UNOFFICIAL STARBOUND MODDING EBOOK 2.0

FOR VERSION: STARBOUND 1.0

EDITOR'S NOTE

Welcome to Version 2.0 of the Unofficial Starbound Modding Ebook..

INTRODUCTION:

Welcome!, some of you may be apprehensive at first thinking it might be too difficult to learn modding because you have no experience in coding or that you are either too young or too old to start. You should dismiss such notions now, as modding in Starbound is not only easy it requires absolutely no prior experience in coding what so ever.

A large majority of Starbound modding deals with simply defining values to parameters.

```
{
      "name" : "example",
      "price": 11
}
```

As you can see from the example above all which is being done is giving a value to each parameter. You can create 80-90% of all items in Starbound through such simple steps.

For the rest of you who has some experience in coding, Starbound also supports Lua scripting in items, weapons, NPC's and objects. It also has a Script Canvas API allowing you to create custom GUI interfaces and even more complex interactions.

One important thing to note is, this E-book is nowhere near exhaustive or complete. I am simply writing it since I enjoy sharing knowledge with the community. I write this on my free time and updates may be sparse and long between. But if you follow the mod page on alerts, you will be made aware when it is updated. I encourage other users to write their own variations of tutorials as multiple perspectives is always better than one.

On a final note the **reference** at the end of the book is there to help provide a little more detail on various aspects covered in the tutorials.

IT IS STRONGLY ADVISED YOU MANUALLY TYPE THE CODE AND NOT COPY AND PASTE.

`

CHANGELOG

Changelog version 2.2

- Fixed Orientations Parameter on page 22
- Added more details to anchors on page 23.
- Added \ Rewrote - Good Modding Practices.
- Added Starbound Community Tools to Useful Links
- Added more Lua Tutorials in Useful Links section.
- Added another question to F.A.Q
- Added Reference Section on Recipes and Learning Recipes
- Added new Tutorial – Armors and Helms.
- Fixed spelling of File name.
- Added crafting table to reference.

Changelog version 2.1

- Fixed a few grammatical issues
- Corrected admin commands on page 13
- Corrected .metdata -> should have been .metadata (forgot a letter). On page 11
- Removed Asperite as it is no longer free. Added a few more other options.
- Added Useful links section to the end of the book.
- Fixed version numbering to be less confusing
- Added correct visibility parameters
- Added Special Thanks section to end of Book for people who contributed in some way..

`

CONTENTS

`

`

FREQUENTLY ASKED QUESTIONS

DOES STARBOUND COME WITH ANY MODDING TOOLS?

Technically no, but Starbound does come with an asset unpacker and packer. These two tools allow you to unpack the vanilla assets and the packer allows you to pack your creation into a .pak file for quick and easy distribution. Chuckle Fish has released a modified Tiled Editor to allow you to quickly and easily create your own custom dungeons. Though you cannot use it to edit existing planet files.

You can get more information about it here: http://starbounder.org/Modding:Tiled

IF THERE ARE NO MODDING TOOLS, HOW DO I MAKE A MOD?

Do not worry, that is what this book is for.

DO I NEED PRIOR PROGRAMMING EXPERIENCE TO MAKE ANY STARBOUND MODS?

Starbound requires absolutely no prior programming experience to make majority of the mods. Without knowing any programming you can make things such as:

> Decorations, Armors, Weapons, Races, Modify Existing Status Effects, New Planets, Dungeons, Ingame Structures, and much more.

SO WHAT PROGRAMMING LANGUAGE DOES STARBOUND USE?

Starbound uses 2 languages for modding. For basic modding it uses JSON (Javascript Oriented Notation) which was shown to you previously. The other is Lua which allows you to have more advanced control of how objects function.

HOW DO I BEST UTILIZE THIS BOOK?

The best way to learn from this book is start from the first tutorial and continue on. The first two tutorials are very vital to learn the basics (First Mod \ First Object). The tutorials following those will not cover topics already covered. It is strongly advised you make the example mod alongside reading it. This will give you the confidence to make your own version of the object and item.

`

TOOLS OF THE TRADE

Before we start modding, there are few things we will need to do, the first of which is gathering the tool required to make basic mods.

Currently and in the foreseeable future Starbound does not actually offer any tools besides a packer and unpacker tool for the assets. Though there is an optional 3rd party tool download for Dungeon Creation. But the primary tools required for modding the game need be acquired elsewhere, all of which are free and available for all platforms. Though it may not be the same tool for each platform.

CODE EDITOR

When it comes to a code editor there are a few basic requirements;

- The editor should stick to UTF-8 encoding format. You will want to avoid Word Document Processors such as Microsoft Word. These use special text characters not supported by the game.

Believe it or not that actually is the only requirement when it comes to a code editor. Any other features you desire is really up to you. A few recommendations though are;

Windows:

- Notepad ++ : https://notepad-plus-plus.org/
  - Notepad++ probably has the lowest learning curve and easiest to operate for new users.
- Sublime Text : http://www.sublimetext.com/2
  - Rich feature set. High learning Curve. Tons of plugins. Evaluation software with no time limit.
- Atom : https://atom.io
  - Atom is an open source completely hackable editor with many features similar to sublime.

Mac:

- Sublime Text: https://www.sublimetext.com/2
  - Rich feature set. High learning Curve. Tons of plugins. Evaluation software with no time limit.
- Atom : https://atom.io
  - Atom is an open source completely hackable editor with many features similar to sublime.

Linux:

- Sublime Text: https://www.sublimetext.com/2

`

    - Rich feature set. High learning Curve. Tons of plugins. Evaluation software with no time limit.
- Atom : https://atom.io
    - Atom is an open source completely hackable editor with many features similar to sublime.

If you don't wish to download any software you are free to use something such as Microsoft Notepad which comes with windows. But you will be missing out on great features provided by the programs listed above.

PIXEL EDITOR

The next tool we need in our shed is a Pixel Editing tool though there are quite a few graphics editors out there we have a few requirements we need to full fill first;

- Must be able to export PNG files
- Must support Transparency Layers
- Must be able to do pixel based editing.

Check the Useful Links at the end of the Book for tutorials on using the tools for Pixel Art.

Again like pixel editors any added features are just benefits. A few good options are;

- GrapfX 2 - http://pulkomandy.tk/projects/GrafX2
    - A basic pixel editor available for Windows, Mac and Linux. Though not well documented and few tutorials exist. Good for advanced users who are confident in learning the ropes themselves.
- Gimp: http://www.gimp.org
    - Gimp is a Graphics Manipulation Package which is a general purpose tool with an extremely rich feature set and thousands of tutorials online. Though for more general users this may feel very overwhelming at first.
- Paint.Net http://www.getpaint.net/index.html
    - I personally have not used it, but a few users have recommended it from the community.

`

UNPACKING YOUR ASSETS AND PACKING YOUR MOD

Before you start modding you have to unpack the assets from the vanilla container. This process does not "remove the files" but simply makes a copy of them for you to look at and learn from. Any changes to the unpacked assets will not make changes to the game until you make a mod of it. Please do not "repack" the assets to replace the vanilla assets file as you will be damaging your game.

Now let us look at the various ways we can unpack the assets.

Video Tutorial: https://youtu.be/Dsjz2IrArVc

UNPACKING THROUGH COMMANDLINE

Please bear in mind the code should be on a single line or it will not work.

