewport.



2. Select the parent object in the outliner or in the viewport with **Ctrl** + Click.

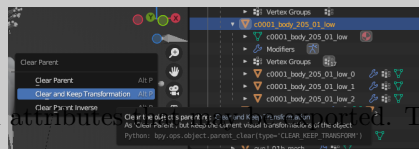


3. Press **Ctrl** + **P** with the mouse in the Viewport.
4. Select either:
 - (a) **Object (Without Inverse)** if you want to parent the object and reset its transform.
 - (b) **Object (Keep Transform Without Inverse)** if you want to parent the object **and** edit the mesh's transform so that it stays in the same place in the viewport.



5. In any situation, if after parenting your child object's transform is not what you expected, then:
 - Select the child object.
 - Press **Alt** + **P** with the mouse in the Viewport.
 - Click **Clear Parent Inverse**.

This will remove the hidden “parent inverse” matrix that sometimes gets inserted when parenting objects. This is harmless, but merely means that you object's transform may not align with what you see in the viewport if it is not a unit transform.



Meshes have certain attributes that are not supported. These are:

- **Export Bounding Box/Sphere:** Define a bounding Box / Sphere on

export.

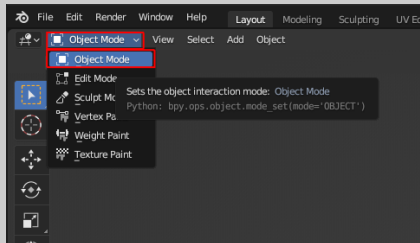
- **Export Normals/Tangents/Binormals:** Export these attributes. They are required for specific material options to work.
- **Unknown Flag:** The purpose of Unknown Flags is not known. Checking and unchecking these may cause or fix crashes.
- **Unknown 0x12:** Purpose unknown.
- The GFS Node sub-panel will appear if the Mesh is not parented under another Mesh. Refer to section 3.1.1 for further details.

Step-by-step: Accessing Extra Mesh Attributes

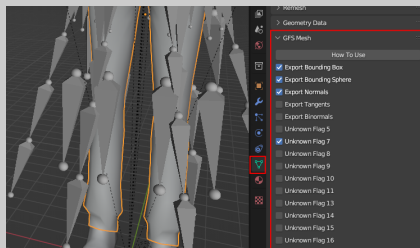
1. Select the mesh in the Outliner or in the Viewport.



2. Switch to Object Mode.



3. Select the Mesh icon in the Properties panel. You will find a panel called “GFS Mesh” containing the additional Mesh attributes.



WARNINGS

- Many tutorials and help articles will tell you to **Apply Transforms** to your mesh to reset their transforms to a unit transform. **This is dangerous.** When you **Apply Transforms**, you are translating, rotating, and scaling the vertices that make up the mesh, rather than applying an extra transform on top of the mesh. This means that, for example, your mesh will now use the origin as the reference point from which it rotates and scales, rather than the point around which the mesh was modelled.

3.1.4 Materials

Materials are not yet sufficiently understood to a degree whereby they can be consistently rendered in Blender. Therefore, materials are only represented in Blender as the Diffuse Texture of the material and all other properties are inferred from attributes or the names of Shader Nodes. Edits to the GFS Material attributes will, more often than not, simply lead to the model crashing any game that loads it. Due to this, editing materials from Blender is **not** currently recommended, except for the purposes of:

1. Adding textures to a material.
2. Perhaps minor tweaks to attributes.
3. Researching what the attributes do.

You will have much more success following the strategy of current model editing guides: post-processing the model in **GFD Studio** by copying materials from other models onto your exported model. How to do this is beyond the scope of this documentation but detailed tutorials are available online.

In the future, when materials are properly understood, a custom Shader Node Group should be implemented that allows the material to be rendered in a meaningful fashion, and allowing the values of attributes to be auto-calculated such that they do not crash the game. **There is not enough knowledge currently to do this and any contributions to material research enabling this feature is highly welcomed.**

There are then two essential pieces to Material editing:

- Texture Samplers
- GFS Material Attributes

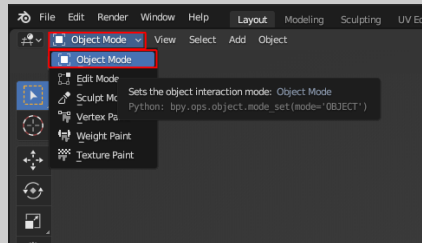
Texture Samplers: You can edit the textures used by a material by accessing the Shader Nodes. The first step is to open the Shader Node Editor.

Step-by-step: Opening the Shader Node Editor

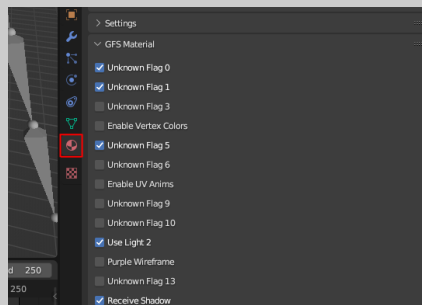
1. Select the mesh in the Outliner or in the Viewport.



