



SEXLAB ANIMATIONS LOADER

MAKING ANIMATION PACKS

Adding animations to Skyrim now cannot be any simpler. The term 'no scripting or CK experience necessary' is true; you do not need to compile any scripts or load the Creation Kit. The only thing required is to edit a text file which already has a template for you to work off. And this guide will take you through doing just that.

Before using this guide, ensure you have at least read the instructions on the download page for SexLab Animations Loader on LoversLab and installed Python 3.x.

The version of this guide is written by Rydin. The version number is 1.1.

THIS GUIDE IS ONLY FOR MODDERS LOOKING TO CREATE A PACK OF ANIMATIONS USING THE HKX FILES TO DISTRIBUTE AND SHARE WITH OTHER USERS. IF YOU ARE ONLY INSTALLING THE PACKS, PLEASE FOLLOW THE INFORMATION ON THE DOWNLOAD PAGE. THIS GUIDE IS NOT FOR USERS AND AS NO INFORMATION ON INSTALLING THE PACKS AS A USER.

UNDERSTAND THE BASICS

If you understand what HKX files and the standard file directory set up for mods, you can skip this section.

Before getting started, you need to know what a HKX file is. A HKX file is the extension used for animations within Skyrim. Animators convert their animations for their chosen animating programs to HKX files which can then be imported into game. Unfortunately, you cannot just drop the HKX file into the animation folder and expect it to work. It has to have a script that tells the game how, where and when to use the animation.

When a HKX is used for an animation, it is normally names in a set convention to make it easy to identify the animation, what actor it is for and what place it has in a sequence. An animator can name their animations anything they want but modders may have a hard time setting up a script for it if they do not understand what each HKX files does and who it affects.

It is very important that you understand this as this is the structure that the SLAnimGenerate.pwy will require in order to read the HKX files. The formula/structure is:

`[animator][_animname]aX_sY.hkx`

To give you an example of one:

`rydin_kissing_standing_a1_s1.hkx`

Now let's break this down. The animator in this case is "rydin" and the animname is "kissing_standing", telling us it is a kissing scene in the standing position. X and Y both equal 1. The "a" stands for actor and the "s" stands for scene. What this tells us is that the animation affects actor 1 and that it is scene 1. If there are more than one actor or scenes, then "a" and "s" will have other numbers after them.

You need to make sure the files have no spelling mistakes or deviations from the structure. Capitals or lower case letters don't affect it but if you then must all match.

Example of a matching set:

`rydin_kissing_standing_a1_s1.hkx
rydin_kissing_standing_a1_s2.hkx
rydin_kissing_standing_a2_s1.hkx
rydin_kissing_standing_a2_s2.hkx`

Example of a mismatching set:

```
rydin_kissingstanding_a1_s1.hkx  
rydin_kissingstanding_a1_s2.hkx  
rydin_kissing_standinga2_s1.hkx  
rydin_kissing_standinga2_s2.hkx
```

In the above example of a mismatching set, the SLAnimGenerate.pwy will generate errors because will not recognize that this is the same scene as one set is "kissingstanding" and the second is "kissing_standing". Also because there is a missing underscore between the animname and actor, it will not recognize this HKX file as it is not in the correct structure.

It is important to note that actor 1 is generally the player role when animations are chosen by SexLab (unless roles are defined by a script). Actor 2 and above will be either player or creature roles.

You also need to understand the basic file structure of files. If you are a beginner at modding, you may not have looked inside the data folder much. You need to know where to put your HKX files. Putting them in the incorrect folders will mean they may not be recognized by both SLAnimGenerate and by the game itself, resulting in the actors standing around in idles during animations.

HKX files belong in the appropriate folders within the meshes folder. For example, actors all belong in the following folder:

```
Data/meshes/actor/character/animations/[packname]/
```

In the animations folder, "packname" will be whatever you call your pack.

Creatures have their own folders, so again you will need to put them in their corresponding folder. For example, the draugr animations belong in:

```
Data/meshes/actor/draugr/animations/[packname]/
```

So if for example If actor 1 and actor 2 are to be human animations, then all HKX files will go into the /character/animations folder. However if actor 1 is human and actor 2 is creature, then actor 1 animations will go into the /character/animations folder and actor 2 will go into the /draugr/animations folder. Failure to do so will result in the animations not playing in game.

