KK Modding 101

Wogrim's Epic Overview of How to Make All Types of Mods

Basic Terminology and Information

Mods are anything that adds a "new item" (term used very loosely) to the game. These are types of things that already exist in the game that people want more of for more variety, or people want to create something specific but don't have the perfect option. It can be a very simple texture item like a new nose option, or some fancy 3D animated tentacle.

A **Port** is an item that is a copy of an item from a different game. As a modder this means some of the work has been done for you because you are not making the item from scratch, but you often have to make some modifications to make it work properly. Clothes ports are still a lot of work for reasons that will be explained later.

Plugins are programs which change how the game works; for the purposes of this guide we will not call these mods. If a plugin creates a new type of item for the game, we will call it a mod if you make an item of this type, such as a new body for the **Uncensor Selector** plugin. Such mods generally need an extra component for the plugin to read them. Plugins are awesome and give us more options of what we can do with mods. All the plugins I mention in this guide come with HF Patch so if you have that you're good to go (and if your mod depends on one of these plugins, hopefully your users already have them).

Hard Mods are mods that directly change the game data, which is usually considered a bad idea because it is easy to break your game install and you sort of can't share them with other people.

Sideloader Mods are mods that the **Sideloader** plugin reads from the "mods" folder in your game install folder. These are usually saved with a **.zipmod** file extension, so they are also called **Zipmods**. Every mod you make should be this type. Your zipmod will not work if the file structure is not correct.

Unity is the game engine that KK runs on. It is free to download, but if you want to use it for modding KK you need version 5.6.2f1. You can make many types of mods without it. There is a Unity project called **Koikatsu Modding Tools** which does a few extra things to help put a mod together, but if you do not understand the structure your mod should be in it will be hard to troubleshoot any problems you have.

Asset Bundle (shortened as **AB**) is the name of a type of Unity game file which has **.unity3d** file extension. Pretty much all data for the game's items are in ABs, and pretty much all data for most mod types goes in ABs. So most zipmods have at least one AB. ABs have **a CAB string** that must be different from all others in the game and in other mods.

SB3UGS (also called **SB3U**, **SB3Utility**, and probably other names) is software that lets you look at and modify ABs. To make a mod, you can have Unity build a new AB with all your item data in it, or you can copy an existing AB and use SB3UGS to modify it. Even if you build your AB with Unity, SB3UGS is VERY useful for looking at game items and other mods to learn how they work, and for troubleshooting your mod if it doesn't work.

The **Manifest** is a required file in a zipmod which contains metadata about your zipmod. It must be called **manifest.xml** and must be in the root of your zipmod and must have a **GUID** that does not conflict with any other mods.

A **List File** is a file in a zipmod with **.csv** format which tells the game what items are in the mod. The game has list files for its own items, but they are inside of ABs. List files have different formats for each type of item, so a mod with multiple types of items will have multiple list files. The game will be confused and appear to be broken if a list file is formatted wrong, so check it carefully when you make a mod. You can make a list file with a regular text editor, but I recommend a table editor to make sure columns line up properly.

Maker is a commonly used name for the game's character creation system, which is where you go to initially test many types of mods. **Saturation Filter** is an effect that gets applied to the camera which generally darkens and oversaturates colors so your items can look different from what you expected. There is a plugin that can turn it off, but it is recommended to leave it on and color your items accordingly.

Free H is another name for the **Have Sex** mode, which is where you would test less common types of mods such as a new sex position or a new map. Free H has different saturation filters for Day, Sunset, and Night, so you may consider how your items look with those.

Adventure Mode (also called **Story Mode**) is the **Main Game** where you play a guy who gets transferred to an all girls school. Some item types are supposed to have a low-detail version for game performance(called **_low** version) when walking around the school (**Roaming**). But there is a plugin many people use called **Force High Poly** which means the low-detail version isn't used. You can look at game files for that item type to see if you are supposed to have a _low version.