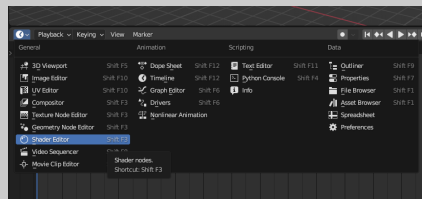
2. Switch to Object Mode.



3. Select the Material icon in the Properties panel.



4. Select the Shader Editor.



You can then set create new samplers to be exported and set their GFS attributes. There are nine types of textures available in the GFS format. Each material can have **one** of each type, and the plugin will recognise which type of sampler a node represents by the **name** of the node. The recognised names are:

- Diffuse Texture

- Normal Texture
- Specular Texture
- Reflection Texture
- Highlight Texture
- Glow Texture
- Detail Texture
- Night Texture
- Shadow Texture

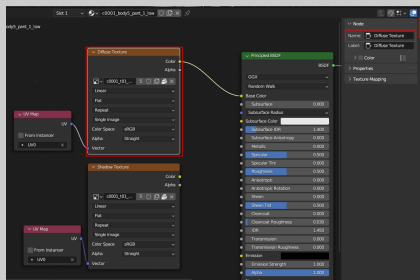
Note also that the UV maps must also have specific names due to the way they are stored in the GFS file format. The permitted names are:

- UV0
- UV1
- UV2
- UV3
- UV4
- UV5
- UV6
- UV7

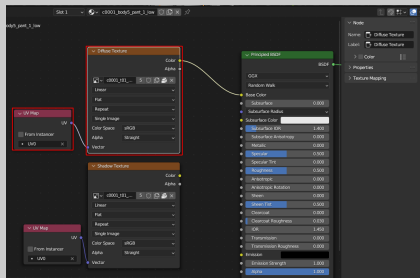
meaning that up to 8 maps are allowed per mesh.

Step-by-step: Editing Texture Samplers

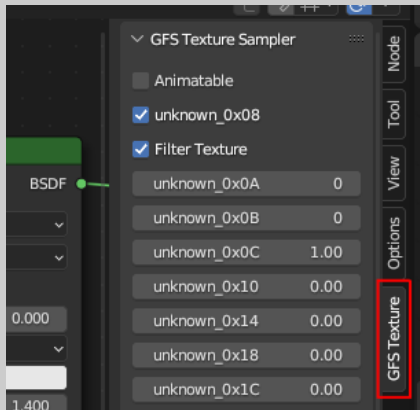
1. Inside the Shader Editor, select or create an Image Node.
2. Select the **Node** tab in the Sidebar. If the Sidebar is not open, press  with the mouse inside the Shader Editor.
3. Set the name of the node to one of the nine accepted names by changing the value in the Name field.



4. Select or create a UV Map node. Select a UV map present on the mesh from the drop-down on the node and hook the UV connector up to the Vector connector on the Texture node.



5. You can access the properties of a texture sampler by selecting the **GFS Texture** tab in the Sidebar. The properties for the sampler are given in the **GFS Texture Sampler** panel.

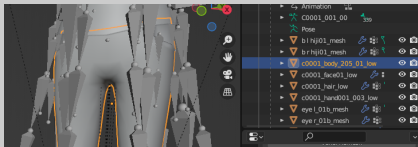


GFS Material Attributes: Materials have certain attributes that may be exported. These are:

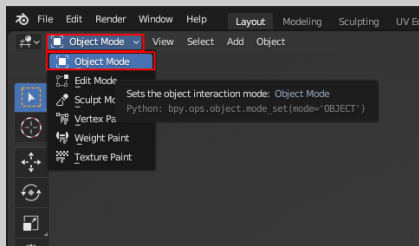
- **Unknown Flags:** The purpose of Unknown Flags is not known. Checking and unchecking these may cause or fix crashes.
- **Enable Vertex Colors:** Use Color Map data on the mesh.
- **Enable UV Anims:** Allow the UV coordinates of the mesh to be animated.
- **Use Light 2:** Purpose unknown.
- **Purple Wireframe:** Render the mesh as a purple wireframe.
- The GFS Node sub-panel will appear if the Mesh is not parented under another Mesh. Refer to section 3.1.1 for further details.

Step-by-step: Accessing Extra Material Attributes

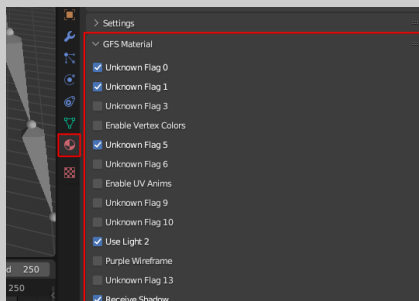
1. Select the mesh in the Outliner or in the Viewport.



2. Switch to Object Mode.



3. Select the Material icon in the Properties panel. You will find a panel called “GFS Material” containing the additional Material attributes.



3.1.5 Textures

Textures must be DDS images with either no compression, or DXT1, DX3, or DXT5 compression. BC7 textures cannot currently be loaded by Blender and are currently unsupported. In the future, BC7 textures should be loadable *via* an automatic conversion to a Blender-readable format.

Materials have certain attributes that may be exported. These are:

- **Unknown 1:** Purpose unknown.
- **Unknown 2:** Purpose unknown.
- **Unknown 3:** Purpose unknown.
- **Unknown 4:** Purpose unknown.