It is important that you familiar yourself with the file structures and formulas. It is the ground work to understanding what mods do and where the information needs to be and is pulled from.

PREPARING THE PACK

If you are just editing a pack that has already been made and you are just adding more animations, you can skip this section.

Once you have the pack downloaded, the first thing to do is prepare the pack you want to make.

- *It is recommended to work with the pack outside of your data folder. If you make any mistakes you can easily delete any mistakes or start a fresh without potentially damaging your existing files for the mod.*

You should start off with the following files:








Name	Date modified	Type	Size
Interface	13/12/2015 22:17	File folder	
Scripts	13/12/2015 22:17	File folder	
SLAnims	13/12/2015 22:17	File folder	
Readme - SLAnimLoader.txt	13/12/2015 22:13	Text Document	4 KB
SLAnimLoader.esp	12/12/2015 22:22	ESP File	1 KB

What you would need to create is a folder path to where your animations are going to be. In this guide the pack will be called 'SLAddAnims'. It will be highlighted in yellow. During this guide, if you have called your pack something else, anything highlighted in yellow should be renamed to match what you want your pack to be called.

The path you need to create is meshes > actors > character. Once these folders have been created, you need to put two folders in the character folder, 'animations' and 'behaviors'. (*It is important you spell the folders as they are shown in the guide. Spelling them differently will cause the pack to get errors*).



Name	Date modified	Type	Size
animations	16/12/2015 15:48	File folder	
behaviors	16/12/2015 16:01	File folder	

Inside the animations folder you will put the title of your pack. Inside the below example we have used **SLAddAnims**:

Name	Date modified	Type	Size
 FunnyBiz	16/12/2015 13:39	File folder	
 LeitoHumanAnimationPack	20/12/2015 10:59	File folder	
 LietoCreatureAnimationPack	20/12/2015 10:59	File folder	
 LietoFurnitureAnimationPack	20/12/2015 10:59	File folder	
 MitosAnimationPack	20/12/2015 10:59	File folder	
 RydinAnimationPack	17/12/2015 12:28	File folder	
 SLAddAnims	20/12/2015 11:00	File folder	

It is recommended that when you make a name for your pack to put this name without spaces as the folder name.

Inside the **SLAddAnims** folder, you would place your HKX files in here:

Name	Date modified	Type	Size
 exmaple_position_act_a1_s1.hkx	17/12/2015 12:26	HKX File	487 KB
 exmaple_position_act_a2_s1.hkx	17/12/2015 12:05	HKX File	269 KB

With these files in place, you are now ready to start editing the text files to create the scripting for Skyrim.

PREPARING THE SOURCE MATERIAL

The next thing to do is to prepare and edit the text file which will hold all the details for your pack which the python program will read to create the appropriate scripts. Head over to the SLAnims > source folder to find the example.txt file inside.

This little text file has been writing out by the mod creator Orxx on how to setup the source for the python script to generate the scripts. You can follow the guide inside the text file or continue reading this guide. Whichever guide you read, it is important that you follow them very closely, as errors in your setup can cause errors in the python generate file or the output scripts for Skyrim and FNIS to read.

Anything line with the hash symbol (#) at the start is just a comment. These will not be read by the python generate file when creating the scripts. There are a few lines in the top half of the text file which do not have the hash symbol in front and these lines need to be amended. Below are those extracted lines:

```
is_example = True
mcm_name = "Super Cool"
anim_dir("SuperCool")
anim_id_prefix("SC_")
anim_name_prefix("Super Cool ")
common_tags("SuperCool")
```

You can delete the comments with the hash symbol and leave the above lines in if you are confident and comfortable that you understand what you are doing. This is optional. Just be sure not to delete the bottom section that starts 'Animation(' as this is your animation coding. If you do, do not worry as the coding will be in this guide for you to copy and paste.

