Parts Pack Production Procedure

# Preamble

This guide is based on the videos: [How to create parts for KSP2](https://www.youtube.com/watch?v=9fQg-oMqcH8) and [How to make engines for KSP2](https://www.youtube.com/watch?v=G-g0AxrFMGM), and on other notes and guidance from the KSP2 Modding Society discord.

Also the web pages: [KSP2 Part tutorial (almost) from scratch](https://luxstice.notion.site/KSP2-Part-tutorial-almost-from-scratch-1f336b7c97ae4280afb6a4e3aa6080b2), and [Tutorial: My First Part](https://luxstice.notion.site/Tutorial-My-First-Part-5f0cf456d7f4443d8c92658c7cc58314)

# Prerequisites

You will absolutely need the following things at a bare minimum:

* Unity Editor
* ThunderKit
* Unity KSP Tools
* Addressables Package (Install via Unity Window menu > Package Manager
  + Change Packages to Unity Registry
  + Search for “addressables”. Click on it and install.
* LuxShader

You will generally also need the following things:

* Blender (or other mesh modeling SW that can produce FBX files)
* Various blender addons such as TexTools, etc. (make a list of the free ones you’re using)
* Substance Painter (or other texture painting tool – could be Blender or Quixel Mixer, but SP is recommended highly)

You probably need a few other things too:

* JSON exports of similar stock parts to show you what some things can or should be set to

# Process

This guide covers the part of the process that takes place in the Unity Editor resulting in an assembly you can load as a codeless mod in KSP2, and how to get that result into the game.

In Unity editor start with a fresh scene. This will hold your entire parts pack.

## Part Pack Prep

1. **Create Unity Project**: Create a new Unity project with an empty scene. You can use the sample scene.
2. **Install Addressables package**: Open Window > Package Manger and select Packages: Unity Registry. Search for addressables and click Install.
3. **Install ThunderKit** in Unity. See [Installing Tunderkit](https://luxstice.notion.site/Installing-ThunderKit-7e00f13efe804f12a31a9f31d7f9fd6e) from the Getting Ready section of the KSP2 Part Tutorial linked above for details.

A screenshot of a computer

Description automatically generated

A screenshot of a computer error message

Description automatically generated

1. **Install Unity KSP Tools** in Unity. See [Installing KSP2 Unity Tools](https://luxstice.notion.site/Installing-KSP2-Unity-Tools-cfaa17f6ed2544378f6a627e5509e3c3) from the Getting Ready section of the KSP2 Part Tutorial linked above for details.
2. **Install LuxShader** in Unity. Place a copy of “LuxShader” somewhere in your Unity project. I put mine in my Parts Pack folder.
3. **Configure Unity Project:** This should have been configured when you installed ThunderKit, but you can check it here. Go to the Addressables Window to configure some project properties by clicking the Unity menu for Window > Asset Management > Addressables > Group. Select Default Local Group in Addressables Groups window. In the Inspector window, check the following properties for your parts pack mod.
   1. Build Path: Should be set to LocalBuildPath with a value of Library/com.unity.addressables/aa/Windows/StandaloneWindows64
   2. Load Path: Should be set to LocalBuildPath with a value of {SpaceWarpPaths.<your\_mod\_id>}/addressables/StandaloneWindows64
   3. Make sure the Load Path is *not* {UnityEngine.AddressableAssets.Addressables.RuntimePath}/StandaloneWindows64 (which is seems to want to be…) If you need to updated these, you can do so by clicking the System Settings button, and from there click Manage Profiles

A screenshot of a computer

Description automatically generated

1. **Create Parts Pack folder**: Under Assets, create a folder (L-Click Assets: Create > Folder) and name it the same as your parts pack mod. If your mod is called My Awesome Mod then this would be Assets\MyAwesomeMod.
2. **Create Materials folder**: Inside your parts pack folder create a folder called Materials. You’ll be storing the textures and materials you need there. E.g., Assets\MyAwesomeMod\Materials. This is just to aid in organization.
3. **Create Parts folder**: Inside your parts pack folder create a folder called Parts. You’ll be storing the part meshes and related things there. E.g., Assets\MyAwesomeMod\Parts. You can have whatever organization you like here, so if you want to group some parts you might create group folders within Parts (e.g., Methalox Engines, Nuclear Engines, Ion Engines, etc.). This is just to aid in organization and is optional.
4. **Create Plugin Folder and Content**: In your KSP2 install’s BepInEx\plugins folder create a folder for your mod. You can name this whatever you like, but it should be unique. This will be the “mod” players will install to have your parts pack and we’ll refer to it as the Base Plugin Folder in this guide.
   1. **Create swinfo.json**: In the base plugin folder create a swinfo.json file with content like this:

{

"spec": "1.3",

"mod\_id": "com.github.schlosrat.SPARK",

"author": "schlosrat",

"name": "Stellar Plasma-Assisted Rocket Kinetics",

"description": "High ISP engines for your low thrust needs",

"source": "https://github.com/schlosrat/SPARK",

"version": "0.0.1",

"version\_check": "https://raw.githubusercontent.com/schlosrat/SPARK/main/SPARK/swinfo.json",

"ksp2\_version": {

"min": "0.1.3.2",

"max": "\*"

},

"dependencies": [

{

"id": "com.github.x606.spacewarp",

"version": {

"min": "1.4.0",

"max": "\*"

}

},

{

"id": "lfo",

"version": {

"min": "0.2.0",

"max": "\*"

}

}

]