These attributes can be edited from the **GFS Texture** Sidebar panel on a Texture Image node in the Shader Editor, as described in section 3.1.4.

3.1.6 Cameras

Cameras are constrained to a bone in the model armature with a **Child Of** constraint. Due to the way that bones are imported, **you will need a 90 degree rotation in the Z axis** on the camera to make it point “horizontally”. The camera will be exported in whatever orientation it is in within Blender, however **it is strongly recommended** to just keep a 90-degree Z-rotation on the camera and do all other positioning using the bone it is constrained to.

For a camera to be recognised for export, you must do one of the following:

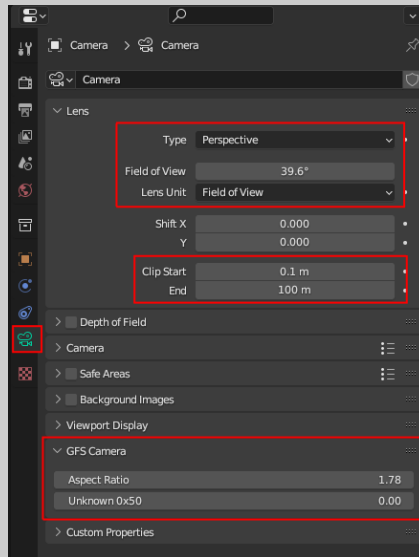
- Constrain the camera to follow a bone with a “Child Of” constraint.
- Parent the Camera to a bone. Note that parenting a Camera to a bone will place the camera at the tail of the bone rather than the head, although this should not affect the pose in which the camera is exported.

Some properties of Blender cameras will be exported to GFS camera properties, and some properties are not represented in Blender. This is described below.

Step-by-step: Accessing Camera Properties

1. In the **Camera Data** Properties panel, there are a number of camera properties. In the **Lens** section, the **Type** must be set to **Perspective** and the **Lens Unit** to **Field of View**. The **Field of View** attribute will be exported, although it may not exactly match up with the in-game field-of-view in the current version of the plugin. Additionally, the **Clip Start** and **Clip End** fields will be exported and will appear in-game as they do in Blender.
2. Furthermore, there are two properties not represented in Blender

that can be exported, found in the **GFS Camera**. **Aspect Ratios** can be represented in Blender, but only for the entire scene and not for individual cameras, so this is not currently represented. **Unknown 0x50** is unknown.



3.1.7 Lights

Lights are constrained to a bone in the model armature with a **Child Of** constraint. Due to the way that bones are imported, **you will need a 90 degree rotation in the Z axis** on the camera to make it point “horizontally”. The light will be **exported assuming a 90-degree Z-rotation on the light**—do all other positioning using the bone it is constrained to.

For a light to be recognised for export, you must do one of the following:

- Constrain the Light to follow a bone with a “Child Of” constraint.
- Parent the Light to a bone. Note that parenting a Light to a bone will place the light at the tail of the bone rather than the head, **and the light will be repositioned to the head of the bone during export due to GFS file format constraints.**

Some properties of Blender lights will be exported to GFS light properties, and some properties are not represented in Blender. This is described below.

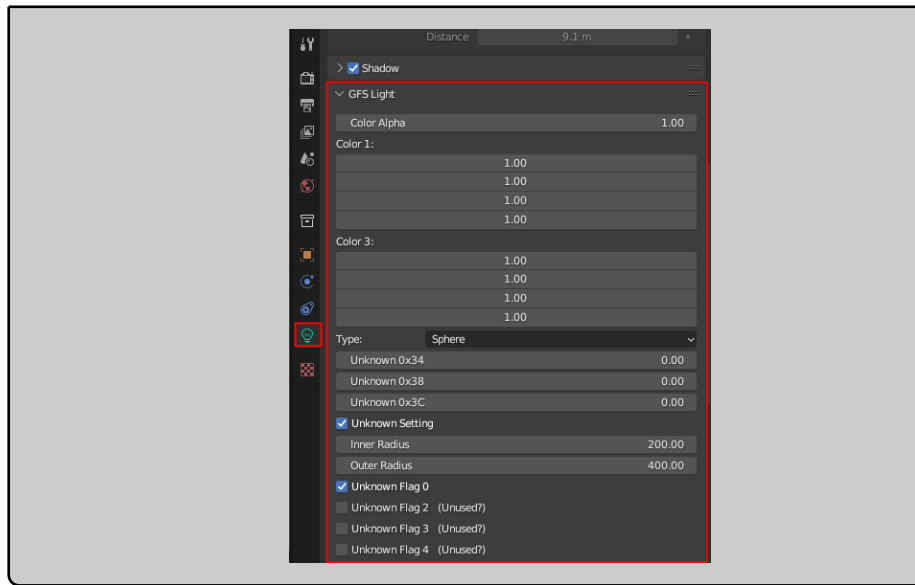
Step-by-step: Accessing Light Properties

1. In the **Light Data** Properties panel, there are a number of light properties. The only Blender property currently exported is the **Color** property, which sets the RGB light color.
2. The additional properties are in the **GFS Light** panel. There are first three elements common to all lights:
 - **Color Alpha** is the alpha channel of the light color.
 - **Color 1** is a color variable that does not seem to be used.
 - **Color 3** is a color variable that does not seem to be used.