It is important that you understand what each of these lines of code mean to you and your pack to help you make the best use of it. Don't worry, they are very simple.

is_example = True

This line was added by the mod author to stop it showing up in the SLAnimGenerate.pyw dialogue. Changing this to false means that it will show up as an option when using the SLAnimGenerate.pyw file. You can either delete this line entirely or you can change it to false. I recommend changing it to false and keeping the line in case you need to hide these text files from the generator in the future.

mcm_name = "Super Cool"

This will be the name of your pack as it shows up on the Mod Configuration Menu (MCM) in game.

anim_dir("SuperCool")

This will be the name of the animation directory. This should be the folder with the HKX files inside them that we set up earlier in this guide. It is recommended to have no spaces in this name.

anim_id_prefix("SC_")

This will be the prefix name of the animations that will be coded for the FNIS file. Most commonly, this

will be the animator's name, such as Leito, Arroko or Rydin. It is recommended to use an underscore (_) after the name to keep the code clean and easily readable. It will be also used in the animation coding covered later in this guide where examples will be given.

anim_name_prefix("Super Cool ")

This will form part of the name of the animation as it will appear in the MCM in game. Most commonly, this will be the animator's name, such as Leito, Arroko or Rydin. It must have a space before the last quotation mark (") or it will join up with the title of the animation in the MCM. It will be also used in the animation coding covered later in this guide where examples will be given.

common_tags("SuperCool")

This is where you can set common tags that will feature on all of the animations in the pack. Most commonly, this will be the animator's name, such as Leito, Arroko or Rydin. If you were making a pack that had all furniture animations, you could also add Furniture in this section. If the tags is not featured in all animation, do not add it here as we will be able to add individual tags later in the animation coding.

Here is the base coding without the SuperCool examples:

```
is_example = True
mcm_name = ""
anim_dir("")
anim_id_prefix("")
anim_name_prefix("")
common_tags("")
```

With the information above and using **SLAddAnims** as our example, the script would look like the following:

```
is_example = False
mcm_name = " Additional SL Anims "
anim_dir("SLAddAnims ")
anim_id_prefix("SLAA_")
anim_name_prefix("SLAA ")
common_tags("SexLab")
```

To give an example of an animator such as Rydin, it would look like the following:

```
is_example = False
mcm_name = "Rydin's Animations"
anim_dir("RydinAnimationPack")
anim_id_prefix("rydin_")
anim_name_prefix("Rydin ")
common_tags("Rydin")
```

Now you will be ready to code in the animations.

ADDING THE ANIMATIONS

Adding the animations has never been easier. The code is really simple and short. There is no papyrus code to deal with and no compiling needed. Knowing what to put where, why to add it and what it does will be covered in this topic to help you get to grips with adding the animations.

Below is the basic code for adding animations:

```
Animation(  
    id=" ",  
    name=" ",  
    tags="",  
    sound="",  
    actor1=,  
    actor2=,  
)
```

Each time you add a new unique animation, you will use the above code as the base code. There is an extra line for stage parameters such as open or closed mouth and if the actor is silent or not but you would only add this line if you need to change anything about the animation positions. We will cover this later in the topic.

Let's go over what each of the lines of code mean and do:

id=" ",

This line will be the ID of your animation. This will be how SexLab recognizes your animation if called upon by name by any other mod. It ties up with anim_id_prefix to create the full ID of your animation. Your ID should relate to the animation and generally be similar to the name of the animation but still be unique that no other animation ID ends up with the same name. It is recommended to use an underscore instead of a space or no spaces at all.

To give an example, let's say we are adding a passionate kissing animation.

```
anim_id_prefix("SLAA_") + id=" Passionate_Kissing" = SLAA_ Passionate_Kissing
```

In the above example the prefix is added to the ID of the animation to create the full ID of the animation. When the python generator turns the code into a test file for FNIS to read, it will use the above logic to create the necessary lines of codes for FNIS to read. There is no need to add the actor (A) or stage (S) details as the generator will pick this up automatically. We will cover this more later in the guide.

name=" ",

This will hold the name of your animation as to how it will be seen in the MCM. It follows the same logic that the ID has, where it will tie up with anim_name_prefix to create the full name of your animation. These can have spaces and are recommended as this is what you will see in game.