The syntax to unpack assets is;

"Location of Unpacker" "Location of pak file" "Location to unpack"

Example:

"C:\Program Files (x86)\Steam\steamapps\common\Starbound\win32\asset_unpacker.exe" "C:\Program Files (x86)\Steam\steamapps\common\Starbound\assets\packed.pak" "C:\Program Files (x86)\Steam\steamapps\common\Starbound\unpacked"

The syntax to pack an asset is;

"<Location of packer>" "<location of folder to pack>" "<location to unpack>\filename.pak"

Example:

"C:\Program Files (x86)\Steam\steamapps\common\Starbound\win32\asset_packer.exe" "C:\Program Files (x86)\Steam\steamapps\common\Starbound\mods\test" "C:\Program Files (x86)\Steam\steamapps\common\Starbound\mods\test.pak"

For users who feel uncomfortable with this you may find some tools on the forum here:

http://community.playstarbound.com/forums/modding.111/?prefix_id=66

`

PACKING AND UNPACKING TOOLS

WINDOWS

- GUI Front End - http://community.playstarbound.com/threads/updated-asset-packaging-unpackaging-gui-frontend-for-1-0.95468/
- Mod Pack Helper - http://community.playstarbound.com/threads/all-versions-win-linux-modpackhelper.92473/

LINUX

- Mod Pack Helper - http://community.playstarbound.com/threads/all-versions-win-linux-modpackhelper.92473/

MISC

- StarFuse (OSX \ Linux) - http://community.playstarbound.com/threads/alpha-0-4-0-starfuse-pak-utility-for-linux-os-x.115082/
    - Tool mounts pak files – and lets you access the files – so you can easily see how mods work without having to unpack it.

`

BASICS OF JSON

Understanding JSON is fairly straight forward. The quickest way of course is to take the time and look through all the vanilla files and see how they are made. There are 3 basic rules to follow to make sure you don't make any mistakes when writing the syntax.

1. **Always add a comma to separate each parameter except the last parameter before the end of a bracket.**

Let us look at an example,

```
{
  "name" : "example 1",   ← comma
  "price": 10, ← comma
  "shortdescription" : "example 2" ← last line before end of bracket
}
```

After looking at the example you should easily be able to tell when to use a comma and when not to use at this point.

2. **Double quotes are only used for text.**

If you look at the previous example you will notice things like "name", "price" and "short description" are surrounded by double quotes, as well as "example 1 and 2". Parameters always will be in double quotes while their values if it is a text field will require them also.

There are certain values that cannot have quotes; Numbers and True or False values.

3. **Most importantly close brackets in the correct order but in the correct location.**

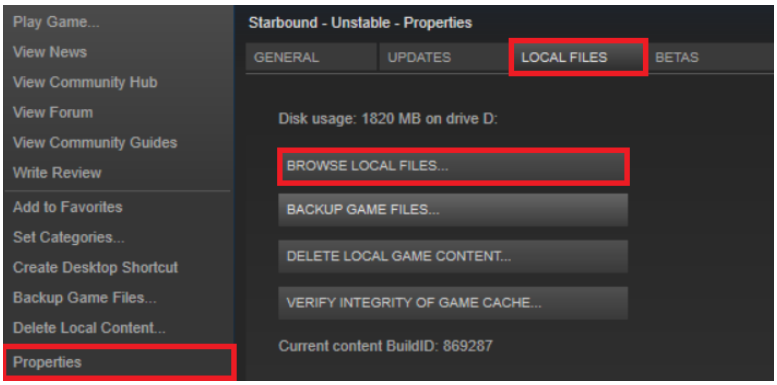
One of the most common mistakes new users make is either they forget to close a bracket, they close a bracket in the wrong order, or they choose a completely different bracket.

YOUR FIRST MOD

INTRODUCTION

By now you should have unpacked your assets, downloaded the required tools, and have a basic understanding of JSON. If you have missed out on any of those things, please go back to the appropriate section and complete it before moving on.

Right click on Starbound in Steam Library → Properties → Local Files (tab) → Browse Local Files.



At this point it should have opened your Starbound directory inside there you will see a mods folder. Open that and create a new folder and name it firstMod.

A FEW SMALL POINTS ON NAMING

The way you name your files and objects is important for 2 major reasons.

1. To easily identify the errors of items associated with your mod found in the starbound.log
2. To prevent incompatibility with other mods who may end up using the same name for an item.

Now in Starbound there are actually 2 names associated with each item. One name is the name displayed to the player most commonly references as the "shortDescription" the other is the name only referenced by the game engine or more commonly known as "itemname" or "objectname".

A good practice for a naming style is using the name of your mod then the item name for example if we made a mod called Black Sheep we could name an item.

"itemname" : "bsheep_wool"

This would clearly tell any modder checking out the starbound.log which mod that item is associated to

SETTING UP YOUR COMPUTER

If you are a Windows user one important thing you need to do before we begin is make sure you can view "Known hidden File Extensions". You will want to see it to be able to know what file types you are dealing with.

http://windows.microsoft.com/en-us/windows/show-hide-file-name-extensions#show-hide-file-name-extensions=windows-7

Also when learning to mod it is best to start with a fresh universe and character as well as no other mods in your mods directory. The reason for this is you want to avoid conflicts with other mods, as well as prevent your universe and character from being corrupted as you experiment.

To do this all we need to do is rename your storage folder to anything you want. Then run the game once, this will cause a new storage folder to be generated for you. Now anytime you want to play with your old universe and files – just rename the new one and rename the old one back to storage.

METADATA FILE

Now we have naming down, let us make our first file. Open your code editor and create a new file in your firstMod folder **_metadata** or **.metadata**.

If you are using windows you might have to do "**.metadata.**" where a period is present before and after the metadata file. Or you can do underscore metadata as listed above.

Now type the following code shown on the below and save.

```
{
  "author" : "Mod author name",
  "description" : " Mod description",
  "friendlyName" : "Name displayed in game",
  "name" : "file name",
  "version" : "1"
}
```

Save and launch the game.

If you did everything properly when you go the mod manager (The gear wheel at the bottom right of the screen) you will see your mod listed . If it is not there double check your work.

`

CREATING OUR FILES

Now that you know your mod is working, we need to take some time to learn a few basics. The first thing is the concept of parallel directories and how to store your files.

When you are modifying a "vanilla" (a file which is already existing in the game) then it must be in the exact same position as the vanilla file in your mod folder.

So at this time open your unpacked assets folder and go to;

```
unpacked\items\tools
```

(Where **unpacked** is the location you unpacked your assets to)

We will be looking specifically at the file flashlight.flashlight

Now to understand a parallel directory we need to make an exact same copy of the directory structure in our mod. If the new folder you made in your mods directory is called firstMod then it should be;

```
Starbound\mods\firstMod\items\tools\flashlight.flashlight
```

It is extremely important to maintain the directory structure when you alter vanilla files or you will get an error claiming a duplicate object is found.

DIRTY EDITS AND PATCHING

One thing many new aspiring modders do is called a Dirty Edit. What this means is you copy the vanilla file into the new mods folder and make the changes directly. This of course will work – but is not the proper way to do things, especially if you are planning on releasing this mod to the community. If you are developing the mod only for personal use – then there is no issue with this technique unless you are planning on using other mods in combination.