There are then three kinds of light, selectable from the drop-down:

- **Type 1**: Unknown light type, does not appear to render in-game. Has no specific properties.
- **Sphere**: A point light that can have an attenuation radius. This has the unknown properties **unknown 0x34**, **unknown 0x38**, and **unknown 0x3C**, plus an **unknown setting**. If the unknown setting is active, the point light will illuminate everything within the **inner radius**, and attenuate to zero illumination between the **inner radius** and **outer radius**. Switching the setting off instead provides the settings **unknown 0x48**, **unknown 0x4C**, and **unknown 0x50**. Attempts to get a light to render using these variables have been unsuccessful.
- **Hemisphere**: A spot light that can have an attenuation radius. This has the unknown properties **unknown 0x54**, **unknown 0x58**, **unknown 0x5C**, **unknown 0x60**, **unknown 0x64**, **unknown 0x68**, **unknown 0x6C**, and **unknown 0x70**. This also has an **Unknown Setting** similar to the Sphere light. In principle these settings could be mapped to the Blender “Spot Shape”, but lights should be better understood overall before implementing this.

At the end of the properties is a list of flags. Only **flag 0** seems to be used amongst the provided flags, and its purpose is unknown.



3.1.8 Physics

Model physics are not currently editable from Blender, but should be preserved for re-export.

3.2 GFS Animations

In this section, the three major kinds of animation are covered. Sections 3.2.1, 3.2.2, and 3.2.3 cover the properties and Blender data required for each of these types to be exported and displayed correctly. Section 3.2.4 covers how to organise the set of animations in an Animation Pack that can be exported. In section 3.2.5, some useful information on editing Blend animations from within Blender is provided.

Currently, only **bone animations** are editable from within Blender. There are (at least) four other kinds of animation, but they are not exposed to the user for the following reasons:

- **Material Animations:** Not enough known about materials to create an animation node with animatable inputs.
- **Camera Animations:** No strategy currently developed to link Camera object animations to the corresponding Armature animation.
- **Morph Animations:** No strategy currently developed to link Shape Key animations to the corresponding Armature animation.
- **Animation Type 5:** Unknown animation type, so nothing can be done with it currently.

GFS Animations are expected to be rendered at **30 FPS**. Blender by default renders at **24 FPS**. Animations will be imported assuming a 30 FPS render speed so that animation frames can be placed at integer frame indices. You should change Blender’s animation speed to 30 FPS to preview the animations at the correct speed, either by changing it in the scene properties or by checking the box to set Blender to 30 FPS in the import window.

If an animation additionally has a custom “speed”, this speed is baked into the animation frames. This will cause the animation frames to be placed at non-integer frame values.

3.2.1 Normal Animations

“Normal Animations” are regular animations. The animated quantities in these animations should override those that are present in the Rest Pose. In-game, all interpolations are done with **linear interpolation (LERP)**, except for quaternion animations which are done with **spherical linear interpolation (SLERP)**. For this reason, to ensure that the animation you see in Blender is accurately reflected in-game, you should ensure the following settings are applied:

- The Strip Blending should be set to **REPLACE**.
- The keyframe interpolation should be set to **LINEAR** for all animations except for **rotation_quaternion**, which should be set to **BEZIER**.

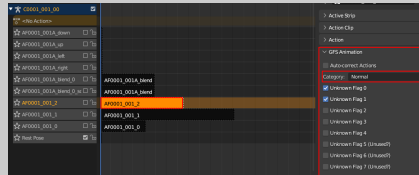
See Section 3.2.5, Subitem “Auto-Correcting Actions” for information on how to apply these settings automatically.

Normal Animations have a set of properties that also need to be exported. Details on these are given below.

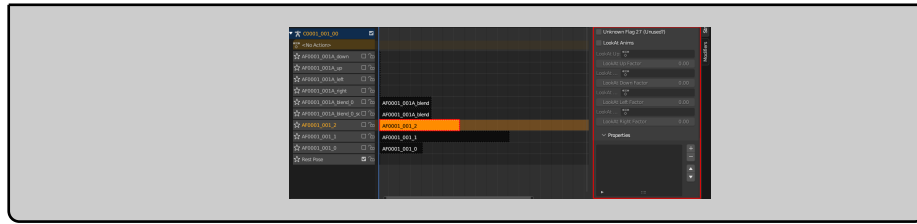
Step-by-step: Accessing Normal Animation Properties

1. The animation properties can be found by selecting a strip in the NLA editor and navigating to the “GFS Animation” panel in the “Strip” sidebar panel. You can use the **N** key to open and close the sidebar whilst the mouse cursor is inside the NLA editor.
2. The **Category** should be set to “Normal”. Most of the **Unknown Flags** are unused. The first five flags are usually ticked if the following animations are present:
 - (a) Flag 0 - Bone animations
 - (b) Flag 1 - Material animations
 - (c) Flag 2 - Camera animations
 - (d) Flag 3 - Morph animations
 - (e) Flag 4 - Unknown animations

but since these do not seem to influence how animations are displayed and do not have an exact correspondence to these types of animation being present, they are left as unnamed flags for now.