To give an example, let's say we are adding a passionate kissing animation.

```
anim_name_prefix("SLAA ") + name=" Passionate Kissing" = SLAA Passionate Kissing
```


tags="",

This is where we can add the individual tags for the animation. It is important that we add the tags correctly as this will be what most mods use to call our animations or update things like statistics and such. Adding the wrong tags or spelling it incorrectly could result in the animations not being called upon or utilized correctly. There should be no spaces and each tag is separated by a comma (,).

sound="",

You can set the sound effects of the overall animation in here. The allowed values are Squishing, Squirting, Sucking, SexMix and NoSound. If there are stages where there needs to be a different sound, then put the sound in this box that will play in most of the animations. We can add an individual parameter later for individual stages. This does not affect the moaning or voices of the actors, only the sound effects.

actor1=,

This is where you will define the details for actor 1, which will majority of the time be the player and female.

actor2=,

This is where you will define the details for actor 2, which will majority of the time be an NPC or creature.

Now that we know what each section does, let's look at a typical code set up. We will use the example of passionate kissing.

```
Animation(  
    id="PassionateKissing ",  
    name=" Passionate Kissing",  
    tags="Loving,Kissing",  
    sound="NoSound",  
    actor1=Female(),  
    actor2=Male(),  
)
```

If we put the above code with the code we made earlier in prepared the source, we would have the following.

```
is_example = False
mcm_name = " Additional SL Anims "
anim_dir("SLAddAnims ")
anim_id_prefix("SLAA_")
anim_name_prefix("SLAA ")
common_tags("SexLab")

Animation(
    id="PassionateKissing ",
    name="Passionate Kissing",
    tags="Loving,Kissing",
    sound="NoSound",
    actor1=Female(),
    actor2=Male(),
)
```

We have now made one pack which features one animation to be added into game. To add more animations, we would simply add another animation coding for the new animation.

```
is_example = False
mcm_name = " Additional SL Anims "
anim_dir("SLAddAnims ")
anim_id_prefix("SLAA_")
anim_name_prefix("SLAA ")
common_tags("SexLab")

Animation(
    id="PassionateKissing ",
    name="Passionate Kissing",
    tags="Loving,Kissing",
    sound="NoSound",
    actor1=Female(),
    actor2=Male(),
)

Animation(
    id="SlowKissing ",
    name="Slow Kissing",
    tags="Loving,Kissing",
    sound="NoSound",
    actor1=Female(),
    actor2=Male(),
)
```

We can do this as many times as we like to add as many animations as we want.

ADDING PARAMETERS

Parameters are added extra amendments to the animations such as positions, schlong alignment and actor sounds. We may also add parameters to the actors such as adding cum, objects and more. In this section, we will discuss the parameters separately.

It is important to know that you only need to add parameters to stages that need amending from the default position. For example, if a scene has 5 stages and in the third stage needs to open their mouth, we would set a parameter for that stage only to make the actor open their mouth. We do not need to define every stage unless every stage needs to be amended from the default.

The defaults are that no cum is applied, the actors are not silent, their mouths are closed, there are no SOS alignments, no rotations needed and there is no object attached.

For example, if doing a silent actor parameter for stages three and four, below is what NOT to do:

```
a1_stage_params = [  
    Stage(1, silent=False),  
    Stage(2, silent=False),  
    Stage(3, silent=True),  
    Stage(4, silent=True),  
    Stage(5, silent=False),  
],
```

Whereas there would be no harm, it is not required and will just make you spend more unnecessary time. You only need to define the stages that have changes:

```
a1_stage_params = [  
    Stage(3, silent=True),  
    Stage(4, silent=True),  
],
```

Below is a description of each parameter.

ADDING CUM

To add cum to an actor is very simple. Firstly, you need to decide if you adding cum to actor 1 or actor 2. Secondly you need to decide where the cum will be added. A majority of the time the parameter will be added to actor 1 who is generally the female character but it is not exclusive.

To add cum, you would add the following line after the actor's gender:

```
(add_cum=POSITION)
```

Position is where you would like the cum to be placed on the actor. The allowed values are Vaginal, Oral, Anal, VaginalOral, VaginalAnal, OralAnal and VaginalOralAnal.

To give an example:

```
actor1=Female(add_cum=Vaginal),
```

This tells SexLab to apply the cum to the vaginal area in the final stage.

It is not currently known if the add_cum parameter can be added to other stages other than the last stage at this time.