Studio (or **Chara-Studio**) is the system where you set up scenes with characters and items to take good screenshots or make a movie, which launches as a separate program from the rest of the game. Some items can be made so they are **Poseable** in Studio with the **Forward Kinematics** (**FK**) controls. There is also a plugin called **KKPE** that lets you pose things you can't pose with FK, but I find it less user-friendly.

A **Shader** is code which tells your graphics card how to draw an item (given textures and other variables). A shader goes on an item's **Material**, but the material basically does nothing except hold the shader so these terms often talk about the same thing. In general though, the word Material is used when talking about a specific item. There are several shaders in the game for different item types to support different features. You can change most of the textures and variables on an item's material in Maker or Studio with a plugin called **Material Editor** (shortened as **ME**) to see how the shaders work and to decide what settings you want for your item.

A **MonoBehaviour** (shortened as **MB**) is a script in Unity that can be put on items. Each type of 3D item needs a MB corresponding to that item type; the game needs the MB for information about showing your item. You can also put custom MBs on an item, to give some other functionality.

Layers are a Unity feature that allows things in the game to exist in basically different dimensions, which can be set to interact or not interact with each other in various ways. In the case of KK, characters (and items attached to them) are on the **Chara** layer (**A** in SB3U) and the world is on the **Map** layer (**B** in SB3U). This separation means the character lighting (directional light that is normally pointing in the same direction of the camera) does not light the world, and any map lights (such as ones placed in Studio) do not light the character. So any 3D items you make need to be in the right layer.

The Main Item Types

Texture Items includes several item types such as nipples, tan lines, eyes, nose, lipstick. These are pretty easy to make if you just take a look at how the game's items work, but may need a little effort to get them the exact look you want. It technically is getting put on a 3D item, but you do not have control over that part.

Accessories are attached to the character at a chosen location so that they can move with the character. Simple accessories are pretty easy to make, animated ones tough. There is a bit of required formatting for the item's structure so that it can be moved and resized with the game's controls. Accessories must have the ChaAccessoryComponent MB.

Studio Items are items exclusive to Studio which players can place in their scenes. Studio items have a messy list file system for putting items in a specific folder. There is a plugin called **QuickAccessBox** which gives a search option, and it lets you give studio items a thumbnail icon (which they do not have in the normal interface). There is special formatting for this icon and it does not get specified in the list file like for most items. Studio items are similar to accessories in that you can make them simple or difficult. Studio items generally must have the ItemComponent MB.

Hair is a very important category which takes some work to do properly (reasons later). If made as an accessory it can be moved around, which is great for things like hair spikes and random side ponytails, but please don't make me move around the main hair piece. Hair must have the ChaCustomHairComponent MB. For an accessory to be treated as hair you need the **KK_HairAccessoryCustomizer** plugin and it must have both the accessory and hair MBs.

Clothes are the most work to make properly. Shoes and socks are medium difficulty, everything else is an epic quest. There are some shortcuts you can take though, and you may decide to ignore clothes features you don't care about. Clothes normally must have the ChaClothesComponent. Clothes can sort of be made as an accessory with the **KK_AccessoryClothes** plugin and the **ChaAccessoryClothes** MB, which allows an accessory to move with the body like clothes do (instead of only being attached at one point), but in all other ways it is treated as an accessory, not clothes.

Texture Mods

Texture Mods are mods that only have new textures, which are mostly easy to make. These can be Texture Items, or changes to textures on any 3D item.

To make a texture mod for a 3D item, just change textures (and other shader variables if you want). The **UV Mapping** (often just called **UVs**) is what defines which part of the texture goes to which part of the item; this information is helpful for deciding where to draw things. Also there is software that lets you do **Texture Painting**, which is when you draw directly on the 3D item. Many of the game underwear and bras are actually just texture changes to the same skin-tight shorts and shirt models; you can make your own the same way. Most of the game shaders for 3D items have the following texture slots available (but not all have to be used):