3. Further down, you can set “Look At” animations for a Normal Animation. A **LookAt animation** is a kind of **Blend Animation** that makes a character look up, down, left, and right. Further details can be found in sections 3.2.2 and 3.2.3. If you tick the box saying that an animation has lookat animations, you **must select all four animations you want to use as lookats** in order to successfully export. You can do this by clicking the animation boxes and selecting the animation that pops up. The only available animations are those that have been given the **LookAt** category. The role of the LookAt factors is unknown. In addition, animations can have GFS Properties like bones can. See section 3.3 for more information.

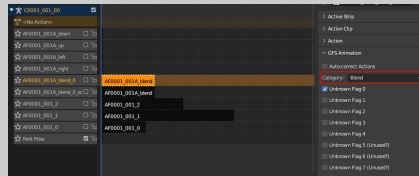


3.2.2 Blend Animations

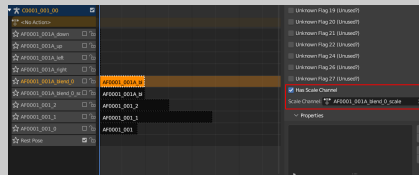
Blend Animations are animations that are overlaid on **Normal Animations**. The animations are combined by combining each transformation channel individually. Positions and scales are added, and quaternions are combined with quaternion multiplication. Therefore, the Strip Blending type should be set to **COMBINE**. Keyframe interpolation should be handled the same as the **Normal Animations**. However, Blender will attempt to **multiply** scale channels instead of adding them under the **COMBINE** Strip Blending mode. For this reason, scales should be separated to their own action with the **ADD** Strip Blending mode, and given the special **Blend Scale** category rather than the **Blend** category. Details on this are given in the Blend Properties Step-By-Step below, which also discusses other Blend animation properties. This means that any scales present in a **Blend Animation** will be ignored on export, and instead taken from a linked **Blend Scale Animation** if one is present.

Step-by-step: Accessing Blend Animation Properties

1. The **category** for Blend animations is **Blend**. The **Unknown Flags** are the same as the **Normal Animations**.



2. Instead of **LookAt** animations, **Blend** animations can link to a **Blend Scale** animation as described above. To link a **Blend Scale** animation, select one from the dropdown.

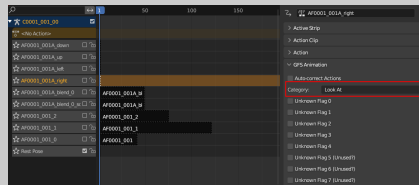


3.2.3 LookAt Animations

A **LookAt** animation is a special **Blend** animation used to represent looking left, right, up, and down. This is typically just a single frame representing the pose. **Normal Animations** can link to a **LookAt** animation from their properties panel. If the **LookAt** animation is not linked to a **Normal Animation**, it will not be exported.

Step-by-step: Accessing LookAt Animation Properties

1. The **category** for a **LookAt** animation is **LookAt**. All properties are the same as **Blend Animations**.



3.2.4 Animation Packs

Animations are exported as an **Animation Pack**. This animation pack is either exported **inside a GMD/GFS file**, or exported as a **separate GAP animation file**. When exporting a GMD/GFS file, you will be given the option to pack animations inside it or not. See section 4.1 for further information.

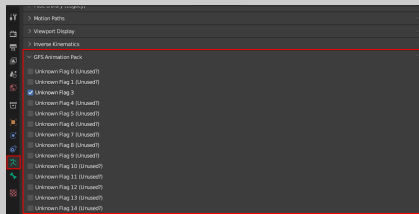
Currently, **only the most recently loaded animation pack properties are available inside the plugin**. That means that if you load a model that contains an animation pack, and then load a GAP file on top of it, the properties of the internal animation pack **will be overwritten**. Since models containing animations typically do not require external animation files, this should not be a problem. When editing external animation packs, ensure that the pack you intend to edit is loaded last. All available animations in the NLA track will be exported, so either delete unwanted animations from the NLA editor before export, or only load the animation pack you wish to edit. If you pack too many animations inside an animation pack, you can fix this with an external program such as **GFD Studio** or the Python interface.

If a sensible way to easily separate individual animation packs without making it complicated to assign animations to them can be found, this feature will be reworked to allow multiple animation packs to co-exist within Blender and export individually.

Animation Pack properties are stored on a model armature. The section below will cover the specifics of this.

Step-by-step: Accessing Animation Pack Properties

1. Animation Pack properties can be found in the **Armature Data** panel of the Properties Panel.
2. The **Unknown Flags** are unknown and appear to be completely unused except for Flag 3. The purpose of Flag 3 is unknown. An animation pack can also have some global **LookAt** animations, similar to a **Normal Animation**. See section 3.2.1 for further information.
3. Bear in mind that **only the properties of the most recently imported animation pack for a particular model** are saved.