Now the problem with Dirty Edits is **incompatibility** with other mods which edit the same file. What will happen is the last mod loaded will make the changes and ignore the changes of the previous mod based on the mod load order, unless of course you do the proper way which is the **Patching System**.

BASICS OF PATCHING

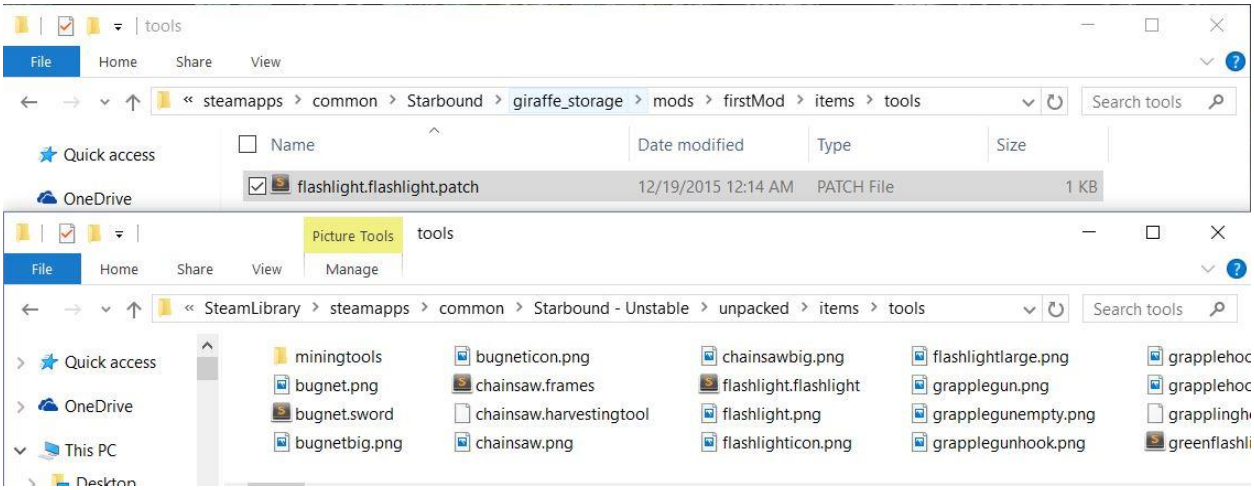
What a patch is essentially is a set of commands which performs a specific operation to a target file to make changes at spots we only wish for those changes to occur. Now if 2 mods change the exact same parameter in the same file – of course there will be incapability and the change will happen again based on the load order. The load order being alphabetical based on the name provided in the metadata file.

But with patching multiple mods can make changes to the same file – as long as they do not edit the same parameter – which is not possible with a dirty edit.

`

Now to learn the basics of patching lets copy (copy not cut) the flashlight.flashlight from the directory listed above into the mod directory as shown previously. Remember it needs to be in the exact parallel folder in order to work.

Now rename the flashlight file in the mod directory to flashlight.flashlight.patch. Anytime you want to make an edit to a vanilla file you must make a 100% sure you end the file name in a .patch. The exception being lua files which cannot be patched.




```
[
 {
 "op": "replace",
 "path" : "/shortdescription",
 "value": "Test Flashlight"
 }, {
 "op": "replace",
 "path" : "/lightColor",
 "value": [100,100,100]
 }
]
```

Now open both the original flashlight.flashlight and the new .patch version we made in the mod folder. In the file in the mod folder delete all the contents inside and place this code instead exactly as you see it.

When in game type;

- /admin
- /spawnitem flashlight 1

The name of the flashlight should now be "test flashlight" and the light will barely be visible.

TESTING IT OUT

You can use this method to edit any existing vanilla file in the game. So one great step into modding is starting to practice editing already existing assets before you jump into making assets of your own.

For some reason if you cannot get it working – download this example mod and compare the one you made to it.

Example File: http://www.mediafire.com/download/43pcrtfi5a6ya2q/firstMod.zip

DISTRIBUTION

With our mod finished we are ready to distribute. If your plan is to distribute it on the main forums or a site like Nexus the best option is to .pak the mod. That will make it easy for users to quickly and easily install it.

If you are planning on putting it on Steam Workshop though you want to keep it unpak'd as the workshop will handle the packing when you upload it.

PAK'ING YOUR MOD.

To turn your mod into a pak file is extremely easy.

Open up your text editor and writ the following code.

```
"Location of asset_packer" "location of folder to pak" "location of
where to save pak file + filename.pak"
```

Though you can't see it properly here, the entire strip of code shown above has to be in a single line. If it word wraps as shown it does not matter – but you may not press "Enter" in any part.

An example:

```
"C:\Install\SteamLibrary\steamapps\common\Starbound\win32\asset_packer.
exe" "C:\Install\SteamLibrary\steamapps\common\Starbound\mods\p_Pets"
"C:\Install\SteamLibrary\steamapps\common\Starbound\mods\p_Pets.pak"
```

Save the file as mymodpaker.bat – make sure it is a batch file and not a text file. The name is not really important but the fact it is a batch file is. Then double click on the batch file to run it. If you did everything properly a new pak file would be made in the destination you have provided.

For Mac and Linux users you can do the similar process through your command line functionality of your operating system.

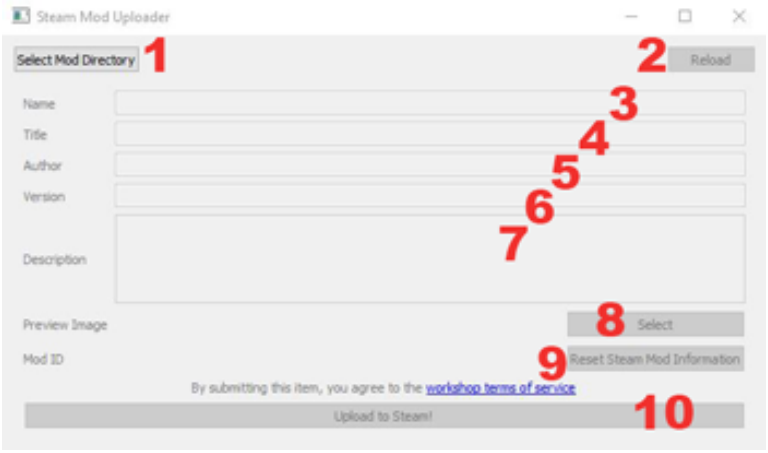
Once you pak the file there is no need to place it in a zip or rar, otherwise it will be less confusing for users.

`

STEAM WORKSHOP

Steam Workshop there are a few extra steps. In this case you do not pak the file, you leave it as a folder with contents. Steam Workshop is only for users who own a copy of Starbound on Steam and not for users who bought it from Humble or GoG.

For Windows users – when you launch Starbound through Steam you will get an option to Launch the mod uploader tool. Choose it.