OPEN MOUTH

In stages which features oral sex, the actor giving may require to have their mouth open on certain stages. We would use the open_mouth=True parameter. To do this, we need to add a new line of code underneath the actor2 line.

```
a1_stage_params = [  
    Stage(1, open_mouth=True),  
],
```

a1_stage_params tells the game that for actor1, certain parameters are about to be set. The next lines then set which stage have these additional parameters. In the above case, Stage 1 is going to require the actor to have their mouth open.

SILENT ACTOR

This works very much like the open mouth parameter. This one says in the actor is silenced, such as no moaning during a stage. We define this using the silent=True parameter.

```
a1_stage_params = [  
    Stage(5, silent=True),  
],
```

The above is telling the game that a1 is silent for stage 5 only.

SCHLONGS

The SOS parameter allows you to amend the angle of the schlong when using Schlongs Of Skyrim. The default is generally set to 3 but it can be anywhere between 0 (lowest, pointing towards the ground) to 9 (highest, pointing up to the sky). Most animations do not need the schlong to be aligned but if they do, the SOS=X parameter to use, where X is a number between 0 and 9.

```
A2_stage_params = [  
    Stage(3, SOS=5),  
],
```

The above is telling the game that the schlong of the second actor (if applicable) needs to be set at 5.

ANIM OBJECTS

Anim objects is an actor parameter. These are in game objects, items and furniture that get attached to an actor to support a scene. They are added much like the add cum parameter but instead uses the object parameter.

```
actor2=Male(object="AOZaZPunishmentPillory"),
```

This is telling the game that a Punishment Pillory is needed for Actor 2 in all the scenes.

It is unknown how the mod currently handles animations objects that swap between actors or are not required for all of the scenes. This is currently being questioned.

POSITIONING

Positioning an animation isn't always required these days and most animators try and make sure they work from their central point, their zero. But if users want to adjust the positions or animator's may have actors out of alignment, SexLab comes with the ability to amend the alignment. These are handled by adding position parameters. The values are forward, up, side and rotate.

Take note that 0 is the central or base position. If you want to align the actor up 5 steps from the base position, you would use up=5. If you want to move them down 5 steps from the base position you would use negatives, so it would be up=-5. Use this logic for the other position parameters.

```
A2_stage_params = [  
    Stage(3, up=5, forward=-15, rotate=-180),  
],
```

The above code tells the game where to position the actor from the central point of the animation.

CREATURES

When an animation features a creature as one of the actors we need to define that it is a creature and what race the creature is. In a majority of cases the creature is the second actor so we would add the code as CreatureMale(race="INSERTTRACE"). If we use the Draugr as an example, this is what we would write:

```
actor2=CreatureMale(race="Draugrs"),
```

This is telling the game that the second actor is a male creature and the race is a draugr.

To find all the codes for creatures please refer to the Skyrim wiki found online.