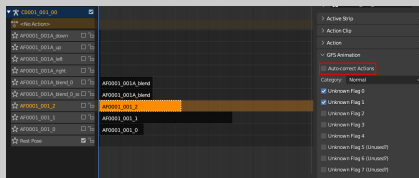


3.2.5 Tips for Editing Animations in Blender

Auto-Correcting Actions

Step-by-step: Auto-Correcting Actions

1. The interpolation and strip blending of an NLA strip and action can be automatically set when switching the action type. Do this inside the Animation Properties.
2. The animation properties can be found by selecting a strip in the NLA editor and navigating to the “GFS Animation” panel in the “Strip” sidebar panel. You can use the **N** key to open and close the sidebar whilst the mouse cursor is inside the NLA editor.
3. Check the **Auto-Correct Action** box.



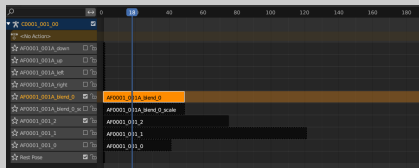
4. Changing the animation **category** will now automatically change the interpolation and strip blending types for those appropriate for

the selected category. This will not be done when first checking the box, so switch to a different category and back to apply the changes.

Previewing with the NLA Editor

Step-by-step: Previewing with the NLA Editor

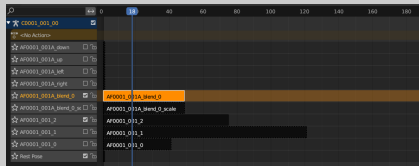
1. Unmute whichever animations you want to preview. You should unmute a **single Normal Animation** plus whatever **Blend Animations** you want to see overlaid on it.
2. The **Rest Pose** should **always** be unmuted.
3. Actions at the **top** of the editor have the highest priority. Should should have the **Rest Pose** at the bottom, followed by the **Normal Animations**, and then any **Blend Animations**.



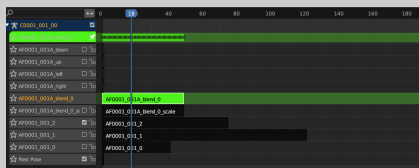
Previewing Blend Animations during Editing

Step-by-step: Previewing Blend Animations during Editing

1. Unmute whichever **Normal Animation** you want to use as a base, and the **Blend Animation** you want to edit in the NLA editor.



2. Press **Tab** to pin the action to the Action Editor.



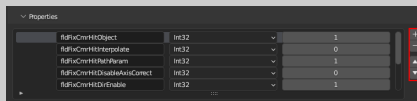
3. You can now edit the action in the Action Editor whilst previewing the full effect of the other unmuted NLA tracks.

3.3 GFS Properties

Bones, armatures, meshes, and animations can have additional **GFS properties**. These can be added with the **GFS Properties** collection widget in their corresponding UI panels. When adding a property, you will be required to provide a name, data type, and then the data the property will store. Although the name fields are free text fields, there do appear to be specific properties that are looked for and will do certain things. For this reason the properties box may be replaced with sub-panels implementing this behaviour automatically once it is fully understood. For now, like with everything else that isn't fully understood about the format, any edits you make will have to be somewhat manual.

Step-by-step: Manipulating GFS Properties

1. Navigate the the properties panel for the selected object.
2. Within the object's properties panel, locate the **GFS Node** sub-panel. Within that panel, locate the **Properties** subpanel.
3. Properties attached to the object will be displayed in the box. The selected property is highlighted in a different colour. You can add, remove, and re-order properties with the four buttons on the right of the box.
4. Change the name of a property by editing the value in the leftmost box.
5. Change the property data type by selecting a value from the drop-down.
6. Change the attached data by editing the rightmost box(es).



3.4 EPLs

EPL files are not currently editable in Blender. GFS files containing EPLs are loadable, but the EPL data will not be accessible. GFS files contained within EPLs can be extracted by other programs or by using this plugin as a library for the Python programming language, as described in section 5. These extracted GFS files may then be possible to open in Blender, edit, and then re-inject into the EPL files with other programs or *via* the Python interface. Note however that, as of this version of the plugin, this is not well-tested nor intended to be an end-user feature. Users are welcome to use the library in this way, but are reminded that since it is an unsupported part of the library, pull requests that improve the feature are significantly more likely to get attention than issue tickets asking for improvements to it.

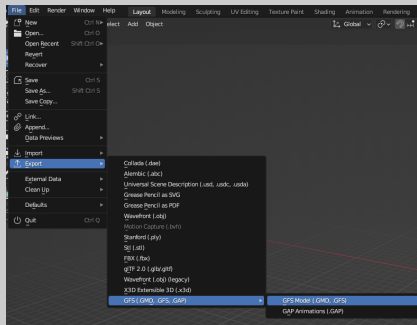
If sections of EPLs can be sensibly imported and exported from Blender, such as submodels, EPLs may be better supported in future versions of the plugin.

4 Export

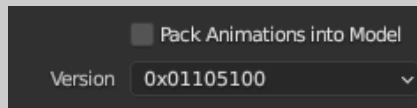
4.1 Exporting Model (GMD/GFS) Files

Step-by-step: Accessing the Model Export Menu

1. Open the File menu and navigate to the **Export > GFS > GFS Model** submenu item.



2. From the export options on the right, choose:
 - Whether to export animations packed into the model.
 - Which file version to export as.
 - Whether to strip any vertex groups from the export that do not correspond to a bone in the armature. Leaving this option unticked means that undefined vertex groups will cause an error when attempting to export.