1. Choose your mod folder and **not** your starbound mods folder. You are selecting the folder your metadata file is in. If you made your metadata file correctly almost all the information will be filled out for you when select it.
2. The Reload button is for you to reload the metadata file if for some reason it is not updating.
3. Name tag is the reference name of the mod which is not shown
4. The Title is the name displayed to users in the Workshop page
5. Author of course is your user name or whatever name you wish to put.
6. Version is the version of your mod – not the version of Starbound your mod supports. It can either be a numerical version or text name.
7. Description is both the description displayed in the game in the mod menu and your Steam Workshop
8. Preview Image is the image displayed on the Steam Workshop as your "Icon". As far as I am aware it can be upto 512 x 512 px and upto 1mb in size. PNG or JPG.
9. Reset Steam Information will actually reset the Mod ID number. This is Mod ID number is EXTREMELY important when it comes to dealing with future updates. The number should always be the same if you wish to update it in the future. The only reason you want to reset it is if you wish to make a Branch Mod for some reason. Note this will change your existing metadata file with the new Mod ID. So you will need to manually retype it in the metadata which will be automatically placed into it after uploading. You can find your previous Mod ID by going to your workshop mod page and looking at the URL. The Mod ID will be the last digits in the URL.
10. Press this button when all the information is correct to upload it.

`

Once your mod is uploaded it will start off as Invisible. Go to the Workshop page of the mod choose visibility from the bottom right and set it to visible once you make any last minute edits.

STEAM WORKSHOP FOR MAC AND LINUX AND WINDOWS

Unfortunately there currently is no GUI tool for Mac and Linux users. So you will have to upload mods to the workshop the old fashioned way through command line. Don't worry it is almost just as simple as the GUI tool and probably a lot faster for some users. **For Mac and Linux use forward slashes where applicable instead of back slashes.** I am writing toward Windows users though the same method is used for all operating systems with the necessary changes.

The first thing you need to do is download SteamCMD:

- Windows: https://steamcdn-a.akamaihd.net/client/installer/steamcmd.zip
- OS X: https://steamcdn-a.akamaihd.net/client/installer/steamcmd_osx.tar.gz
- Linux: https://steamcdn-a.akamaihd.net/client/installer/steamcmd_linux.tar.gz

This is a command line tool which gives you access to most of Steam Services.

Next we need to create a new txt file let us call it s_mymod.vdf. Inside type in the following code,

```
"workshopitem"
{
"appid"          "211820"
"publishedfileid"     "0"
"contentfolder"     "C:\examplemod"
"previewfile"          "C:\example.jpg"
"visibility"          "0"
"title"          "Title"
"description"          "Description"
"changenote"          "Initial Release"
}
```

App ID: Is the ID # of the game. The number provided is for Starbound Stable

Published Field: Is the Mod ID number. Leave it "0" if you never uploaded the mod before. When you upload the mod it will automatically be changed by Steam CMD to the Mod ID #.

Content Folder: Is the folder for your mod

Preview File: Is the image your mod will have in Steam Workshop. 512 x 512 px and less than 1mb. PNG or jpg. The

Visibility – If your mod will be visible on initial upload or not. 0 = visible, 1 = friends only, 2 = hidden

Title: The name displayed to users on steam workshop

`

Description: Description of mod in Steam Workshop

Change note: For updates changelog.

As you may have noticed using Steam CMD gives you a few more features then the GUI uploader.

Once you fill in the code – save the file.

Now run Steam CMD – it will take a few seconds to unpack. If it closes by itself launch it again.

Once Steam CMD window is open and it has finished unpacking type;

***All code you type must be in a single line.**

```
login myLoginName myPassword
```

Once it says you have successfully logged in – it may ask you to authenticate with Steam Guard type in the Steam Guard number sent to you.

Finally to upload the mod type;

```
workshop_build_item c:\example.vdf
```

Hit enter and it should say you are uploading the mod to the Workshop if you did everything right. Finally go to your workshop page and you should be able to see the mod listed. To make updates to the mod use the same VDF file. You can type in change notes for each update – just make sure you keep the new Published Field that Steam CMD will change automatically in the VDF after the first upload.

`

GOOD MODDING PRACTICES

When modding there are some good practices you should keep in mind to help maintain compatibility with other mods as well as help other modders quickly go through issues and identify issues.

- **Naming Objects and Items**
  - When choosing a file name for your Object and Items try to make sure the name is unique. Do not confuse File name for Short Description (The name displayed to users) – I am referring to Object Name an Item Name. You want to pick a name which is not generic so another mod may accidently have it causing a crash. To do this adding a mod tag into the name helps immensely in detecting which mod is causing an issue, and avoiding possible conflicts. For example if your mod was Bob's Adventures and your item was a wrench naming the item ba_wrench would be ideal.
- **Finding all possible choices for Parameters.**
  - One question is often asked, "what are the other parameters for certain variables like "anchors" in Objects?". If you use an advanced code editor such as Notepad ++, Sublime, Atom, etc you have an option called Search Files and Folders the default key for most of them is Control + Shift + F for windows. This function will search all text in all the files in those folders for that value. Then it will display all the results in a new tab. Double clicking on one of the results will open another new tab showing you the file. This is the best way to do 2 things.
    1. One you can easily find almost all vanilla parameters possible through this method.
    2. You can also find all files which call upon that object or item. For example you may want to know how to add a new biome and where all the places the biome is defined. By searching for the name of a vanilla biome – it will list every file which it is linked to. Allowing you quickly to make a map of every file you need to recreate for your own custom mod!.
- When making Tiles and Liquids
  - Tiles \ Materials \ Liquids a unique ID number and if that ID number is used in another mod it can cause it to crash. To fix this issue on Starbounder a page was made for mod authors to designate the numbers in which they are using.
    - http://starbounder.org/Modding:Materials:Mods
    - http://starbounder.org/Modding:Liquids:Mods
  - If you need help with the Wiki Please make a post here - http://community.playstarbound.com/threads/lets-get-wiki-basic-discussion.51896/
- Tutorials and Guides
  - If you learn anything useful from the book – please feel free to write your own guide even if they cover the same topic. The more people who are willing to share with the community – the more people who will become interested in modding. Sharing information and making it available to everyone allows for a much more richer and vibrant modding community.

CREATING A BASIC DECORATIVE OBJECT

At this point you should have made your first successful mod. Now that we learned how to modify existing vanilla files we will look into making completely new ones on our own.

I am not going to cover the basic of creating a mod info file and new folder as we just covered it. So please take the same steps for this new mod as done previously.

So let us begin.

THE SETUP

The first few steps we are going to need to do is create a new folder in our mods folder and create a new metadata file inside of it. In this case let's go with the name firstObject.

Now you can either make your own image and follow along or use an image from vanilla files and try to interpret the changes yourself.

If you wish to try and create your own graphic the requirements are;

- The image editor must export the file as a PNG with a transparency layer.
- Smallest visible block in Starbound is 8 x 8 pixels.
- The player character is about 24 x 32 pixels (though the actual frame is 43 x 43)

Don't feel like you need to create an amazing piece of art as that is what discourages most new modders feeling their work is not good enough. Just make something passable and either you will eventually get better – or someone may help you out in the future. But don't wait till it's perfect or your mod will never be made.

As you can see – even a professional artist such as Jim Davis had a very humble beginning for Garfield.

So do not worry too much about how your first art piece looks like either. Over time it will prove as you refine it and get more skill.

A few tips;

- The object dimensions should be based on a factor of 8.
- If you are making an object such as a table to place things on make sure there are no empty pixels within a factor of 8 above the table. For example a table only 14 pixels high will result in objects in game floating 2 pixels above it.
- You can have objects in other dimensions – but it will result in items not lining up properly time to time. Though you can define custom collision parameters manually but that is an advanced topic. See Outpost Stores or some farmables for an example of how it's done in the vanilla game.