GENERATING THE JSON AND FNIS FILE

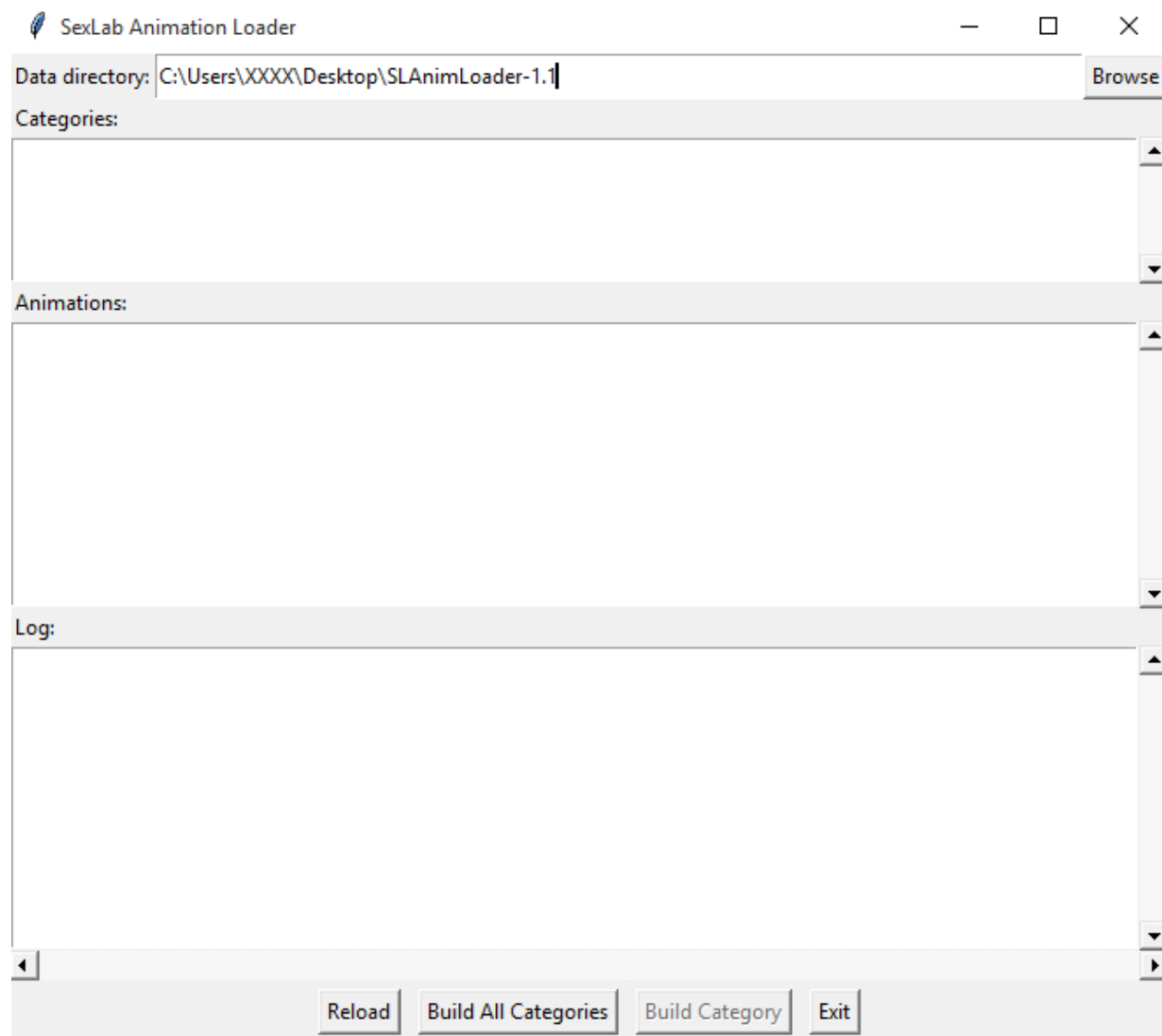
Once you have completed the source file you are ready to generate the necessary files for the mod. This is where the magic happens.

Navigate to the SLAnims folder and locate the SLAnimsGenerate.pyw file and run the file.

Ensure you have python 3.x installed at this point.

Name	Date modified	Type	Size
source	13/12/2015 22:17	File folder	
SLAnimGenerate.pyw	13/12/2015 17:42	Python File (no co...	49 KB

You will be presented with the below screen:



Now, if your categories box is blank, then you have left `is_example` as true. Make sure it is set to false and save your file and click Build Categories at the bottom.

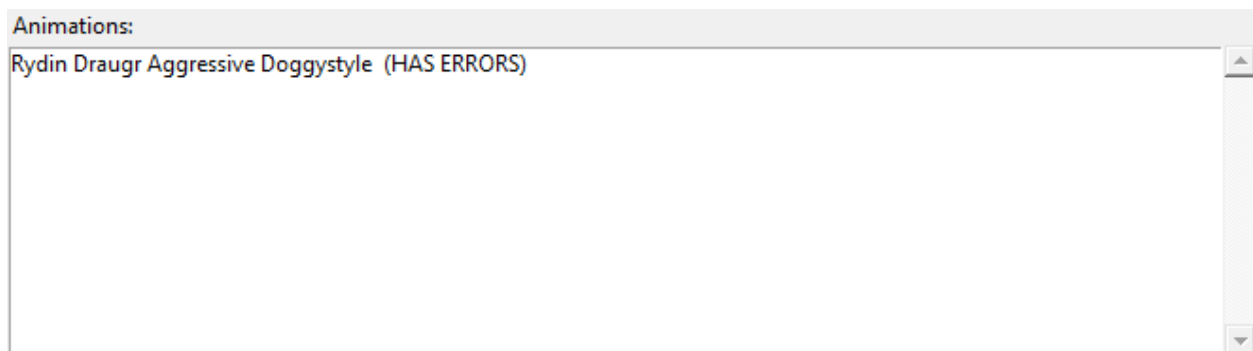
What you should see in the categories box is your text files in the source folder, whatever you have named them. Here is an example:



When you select your pack from the list, the Animations box should populate with a list of the animations from your source file, using the `anim_name_prefix` + name from your source file.



Now, if you have any errors at all anywhere in your setup, it will alert you to this with the following message:



Clicking on the line will tell you what the error is and should point you to what the issue is. Errors will be covered in a later section in the guide.

To create the JSON and FNIS files, simply click Build Categories (or all if you have more than one pack). This will generate the required files for the mod and FNIS. You will get the details in a log out put at the bottom.

```
Log:
=== Build Logs ===
Rydin: JSON already up-to-date
Rydin: FNIS lists already up-to-date
=== Category Info: Rydin ===
JSON output up-to-date: SLAnims\json\Rydin.json
All FNIS lists up-to-date:
- Up-to-date: meshes\actors\character\animations\RydinAnimationPack\FNIS_Rydin
- Up-to-date: meshes\actors\draugr\animations\RydinAnimationPack\FNIS_RydinAni
```

If you haven't already, you can now pack your files into a compressed file (zip or rar, etc) and add them to your data directory, recommended by using a mod manager of your choice. Once you have the files in your data folder there is one final step.

CREATING THE FNIS BEHAVIOR FILES

You are almost finish in completing your pack. The last and final step is to generate the behavior files FNIS needs to complete the process.

With your files in your data folder (or virtual data folder), locate Generate FNIS for Modders and run the program. Navigate to where your animations directory is and you should find a text file called FNIS_[packnamehere]_list.txt which is where all the code for your animation is. Load this up and the generator will create the appropriate behavior file in meshes/actors/character/behaviors called FNIS_[packnamehere]_Behavior.HKX

Once this has been done, you are left to just run Generate FNIS for Users to add your files to the game.

ERROR MESSAGES

all actors must have the same number of animation stages:

This error happens when you do not have the same amount of stages for each actor. If you have two actors, there should be one stage per actor, for three actors there is three stages per actor and so forth. Even if one actor uses the same animations for multiple stages, you need one file per stage for each actor.

no animation files found:

This error occurs when you have the animation anim_id_prefix and id does not match the name of the HKX file. For example, if your anim_id_prefix is "SLAA_" and your animation id is "Passionate_Kissing" then your HKX files should be SLAA_Passionate_Kissing_Ax_Sx.hkx. Any deviation will mean the generator will not find the files. You should also check your animations are in the correct folder and your anim_dir is set to the correct folder name. Again any deviations will cause the files not to be found.

If you received any more error messages, please copy and paste the message from the generator or take a screenshot and post it to the support thread.

MAKING ANIATIONS PACKS TO SHARE

If you wish to share the animation pack you have just made, then all you need to do is put the files together in a compressed file. The best thing to do is take your original template that you made and add the behavior file that you created into the appropriate folder we made at the start of the guide. Compress this and upload it and you are ready to share.

When sharing packs, you should consider for housekeeping to change the version number in the generated FNIS file. If you go to the FNIS_[packnamehere].list.txt inside the animations folder, at the very top it will say 'Version' followed by a number. It is recommended you change this number manually each time you release a new pack to the public, such as version 1.1 to version 1.2 for example for minor updates. Change it to version 2.0 when you make some major changes. This is just a guide but you do not need to follow the above logic. You will need to do this before you run Generate FNIS for Modders or the changes will not be reflected in the behavior HKX file.

Remember to give credit to the animators. It's great that you can make your own packs with your own amendments but remember that without the animators, you wouldn't make the pack. Give credit to those people when sharing packs and do not claim as your own.

You should also consider that your personal taste may not cater for everyone. Warnings or information in what the pack contains and any specific information to help the user downloading know what they are installing is a good idea too.