- **MainTex** is what defines the base color of an item if it does not use the color picker, otherwise it should be white. Transparency can be used to hide parts of the item. On game items, name usually ends with **_t**
- **DetailMask** lets you paint highlights, shadows, and lines on an item, with the shadows and lines reacting to the shadow and outline game settings. On game items, name usually ends with **_md**
- **LineMask** basically gives the same options as DetailMask but they work slightly different, and isn't used on many items. On game items, name usually ends with **_line**
- **ColorMask** lets you set areas of an item as being colorable with the color picker with red, green, and blue channels. On game items, name usually ends with **_mc**
- **NormalMap** changes the angle at which light is calculated as hitting the item, which can be used to make small details look more 3D. It is usually calculated by software, but the game's shaders need it in a special format. On game items, name usually ends with **_n**
- **AnotherRamp** is usually seen on items that have a part that is supposed to be metal; it interacts with the texture highlight to give less shine at different angles so it glints when moving (fake reflective look). It doesn't use the UVs like other textures. On game items, name usually starts with **Ramp**

Unskinned Item Mods

A 3D **Model** or **Mesh** is the physical structure (usually made of many triangles) that generally defines the shape of an item (shaders can change the shape). We call the mesh **Unskinned** if it does not move relative to itself. So an item like a sword is an unskinned accessory which moves when the character moves, but the sword itself does not bend or anything. Many accessories and studio items are unskinned.

Modeling Software such as **Blender** (free) or **Maya** (industry standard) is used to make these items; they can take some hours to learn, but once you have experience it is just two extra steps from creating a texture mod for a 3D item: one step is creating the mesh, and the other is called **UV Unwrapping** which is how you define the UV mapping. If you need to learn how to use modeling software, there are many video tutorials on YouTube.

A port of an unskinned item is relatively easy, but expect to make texture changes. You usually will want to change MainTex and create a ColorMask for it to be colorable, you may create a DetailMask if you want related features, and you will probably have to convert the NormalMap if it had one.

Skinned Item Mods

A **Skinned** mesh is one that is attached to an **Armature** (also called **Skeleton**) so that when the **Bones** move, corresponding parts of the item move with it. The character's body is a skinned mesh because parts move according to regular animations, dynamic animations (boob jiggle), and it changes shape with sliders in Maker. An item must be skinned for any of these partial movements**. Rigging** is the process of building the skeleton, and **Skinning** is the process of attaching the mesh to the skeleton (both are done in modeling software). These can be done before or after making textures for the item.

There are a few techniques for skinning. The manual way is to "hand-paint" (called **Weight Painting**) how much each part of the mesh is affected by each bone (**called Bone Weight**). This is pretty easy for robotic things where each piece is only affected by one bone, but can take many hours for parts affected by multiple bones (Unity max is 4). There is an **Automatic Bone Weight** calculation which can get you pretty good results if you're making your own skeleton, but it isn't very good for clothes. The ideal method for clothes is to **Copy Bone Weights** from an already-existing item of similar shape or from the character body itself, so that the item moves well with the character skin. You will probably still see some clipping in animations that needs to be fixed with weight painting; bring body mesh/skeleton and some animations into your modeling software for quick testing.

Hair (and Accessories that function as hair) are usually skinned so that they aren't stiff as a rock. To get the hair to move the item must also have a special MB called **DynamicBone** which does the physics calculations on whichever bones you tell it. All hair must be UV mapped a particular way to work with the hair gloss system, and the color mask should be set up for people to be able to change hair root and tip colors; look at some game hairs for examples. To properly place non-accessory hair, you have to do a special head relocation before you bring it into your modeling software to make sure your hair lines up with it. Non-accessory hair can be posed with FK if you use specific bone names that the game hairs use.

Clothes items are all skinned to the character skeleton, so you don't normally do any rigging for them. But there is a plugin called **ModBoneImplantor** which allows you to add bones if want some kind of dangling piece of clothing to move around with extra bones that you rig yourself. This requires a special MB called **BoneImplantProcess** be put on your item, plus DynamicBone to give it physics. These added bones cannot be posed with FK in Studio.