THE PIXEL ART

I am not going to do anything special for the sake of the tutorial. I am going to make a simple square but on purpose I am going to make it 22 pixels high and 24 pixels wide.

The cube I made above is a canvas size of 24 x 24 but 2 pixels on top are left blank. The importance of this will be explained soon as the issues this could cause when it deals specifically with stacking and tables.

Let us put this file in:

Starbound \ Mods \ firstObject \ objects \ cube \ cube.png

Though to be clear since we are making a custom object it is not necessary to place it in the object folder. Since it is a custom object you can follow any folder directory structure you wish, maintaining parallel directory structure is only important when modifying vanilla files. I am simply doing it this way for sake of good practice to follow for beginner users.

FRAMES FILE

The next thing we need to make after the image is the frames file. The frames file basically defines the image for the game. It needs to be the exact same name as the image (without the extension of course). .It tells the game how big the image is – how many frames the image has – if the image is animated, which sequence to play the animation, and what name designations are for frame setups etc. The short story is, it is very important file so do not forget about it.

```
{

  "frameGrid" : {
    "size" : [24, 24],
    "dimensions" : [1, 1],
    "names" : [
      [ "default" ]
    ]
  }
}
```

Size: Is the size of each individual frame, and not the entire image. So even if the entire collection of images is 240 x 240. If each frame is only 24 x 24. The size will remain only 24 x 24.

Dimensions: Is how many frames exist within the image. So if we were dealing with multiple variables of objects or animations – we would list how many columns then rows.

Some examples would be;

Objects \ generic \ box 2	Objects \ generic \ burning coals

`

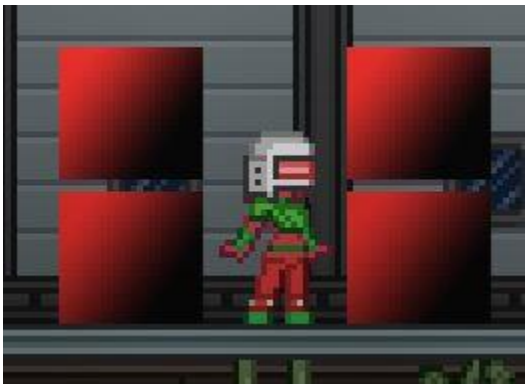
Where you have multiple variations and the other one being animated respectively.

Names: As you can see from the examples also names are important when determining animation as well as variations. You can easily create multiple variables of an object with a single image and only choose one variant to be displayed in the image.

One example is my Purchaseable Pets mod – which has a single image for 7 objects. Each object calls upon a specific variant in the image.

Now save the file as cube.frames

One important point to note is the 8 pixel rule and platform collision.



On the left side you see the cube using the exact frame dimensions of 24 x 22. On the right side you see the object using the frames of 24 x 24 with the top 2 pixels left blank.

You will notice regardless of which one you use when trying to stack the objects on top of each other – there is a 2 pixel space. So when dealing with objects which can stack items on top of it – it is always better to keep to multiples of 8 in your design or you will end up having floating objects. Unless of course that was your intention in the first place.

`

THE OBJECT FILE

The object file is the meat of the object creation. Also reasonably straight forward if you take the time to look at other vanilla objects and see how it is done. Through the object file you can attach scripts (such as the Tech station) or you can turn them into containers or even crafting tables.But for our example project we are going to keep things simple and only make a simple decoration object.

```
{
  "objectName" : "fo_cube",
  "colonyTags" : ["misc"],
  "rarity" : "Common",
  "category" : "decorative",
  "price" : 25,

  "description" : "My first cub",
  "shortdescription" : "Cube",
  "race" : "generic",

  "apexDescription" : "What cube",
  "avianDescription" : "The cube.",
  "floranDescription" : "Cubess",
  "glitchDescription" : "Cube",
  "humanDescription" : "Cube.",
  "hylotlDescription" : "Cube.",
  "novakidDescription" : "A cube.",

  "inventoryIcon" : "cube.png",
  "orientations" : [
    {
      "image" : "cube.png:<color>",
      "imagePosition" : [-8, 0],
      "frames" : 1,
      "animationCycle" : 0.5,

      "spaceScan" : 0.1,
      "anchors" : [ "bottom" ],
      "collision" : "platform"

    }
  ]
}
```

So let us create a new file cube.object and write in this following code.

To get a basic idea of all the parameters.

objectName : Is essentially the name the game uses and is not displayed to the user. It is useful to create a unique name that is easily identifiable in logs so other users know where the object or item came from if an issue arises.

colonyTags – these are unique tags used by colonist system to help determine which colonist is generated. Specific tags are required for specific colonists so it's a good idea to look through the files to determine which tag to use.

Description: Is the object description displayed to the user

shortDescription: Is the name of the object displayed to the user

<race>Description: Is the unique description said by each race when scan tool is used. This parameter also works with custom races, as long as you use <race> parameter exactly as the one provided in the species file of the custom race.

Orientations: orientation is how the image displayed. It is a good idea to look through vanilla files and compare.

`

imagePosition: Is the anchor point of the object when it flips.

spaceScan: is the collision box created for the object. Do not change this value if you want it done automatically. If you want to create a custom collision box use spaces instead. See farmable objects as an example.

anchor: Is how the object attaches to the game world. (top, bottom, background, left, right). When the object is touches a collision on that particular side – it displays that particular image.

collision: optional value if you to be able to stack objects ontop of the object. Otherwise you need not even include this parameter.

RECIPE FILE

The recipe file tells the game how to craft the object and which objects can craft the object. There are of course some limitations for example you can only have one output for the item. The output determines which object or item the recipe is for – not the file name. An important point to keep in mind and one of the reasons why only a single output is allowed.

```
{
  "input" : [
    { "item" : "money",
"count" : 1 }
  ],
  "output" : { "item" :
"fo_cube", "count" : 1 },
  "groups" : [ "plain" ]
}
```

Each "group" is associated with a crafting table filter. Add the groups which you wish – and filters in the crafting table defined by the object file will show that recipe.

The "plain" group is for crafting filter for the main character.

One thing you will notice is the game will chose the highest value in case of recipes. Since I chose a cost of1, it will pick 25 in the game. The value which is listed in the object file.

One final important note is once you unlock a recipe for an item all associated recipes will be unlocked at the same time. So you can't have recipes for one item unlock at different times. To get around it you would need different crafting stations which unlock at different times if you want a variation in recipes.

PLAYER.CONFIG

Once the recipe file is made we have to "learn" the recipe so we can craft it. We can either do it through player.config where they automatically know the recipe or having another item on pick up teach the recipe through learnBlueprintOnPickup.

```
[ {
"op" : "add",
"path" : "/defaultBlueprints/tier1/-",
"value" : { "item" : "fo_cube" }
} ]
```


`

WRAPPING IT UP

If you did everything properly when you launch the game and press "C", your main character should be able to craft the cube without a crafting table.

Example mod: http://www.mediafire.com/download/ogtuk25yfsk665k/firstObject.zip

`

MAKING YOUR OWN ARMOR

As always I am not going to repeat over areas covered in the earlier tutorials. So please do not skip to this area without finishing the previous ones. Making armor is only slightly more difficult than making a custom object. The most complicated part is setting up the assets properly. So if you are confident in using an image editor program you should be fine.

INTRODUCTION

Now before we begin – make sure you setup a new mod folder with the _metdata file as we did previously. I am going to follow the vanilla file structure and follow items \ armors.