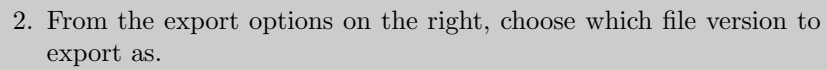


3. Set a filename and export.

4.2 Exporting Animation (GAP) Files

Step-by-step: Accessing the Animation Export Menu

1. Open the File menu and navigate to the **Export > GFS > GAP Animation** submenu item.



3. Set a filename and export.

5 BlenderToolsForGFS as a Python Library

NOTE: This section is for users of the Python programming language who wish to write scripts using the plugin. A user will not get much out of this section without some knowledge of Python. If you need to use the library's scripting API, you are encouraged to pick up the basics of Python using the wealth of information freely available on the internet.

NOTE: The API is currently unstable. This document should be correct for whatever release of the tools it is labelled for, but the same cannot be said for development versions. The documentation should be fully updated once feature development for each new release is complete. You should expect the API to change between versions as the file format becomes better understood and smarter ways of arranging and creating data become clear.

The Blender plugin can also be used as a plain Python library for manipulating GFS and EPL files. This is possible because all Blender-related code is not loaded by default when initialising the plugin, and because the file-editing code is well-separated from the Blender interface. Furthermore, the non-Blender part of the plugin only utilises the standard library, and therefore no additional third-party libraries are required in order to use the plugin as a library. Using the plugin in this manner is outlined in section 5.1. **However, bear in mind that this is not fully documented since it is not yet intended as an end-user feature.** Adventurous users can use the Python API, but should bear in mind that there are many ways to create annoying data inconsistencies. This can be remedied with further development to make the Python API safer to use by an end-user.

If you have the 'bpy' package installed, you can activate the plugin's Blender interface and write Blender scripts outside of Blender. You can also write these scripts inside Blender with the scripting feature. Notes on how to use the plugin in this manner are given in 5.2.

5.1 Python Library Usage

GMD, GFS, GAP, and EPL files can be manipulated with the Python API, which can be imported independently from the Blender Interface.

5.1.1 GFS/GMD/GAP

The first thing to do is to import the **GFSInterface** object.

```
1 from BlenderToolsForGFS import GFSInterface
```

This object can be used to access various elements of the file. First of all, you will need to load a file.

5.1 Python Library ~~Usage~~ BLENDERTOOLSFORGFS AS A PYTHON LIBRARY

```
1 gi = GFSInterface.from_file(filepath=...)
```

The GFSInterface can be written to a file with the **to_file(filepath, version)** method.

```
1 # version is an int, e.g. 0x01105100
2 gi.to_file(filepath=..., version=...)
```

The available attributes of the GFSInterface are given below. These are likely to be changed to read-only properties with **add()** and **remove()** methods in the future.

Name	Type	Description
meshes	List[MeshInterface]	A list of meshes.
cameras	List[CameraInterface]	A list of cameras.
lights	List[LightInterface]	A list of lights.
epls	List[EPLInterface]	A list of epls.
bones	List[NodeInterface]	A list of nodes/bones.
materials	List[MaterialInterface]	A list of materials.
textures	List[TextureInterface]	A list of textures.
animations	List[AnimationInterface]	A list of normal animations.
blend_animations	List[AnimationInterface]	A list of blend animations.
lookat_animations	LookAtAnimationsInterface	A container for the global lookat animations.

New elements can be added to these attributes with the following methods.

Name	Return	Inputs	Input Description
add_mesh	MeshInterface	node_id	Node that the MeshInterface should be attached to.
		vertices	A list of Vertex objects.
		material_name	The name of the material used by the mesh.
		indices	Vertex indices that are used to create the triangles. This is a flat list, with every three indices creating a triangle.
		morphs	A list of lists of 3D position vectors. Each sub-list represents a position offset for its corresponding vertex. Each sub-list should have the same number of entries as the mesh vertices.

		unknown_0x12	Unknown integer. Usually 0.
		unknown_float_1	Unknown float. May be None is unknown_float_2 is also None.
		unknown_float_2	Unknown float. May be None is unknown_float_1 is also None.
		keep_bounding_box	Bool. Whether to create a bounding box on export.
		keep_bounding_sphere	Bool. Whether to create a bounding sphere on export.
add_camera	CameraInterface	node_id	Node that the CameraInterface should be attached to.
		view_matrix	A row-major 4x4 transform matrix represented as a flat list of 16 elements.
		zNear	The near clip distance.
		zFar	The far clip distance.
		fov	The Field of View in degrees.
		aspect_ratio	The aspect ratio.
		unknown_0x50	Unknown float.
add_light	LightInterface	node_id	Node that the LightInterface should be attached to.
		type	1 = Type1, 2 = Sphere, 3 = Hemisphere.
		color_1	A 4-vector representing an RGBA color. Apparently unused.
		color_2	A 4-vector representing an RGBA color. Main light color.
		color_3	A 4-vector representing an RGBA color. Apparently unused.
add_epl	EPLInterface	node_id	Node that the EPL should be attached to.
		binary	An EPLBinary object. You can create EPLBinaries manually, but it is recommended to consider this function off-limits for creating new EPLs.
add_node	NodeInterface	parent_idx	Index of the NodeInterface's parent NodeInterface.
		position	3D position vector.
		rotation	XYZW quaternion.

		scale	3D scale vector.
		unknown_float	Unknown.
		bind_pose_matrix	A row-major 4x4 transform matrix represented as a flat list of 16 elements.
add_material	MaterialInterface	name	The material name.
add_texture	TextureInterface	name	The texture name.
		data	A bytes-like object representing raw data for a DDS file.
		unknown_1	Unknown int. Usually 1.
		unknown_2	Unknown int. Usually 1.
		unknown_3	Unknown int. Usually 0.
		unknown_4	Unknown int. Usually 0.