Half States (also called **Shift** and **Hang** States) is a feature unique to some clothing slots. These are a separate mesh that gets shown when undressing a character. The meshes must be put in specific places within the item for the game to be able to find them and deactivate properly. Half states often share the same textures as the full state but just have a different mesh. To create a half state for your own item, it is the least work if you get the full state 100% completed first (including skinning) and then copy it and edit for a half state.

Some clothes slots and the body receive an **AlphaMask** texture from the list file of the item in the layer above them, which hides parts depending on the other item's state to prevent some clipping. So clothes that receive an AlphaMask need the UVs unwrapped like the game's clothes for this to work properly. These masks in the list files are called OverBraMask, OverInnerMask, and OverBodyMask depending on which layer they hide. Appropriate clothes slots should also have textures for **Liquids** (semen from sex stuff), but you can use the default game ones if the UVs are similar to game items.

Studio Items are sometimes skinned so that they can be posed with FK. Such an item needs an extra list file called the **ItemBoneList** to tell the game which bones to allow FK on.

Other Types of Mods

A **Clothes Item Slot Move** is a mod that puts an existing clothes item into a different slot, so that it can be worn at the same time as another item in its previous slot (have no fear, it is also still available in its previous slot). This does not look very good if the items clip through eachother a lot, it does no good if your character is already using all clothes slots, and there is an issue with clothing half states which is outside the scope of this guide. BUT in the cases where you can do it there is no easier mod to make because your list file entry just points to an already-existing item so your mod doesn't even need an AB. There is a way to fix some cases of the clothing half states problem, but it will require you to make a copy of the existing item and make small modifications.

A **Clothes Item Merge** is a mod that merges two clothes items together, which you would do for similar reasons to a Clothes Item Slot Move. This can sometimes be done with just a little work in the list file and AB to tell the game to draw the meshes from both items at the same time, but you are limited to 2 color-picked textures, and a few items already use both. You can get around this problem by combining the meshes and textures into one side-by-side piece, but the UVs will have to be moved to compensate so it's kind of a pain. And you can't color pick them separately. And you can't half state / hide the items separately which may break your waifu-undressing immersion.

A **Self-Animated** item is a skinned item that has a built-in animation. These are things like wings or tails that move in a fixed pattern, or maybe just something that constantly spins. You use Unity to make sure the animation is built into it, but you can create the animation itself in modeling software.

A **Studio Filter**(also called **Scene Effect**) uses a special image called a LUT to do a fast (but low quality) color correction to what the camera sees; this is the same method used by the Saturation Filter mentioned earlier. It is easy to make once you understand how to make a LUT. KK Modding Tools has an example.

Character Poses and **H-Scene Animations** (sex positions) can also be created. I do not know the specifics of how to create them, but they are probably easier than skinned items once you get past the learning curve and have IKs set up for moving the character bodies. Omega has a guide on this, which also contains a Blender project already set up to make animating easy.

A **Map Mod** adds a new map to the game, for use in Studio or in Free H. It is more work to set up sex locations (HPoints) but KK Modding Tools has an example to look at. Map mods need to be created in Unity.

A **Body Mod** is when you create a new body to be selected with Uncensor Selector plugin. I don't know how it is made (apart from tweaking the game body), but I can tell you some reasons why they generally do not make big changes to the body:

- The game's animations will be messed up if you change the rigging
- Clothes will no longer fit if you make significant changes to the mesh
- The ways people would want to change the body are easier as texture mods and accessories

A **Head Mod** has pretty much the same problems as a body mod, but instead of clothes problems it's face parts. There are a couple good ones that change the teeth and one that removes human ears so you can comfortably put on accessory ears. KK Modding Tools comes with an example to look at.

A **Shader Mod** is a mod that puts a new shader in the game for use in ME; information about your shader is put in a specially-formatted part of the manifest. This allows people to change an item to your shader and change its properties with ME. Also, other modders may copy the shader to create items with your shader already on it. It is easy to make an existing shader into a shader mod, more time consuming if you want to learn to create your own shader, and VERY time consuming if you want to create a shader that works with important game features like the color picker, or shadows and outline that react to the game's graphics settings.