Once we have the folder structure set up and metadata file written out. Go to your unpacked directory and go to; \unpacked\items\armors\biome\tar\tar

Because of the way the tar armor set is designed to essentially look like a blank canvas humanoid we can easily use it as a canvas for our own custom armor, but before we jump into image editing we need to modify the JSON files so we do not accidently get duplicate item errors later on. Also as Tar is essentially a cosmetic item – we will need to make some modifications in order to turn it in to an actual armor. With that said – if you just want to make a cosmetic item you can just edit the tar files without adding the extra bits for armor.

So for now the first step is to copy all the tar files over into our new armor mod folder.

SETTING UP THE JSON

Before we start editing the images lets setup the JSON files. Since we are making completely custom items we don't need to worry about patching so we can edit the files directly. First though lets rename only the JSON files (Do not rename the image files – we will discuss why later) to seb_armor as the primary name. Go through the JSON files and make the appropriate changes.

I am going to give an example of the chest file with the modified armor parameters. In order to fit everything on the page – I had to remove some of the; color codes, comments and rework the spacing. You do not need to do these steps.

The part I suggest you do though is remove the effect sources parameters. Effect sources is essentially a "particle effect" that happens when you wear the armor. As we will not be using it, I suggest removing that parameter completely.

The part highlighted in green though is specific to armors and only needed if you are going to make an armor item as opposed to a cosmetic item. In another words – you do not need to add these parameters for cosmetic items.

`

```
{  "itemName" : "tarchest",
  "price" : 2500,
  "inventoryIcon" : "icons.png:chest",
  "maxStack" : 1,
  "rarity" : "Rare",
  "category" : "chestwear",
  "description" : "A sticky shirt made from thick tar.",
  "shortdescription" : "Tar Shirt",
  "tooltipKind" : "armor",
  "maleFrames" : {
    "body" : "chestm.png",
    "backSleeve" : "bsleeve.png",
    "frontSleeve" : "fsleeve.png" },
  "femaleFrames" : {
    "body" : "chestf.png",
    "backSleeve" : "bsleeve.png",
    "frontSleeve" : "fsleeve.png" },
  "colorOptions" : [
    { "ffca8a" : "66538d", "e0975c" : "4b3b6d", "a85636" : "2e1e3b", "6f2919" : "080207" },
    { "ffca8a" : "838383", "e0975c" : "555555", "a85636" : "383838", "6f2919" : "151515" },
    { "ffca8a" : "b5b5b5", "e0975c" : "808080", "a85636" : "555555", "6f2919" : "303030" },
    { "ffca8a" : "e6e6e6", "e0975c" : "b6b6b6", "a85636" : "7b7b7b", "6f2919" : "373737" } ],
  "level" : 4,
  "leveledStatusEffects" : [
    { "levelFunction" : "standardArmorLevelPowerMultiplierMultiplier",
      "stat" : "powerMultiplier",
      "baseMultiplier" : 1.25 },
    { "levelFunction" : "standardArmorLevelProtectionMultiplier",
      "stat" : "protection",
      "amount" : 0.5 },
    { "levelFunction" : "standardArmorLevelMaxEnergyMultiplier",
      "stat" : "maxEnergy",
      "amount" : 5 },
    { "levelFunction" : "standardArmorLevelMaxHealthMultiplier",
      "stat" : "maxHealth",
      "amount" : 5
    } ],
  "itemTags" : [ "tier4armour" ] }
```


`

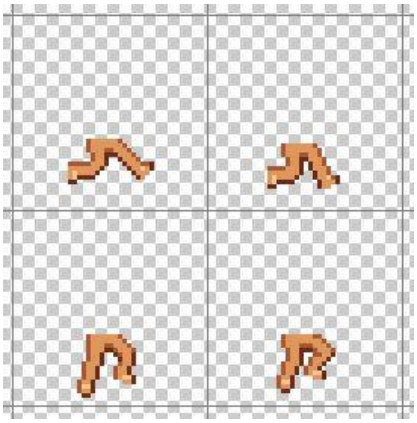
UNDERSTANDING THE PARAMETERS

Now let us take a brief look at each parameter so you better understand what to change and for what to change and how. I am going to skip over the obvious ones unless I feel explanation is necessary.

- Item Name: I cannot stress this enough, Item name is not shown the player, it is a unique ID used by the game. Since it is unique try and add a mod tag or some sort of unique identifier to the name to make it easily identifiable and unique to prevent conflict.
- Short Description: Is the actual name of the item shown to the player. It does not matter if this shares the name with another item. No conflict will occur
- Tool tip kind: This is the interface used when mouse-over the item. You can see other interfaces and parameters in the interface folder.
- Color options: this is one of the topics which is slightly difficult to understand for new users. The color options consists of 2 parameters; "base color":"new color". The code shown are hex colors. The base color is the color of the pixel you want to change in the original image. While the new color is the color that pixel will change into. I will cover this much more in-depth in the in reference part of the book.
- Level: Level is what tier of armor group this item belongs to.
- Leveled Status Effects: Is how the armor modifies the basic values set in player.config as well as weapons.
- itemTags: Not entirely sure, likely used for merchants when auto populating their lists.

SETTING UP THE IMAGE.

Once the data has been set we can now work on the image, which by far is the most complicated part of this endeavor. Now for those users who are using a basic image editing app – you will need to open each image one at a time and edit them. For users with more advanced image editors such as GIMP or Photoshop, or even Paint.Net – you can cheat a little and load all images into a single file. Though you will need to separate them out again before saving.



For the users who are not confident in what they are doing I strongly suggest editing a single image at a time.

IMPORTANT THINGS TO KEEP IN MIND

1. **Each frame size is 43 x 43 pixels.**

No when you open the file – each image will be in its own frame. It is extremely important you stay within each frame. The black lines shown in the image to the left will not be present in the image. I added them to give a better understanding of the amount of space you have to work with.

So if you have an advanced editor such as GIMP – you can set guide lines to help you better draw your image. You can get more information on GIMP's grids and guides here;

https://docs.gimp.org/en/gimp-concepts-image-grid-and-guides.html

Do not worry these guides will not appear on the final image, as it is simply an overlay in the image program itself.

2. Brush vs Pencil

Ideally when dealing with pixel art using the pencil tool is always better than using the brush. Simply because Pencil tool is pixel perfect while Brush has some anti-aliasing involved (Adds pixels of various opacities around the main point). In some cases when dealing with shading a brush may help add more color gradients – though most of the time you will want to stick to using the pencil tool only.

3. Do not accidently shift the image

When editing the image – if you accidently shift the image by even one pixel it will have disastrous consequences. As you can see from the example to the left – I purposefully moved the entire set of frames then saved it. When attempting to see it in the game – you will notice the characters legs and pant legs are actually not aligning up properly.

4. Do not worry too much about perfect pixel transitions

I know a few of you are going to worry about pixel perfect transitions, where if a specific color pixel is in one location it must be placed in this specific location in the next image. Considering the speed of which the frames play – spending time on trying to make a 100% accurate transition is just going cause you to give up easier. Simply focus on transitioning the most important elements in a rough position and see it looks in game. If it seems right, that is more than enough.