TODO: Description of each Interface

5.1.2 EPL

TODO

5.2 Blender Script Usage

Most Blender scripts will need to begin by importing the **bpy** module.

```
1 import bpy
```

If you are writing a script outside of Blender, you will also need to add the **BlenderToolsForGFS** module to your import path (or create your python script in the same directory as a copy of the BlenderToolsForGFS library), and then import and register the module with Blender by calling the module's **register()** method.

```
1 import BlenderToolsForGFS
2 BlenderToolsForGFS.register()
```

After the module is registered, you can import and export data using the operators that the plugin adds for these purposes.

```
1 bpy.ops.import_file.import_gfs(filepath=...)
2 bpy.ops.import_file.import_gap(filepath=..., armature_name
  =...)
3
4 # Need to select the armature in order to use export functions
5 bpy.data.objects[...].select_set(True)
6
7 bpy.ops.export_file.export_gfs(filepath=...)
8 bpy.ops.export_file.export_gap(filepath=...)
```

These operators have the following keyword arguments:

import_gfs		
Keyword	Type	Description
filepath	string	The filepath to the file to import.
set_fps	bool	Switches Blender animation to 30 FPS. Default: False.
debug_mode	bool	Allows exceptions to be raised instead of silently suppressed by popup windows. Default: False.
merge_vertices	bool	Attempts to create a manifold mesh out of the input vertices. Default: True.
import_gap		
Keyword	Type	Description
filepath	string	The filepath to the file to import.
set_fps	bool	Switches Blender animation to 30 FPS. Default: False.
debug_mode	bool	Allows exceptions to be raised instead of silently suppressed by popup windows. Default: False.
export_gfs		
Keyword	Type	Description
filepath	string	The filepath to the file to import.
version	string	The file version to export as. Default: 0x01105100.
debug_mode	bool	Allows exceptions to be raised instead of silently suppressed by popup windows. Default: False.
pack_animations	bool	Whether to pack animations into the model or not. Default: False.
export_gap		
Keyword	Type	Description
filepath	string	The filepath to the file to import.
version	string	The file version to export as. Default: 0x01105100.
debug_mode	bool	Allows exceptions to be raised instead of silently suppressed by popup windows. Default: False.

TODO: Custom PropertyGroups

6 Future Development

Listed here are a few directions that future development could go. This is not a promise of feature development; just an inexhaustive record of ideas for any contributors who might want to improve the tools in some fashion.

6.1 Blender Interface

- Create an accurate Shader Node for GFS materials. This will require the material data structures to be fully researched beforehand.
- Redesign the animation import/export so that more than just node animations can be imported. Material animations would need to be somehow linked to a corresponding node animation, and so forth. An idea on how to do this would be to have a list of exported animations on the animation pack, and each animation entry can contain a **CollectionProperty** of **PointerProperties** to **bpy.types.Actions**. Each animation can additionally have a checkbox next to it that will mute or unmute **bpy.types.Actions** that are linked to it simultaneously, so that the user does not have to individually unmute every animation on every object to preview the entire animation.
- Related to the above: a **CollectionProperty** of Animation Packs could be implemented so that multiple Animation Packs can be imported and exported. Since we would already be collecting exported animations into a **CollectionProperty**, that would just mean that each AnimationPack would be in charge of its own **CollectionProperty** of animations.
- After the previous points are implemented and are guaranteed to work well both technically and from a user's perspective, then material, camera, and morph animations can be added.
- Physics colliders and joint physics can be imported as meshes with rigid body physics and a bunch of properties. Buttons to generate these meshes can be added to a menu somewhere.
- EPL submodels should be imported into the scene tree too, but only after the animation and material refactors have been implemented.
- GFS Properties could be turned into a group of settings instead of an arbitrary list (or both could be available). For example, any Properties required to create collision triggers could be added with a subpanel rather than asking the user to copy out a list of collision properties. However, the various sets of properties would have to first be researched and property understood.
- The vertex-merging algorithm should be refactored to work by merging vertices such that **any face** will not be removed, rather than just checking if there are two overlapping antiparallel faces.

6.2 Python API

- RootNodes should be moved out of the node list. Any parentless nodes in the GFS node list should be automatically parented to the RootNode, which must always exist by construction. Currently the GFSInterface is fragile, in the sense that if the first node is *not* the root node, everything explodes.
- Cameras need a proper Interface.
- Lights need a proper Interface.
- The AnimationInterface needs to be more cleanly designed: it currently has a lot of awkward heuristics to serialise node animations and can duplicate tracks and so forth due to a lack of understanding of how all the animation keys work.