5. The Frames File

Now if you look through vanilla armors you will notice the frames file missing in each one. You will also notice all the images have the exact same name. The reason for this is all the images share the exact same frames file, and since the frames file has to be the exact same name as the image – all images have the same name. Since it is the item file which designates which image to use and where – there is no issue of duplicates when it comes to image names only.

Another important point to keep in mind is all armors are kept 1 level below the frames file. To help you understand;

- The frames file is located in; \items\armors
- While the armor item will be located in; \items\armors\exampleArmor

This is an important point to keep in mind. If you do not follow the vanilla file structure when making custom armor – you will need to include a frames file – if you keep your armors or vanity items in a

`

different location. This also is important if you name your image differently than the vanilla naming convention.

6. Color Options

If you are going to include the ability to dye the armor – make sure you fix the color options appropriately to the new colors you have chosen. Otherwise the game can crash if someone tries to dye the armor. If you do not wish to deal with armor dying – remove the parameter completely.

TESTING OUT THE ARMOR

Once you have finished one piece it is a good idea to give it a test to make sure everything works before continuing.

Make sure you set up everything properly – as well as the .metadata file and launch the game. Check the mod manager in bottom right to confirm it is loaded. Go to the game and spawn the item.

If you did everything right – the armor will spawn. If for some reason it is empty or not spawning. Check your starbound.log in your storage folder. Remember to read the COMPLETE LOG FILE. Do not stop reading at the first error you see, read through each line carefully even if you do not understand what majority of it means. Sometimes the actual issue is buried in the log and takes a little effort – but a large majority of errors are given in plain English if one takes the effort to look.

HELMETS

Helmets get their own special section because they have one feature other armors pieces do not, the mask file. While the helmet acts similar to other armor parts in that it just overlaps whatever is underneath, the helmet does one extra step and completely hides the "hair" part of the character. The hair part is in quotes because some users have gotten pretty creative with the hair parameter of their custom races.

`

The mask file though allows the hair to be shown through the helmet. Let us take a look at the example image below.

- The first frame – has no helmet
- The second frame – has the helmet but the mask file is empty.
- The third frame – has a mask for the upper layer, allowing her hair to be shown through.

The mask dimensions is the size of a single frame.

When making a mask there are only 2 colors which are allowed – pure black and pure white. Using black allows any element in that area to peak through, while white blocks it.

FINISHING IT UP

Now that you have a complete idea behind the armor – make sure you add the appropriate recipe files and have some method of learning it (for example through player.config).

Now to give you an example of the importance of the item name over file name. Take a look at the player.config in the example mod in the download link provided. You will notice the item name and file name of the recipe are different. Yet in the player.config file we only wrote the actual item name. I did this only to stress the importance the actual item name must be used.

Example Files: http://www.mediafire.com/download/em55wmwuez8kmob/seb_armor.zip

REFERENCE

Some information in the Index may at some point become outdated, inaccurate or missing information. If you find such a mistake please report it in detail here:

http://community.playstarbound.com/threads/unofficial-modding-ebook.96671/

FRAMES FILE AND THE SPRITE SHEET

PARAMETERS

Now we will cover the various parameters seen in the frames file.

SIZE

The size parameter tells the game what is the size of a single frame. Not the size of the entire document. Every frame must be the exact same size in the sprite sheet.

DIMENSIONS

Dimensions is the number of frames in the sprite sheet based on Number of columns by the number of rows. So if your sprite sheet had 4 columns and 6 rows. It would be [4,6].

NAMES

Names lets you call upon a specific image in the sprite sheet. For example if you want to use a single image for 10 objects. An in the sprite sheet has 10 objects, you can call each image separately.

A good example can be found in box1 frames: \unpacked\objects\generic\box1

ALIASES

Aliases helps you define specific "states" for your animation file. So if you want particular frames to be called upon through lua. Otherwise this is not needed at all.

An example can be found at: \unpacked\objects\tiered\tier3door

`

THE OBJECT FILE

The object file is a simple JSON file which describes the attributes of the object. From its name to its collision box and even allows you to attach scripts to the object.

```
{
  "objectName" : "testObject",
  "colonyTags" : ["misc"],
  "rarity" : "Common",

  "category" : "storage",
  "price" : 100,
  "description" : "Item description goes here",
  "shortdescription" : "Name of Object",
  "race" : "generic",

  "inventoryIcon" : "testObject.png",
  "orientations" : [
    {
      "dualImage" : "testObject.png",
      "imagePosition" : [-8, 0],
      "flipImages" : true,
      "spaceScan" : 0.1,
      "anchors" : [ "bottom" ]
    }
  ]
}
```

Now the example given to the left is essentially some of the most basic requirements for an object.

OBJECTNAME

Object Name is one of the most important parameters. The object name does not appear in game, it is simply used by the game to identify the object. Which means you need to create a unique enough name that no other mod may end up having the same name. Also if you use a prefix to designate your mod – it will be useful for you as well as other modders to find out which mod is producing errors in the starbound.log As every modder will not be familiar with items from another mod.

TAGS

Tags is an optional parameter used by the NPCs in the game for the colonization system. There are specific tags each NPC type looks for. You can use a Find in Files search of a popular code editor to find all the different tags available. I will not be listing them out.

RARITY

Rarity has no particular purpose besides creating a unique borderline around the object in inventory. You cannot create new rarity categories. The only Categories currently available are; common, uncommon, rare, and legendary.

PRICE

The price defines the cost of the object to merchants

DESCRIPTION

This is the description you will find in the description box of the item.

`

SHORTDESCRIPTION

This is the actual name of the object which appears in the game. There is no issue if this name matches the same name of another item.

RACE

Serves no real purpose

<RACE>DESCRIPTION

This unique parameter will allow any vanilla or custom race to say this specific line when using the magnifier on an object. In the case of custom mods – the <race> parameter must be the exact same name as in the name in the species file.

INVENTORY ICON

This will choose the image within the same directory as the object file. Ideally the image should be 16 x 16 px for best clarity but you can use any resolution and the game will downscale it for you.

ORIENTATIONS

Orientations is a pretty huge topic by itself so I will not be covering everything here. Long story short Orientation is how the object is placed in the environment and if it interacts differently based on how it is placed. For example a person could use a different image based on the direction it is placed. To keep things simple use dualImage and spacescan – which will handle basic orientations and collisions. If you want to use more of the complicated features it is a good time to look through the vanilla files and see how it is done. It is much better to learn through example of the vanilla assets.

`

CRAFTING TABLE

A few users have asked about crafting tables – since I didn't feel it required its own tutorial. I thought I would just give a quick idea. As always the best way to learn is just to look at the example files and see "what is different". So let us look at the file Iron Crafting table found at;

\objects\crafting\ironcraftingtable

The only difference between a crafting table and a decorative object are these few lines,

```
"interactAction" : "OpenCraftingInterface",
  "interactData" : {
    "config" : "/interface/windowconfig/crafting.config",
    "paneLayoutOverride" : {
      "windowtitle" : {
        "title" : "  Iron Crafting Table",
        "subtitle" : "  Heavy duty crafting!",
        "icon" : {
          "file" : "/interface/crafting/ironcraftingtable.png"
        }
      }
    },
    "filter" : [ "plain", "craftingtable", "ironcraftingtable" ]},
```

Config: Is the window style you want to use. If you do not know how to edit UI stick to the default one. Though you can make custom UI if you are creative. Such as the one done by Pixel Good Store

The large buttons to the left are actually the category tabs. As you can see the user completely overhauled the look. Though as I said earlier – if you don't know how to design the UI – it is best to avoid it until you get some practice under your belt.

Title: The name shown on the top of the crafting table

Subtitle: The small text under the titile.

Icon: The tiny image shown in the UI.

Filter: Filter are the "recipe groups" which it will show.

`

METADATA

The modinfo informs the game a mod exists, the name of it, and how to treat the modinfo file. The modinfo file has 3 parameters of which only the name is necessary.

```
{
  "author" : "Swat | Elite",
  "description" : " Mod description",
  "friendlyName" : "Purchasable Pets",
  "includes" : [],
  "name" : "p_Pets",
  "requires" : [],
  "version" : "1"
}
```

NAME

The name is the designated of the mod. This is not shown in game and so it is a good idea for it to be unique enough it won't cause a conflict with another mod. When you pack the mod you will be asked to "name" your mod. That name provided upon packing is what will be shown in the game screen.

FRIENDLYNAME

Friendly Name is the name displayed in the game mod menu.

AUTHOR

Mod author's name or username to be displayed in game.

DESCRIPTION

Description is the information provided in the large text box in the mod menu.

Special Note for Steam Workshop Users.

If you are going to use steam workshop to distribute your mod. Keep in mind Steam overwrites the description when edited through the workshop. So the description displayed in the Steam workshop will be the same one displayed in game.

VERSION

This is the version designation of the mod, it does not need to be a number. The value provided is a string so it must be in quotes.

REQUIRES

The "requires" parameter tells the game this mod requires assets of another mod. The name provided here has to be the exact name provided in the other mods modinfo file. This mod will not allow the game to load without the other mod present. This in turn will also cause this mod to load after the mod provided in the "requires" field.

`

INCLUDES

The "includes" parameter is a more flexible choice unlike requires. This parameter causes the mod to load after a particular mod. You use this parameter when your mod does not require assets of another mod, but may edit the same files as another mod. In which case you may want your mod to load after another.

PRIORITY

Priority forces a load order priority for some mods which demand to be loaded earlier than others to prevent compatibility issues. Default is 0 – otherwise priority based on the alphabetical order of the mod.

RECIPES AND LEARNING RECIPES

One of the most common questions in the forums I see is, "I made my item but it is not being shown."

One extremely important detail is you **always** write the itemName or objectName in any of the options below. The actual file name itself does not matter. So you can not have a method directly where you have to unlock "different variations" of the recipe. Once you learn the recipe – you unlock every variation of the recipe for that specific item or object.

There are 3 methods in which a player can unlock a recipe.

- The player.config file
- Through an item by learnBlueprintsOnPickup
- Or through the Species File

THE PLAYER.CONFIG FILE

Recipes added through the player.config file will cause that item recipe to be learned instantly. When adding to the player.config you must add each element one at a time, example;

```
[ {
"op" : "add",
"path" : "/defaultBlueprints/tier1/-",
"value" : { "item" : "seblegs" }
},
{
"op" : "add",
"path" : "/defaultBlueprints/tier1/-",
"value" : { "item" : "sebhelm" }
} ]
```

LEARN BLUEPRINTS ON PICKUP

If you want the recipe to be learned after a user finds a specific item, this is the best way to go. Simply add the line,

```
"learnBlueprintsOnPickup" : [ "example", "example", "example" ]
```

This will allow the user to learn the recipe after picking up that item.

SPECIES FILE

Another method of learning the recipe is through the species file. The advantage of the species file is it limits the recipes to be only accessible by a specific race.

`

USEFUL LINKS

Patching Guide:

- http://community.playstarbound.com/threads/basic-patching-now-with-path-guide-v1-9.84496/

JSON Linter:

- http://json-schema-validator.herokuapp.com/jsonpatch.jsp
- http://helmet.kafuka.org/sbmods/json/patch.html

Starbound.log and interpreting Errors.

- http://community.playstarbound.com/threads/how-to-report-mod-errors-fixing-most-problems-yourself-v1-1.69198/

Lua Tutorials and Documentation:

Keep in mind not all Lua functions are available in Starbound. You just need to get a basic grasp of; How to write a function, Basic Operators, Tables, Arrays, Variables, Loops \ While, If \ Else, Math Functions. Once you learn these basics – you should be able to easily utilize the Lua API to your own ends. The Lua docs for the game are found in the Docs folder in your Starbound Directory. Starbound currently uses version 5.2 of Lua.

- http://community.playstarbound.com/threads/lua-part-1-terms-and-strings-v1-2.84833/
- https://www.lua.org/pil/contents.html
- http://www.phailed.me/2011/02/learn-lua-the-hard-way-1/
- http://luatut.com/crash_course.html
- http://www.tutorialspoint.com/lua/

Lua Online Interpreter:

- https://repl.it/lua

Lua Starbound Docs. (Also found in your Starbound \ docs directory.)

- http://starbounder.org/Modding:Lua

Pixel Art Tutorials

- Gimp Pixel Art Tool Setup : https://www.youtube.com/watch?v=PONe4IIYSnQ
- 30 Pixel Art Tutorials: http://www.hongkiat.com/blog/pixel-art-tutorials/
- 80 Pixel Art Tutorials: http://www.andysowards.com/blog/2012/80-epic-pixel-art-tutorials/
- Pixel Art with Paint.Net - http://blog.en.uptodown.com/draw-pixel-art/

`

STARBOUND COMMUNITY TOOLS

- Starmodder Kit (Windows) : http://community.playstarbound.com/threads/starmodder-kit-the-modding-ide-wip.117686/
- GUI Front End – for unpacking and paking files (Windows) : http://community.playstarbound.com/threads/updated-asset-packaging-unpackaging-gui-frontend-for-1-0.95468/
- Star Fuse – mount pak files for easy access (Mac \ Linux) : http://community.playstarbound.com/threads/alpha-0-4-0-starfuse-pak-utility-for-linux-os-x.115082/
- Mod Pack Helper (Windows \ Linux) : http://community.playstarbound.com/threads/all-versions-win-linux-modpackhelper.92473/
- Collisioner – helps create custom collision coordinates for objects and items: http://community.playstarbound.com/threads/collisioner.92466/
- Starbounder – File Folder asset Navigation (Windows) : http://community.playstarbound.com/threads/release-starbounder-navigation-made-easy.105973/
- Recolor Maker – Quickly Recolor Assets (Windows) http://community.playstarbound.com/threads/update-v1-1-2-recolor-maker-2.105981/
- Lua API MD File Viewer (All OS) : http://community.playstarbound.com/threads/api-help-file-viewer.120433/
- Lua Syntax Sublime Plugin (Requires Sublime Text Editor) : http://community.playstarbound.com/threads/sublime-text-starbound-lua-syntax.96330/

`

SPECIAL THANKS TO CONTRIBUTORS

Corrections by: Tofu Mc Dog, Sylvester334, pythondude325

Additional Information: Ev1lord, v6000

CLOSING REMARKS

If this book has helped you get into modding, please remember if you learn something new try and share it with the community. Sharing what you learn is the best way to support community growth, especially for those who are multi lingual, foreign language tutorials are nearly nonexistent..

If you make a mod and have read this book, feel free to drop me a line on my Profile Page, http://community.playstarbound.com/members/the-suit.12375/ - I always love to see what people make.

Till Next Time.

-The Suit

`