}

* 1. **Create “addressables” folder**: In thebase plugin folder, create an “addressables” sub-folder. This will be where you put the files created by Unity after a build (discussed below).
  2. **Create “localizations” folder**: In thebase plugin folder, create an “localizations” sub-folder (NOTE! This is currently required to be exactly this – localizations (plural), not localization (singular). This will be where you put the localization files (e.g., english.csv, spanish.csv, etc.).
  3. **Create Localization file(s**): Localization files are CSV files following a particular format. These must have lines ending with LF not LF/NL, and they must reference the same <part\_name> you use in the Part Production process below. There are some other restrictions for content, particularly that if you want a string that contains a “,” that string needs to be enclosed in quotes or the comma will mess with how the strings are parsed. These files are where the part’s **Title**, **Subtitle**, **Manufacturer**, and **Description** are configured. Here’s an example:

Key,Type,Desc,English

Parts/Title/spark\_spt100,Text,,SPT-100

Parts/Subtitle/ spark\_spt100,Text,,Hall Effect Thruster with Xenon Tank

Parts/Manufacturer/spark\_spt100,Text,,"Stellar Plasma-Assisted Rocket Kinetics, inc."

Parts/Description/spark\_spt100,Text,,"The SPT-100 is the pinnacle in tiny (0.625m-class) Ion engines, providing high Isp and low thrust with an integral toroidal xenon tank. Strap this little guy onto a probe core and be sure to bring plenty of electrons because you're in for a long slow ride to anywhere!"

Parts/Title/spark\_x3,Text,,X3 NHT

Parts/Subtitle/spark\_x3,Text,,Three-Channel Nested Hall Effect Thruster

Parts/Manufacturer/spark\_x3,Text,,"Stellar Plasma-Assisted Rocket Kinetics, inc."

Parts/Description/spark\_x3,Text,,"The SPARK X3 is the pinnacle in small (1.25m-class) Ion engines, providing high Isp and low thrust. Strap this bad boy onto your large probe or small capsule and be sure to bring plenty of electrons because you're in for a long slow ride to anywhere!"

## Part Production

1. **Create Root Part Object**: Create an empty game object under your scene (L-click Scene: Game Object > Create Empty). Name this object the same as your part.
   1. Recommended naming scheme: <mod\_name>\_<part\_name>. If your mod is called “My Awesome Mod” and your part title is “My Part”, then your part name might be my\_awesome\_mod\_my\_part for example. Part names must be unique, though you can have any descriptive title you like (that is done later in Localization). A naming scheme like this helps to prevent naming collisions in case anyone else might make a part they want to call “My Part”, like yours.
2. **Create Model Object**: Create an empty game object under your root part object and name this one “model”.
3. **Create Part Folder**: Create a Part folder named for your part inside your Parts folder. E.g., Assets\MyAwesomeMod\Parts\MyPart, or Assets\MyAwesomeMod\Parts\ThisGroup\MyPart if you’re grouping parts.
4. **Bring in FBX**: Drag a copy of your part’s FBX file into the part folder.
   1. If you have baked textures for your part that go with the FBX (meaning, they’re based on the UV Unwrap specific to that FBX), then drag those into this same file with the FBX.
5. **Create Part Object**: Drag a copy of the part FBX from *the Part folder in Unity* to the *model object created in step 2*. Don’t drag the FBX file from your computer’s files system, you need to use the copy you just placed in step 4. This will create a prefab for your part as a child of the model object.
6. **Unpack Prefab**: Left-Click part object: Prefab > Unpack.
7. **Remove Unnecessary Things**: Remove any parts that came in with the FBX that you don’t actually need in the game like lights, cameras, empty nodes, etc. If it’s not an actual part you want the game to render, then delete it.
8. **Orient Part**: Your model will appear in the Unity scene oriented as you built it in Blender, but this may not be the way you want it to be oriented in the game. If you need changes to the position, rotation, or scale of the part do those now using the Transform panel within the Inspector Window with your part object selected. For example, to flip a part over just give a rotation of 180 in Z, etc.
9. **Create Mesh Object**: L-Click part object: Create Empty. Name this object “mesh”.
10. **Create Collider Object**: L-Click mesh object: Create Empty. Name this object “col”.
11. **Create Collider**: In the Inspector window for the col object click “Add Component”. Search for Mesh Collider and pick it. This will create a Mesh Collider component in the col object. Click the arrowhead to the left of it to expand it and see its properties.
    1. **Select Mesh**: In the Mesh Collider properties select the Mesh you want to use. If your FBX is all one object you can pick that, or you can pick a suitable primitive like cube or cylinder, etc.
    2. **Position, Rotation, and Scale**: Set the position, rotation, and scale of the mesh to encompass the part. You should see a green mesh represented in the Scene window to help guide you to make sure you’ve got the right position, rotation, and scale.
    3. **Convex**: Check “Convex”
12. **Add Core Part Data**: Select your *Root Part Object*. In the Inspector Window click “Add Component”. Search for “Core Part Data” and pick that. Open it up and configure as follows:
    1. **Part Name**: The Part Name needs to be the same as what you’ve used for the Root Part object, i.e., my\_awesome\_mod\_my\_part or whatever you used.
    2. **Author**: Use what you like here, typically your KSP Forum screen name or whatever you go by as your modding author name.
    3. **Category**: Select an appropriate category for your part.
    4. **Family**: If you wish to identify a “Family” for your part, this needs to be a particular string. You can find examples in the game’s files, or part JSONS, or ask in the KSP2 Modding Society discord to get this information.
    5. **Co Lift, Co Mass, Co Pressure, etc.**: These parameters allow you to set the Center of Lift, Center of Mass, Center of Pressure, etc. Adjust these to get the markers in the Unity scene where they should be for your part. Typically, Co Pressure and Co Lift are in the same place.
    6. **Fuel Cross Feed**: Check if fuel should be able to transit through your part on the way to other parts. Typically set to true, but not always.
    7. **Mass**: Set this in metric tons, not Kg.
    8. **Attach Rules**: Check the types of attachment your part should allow. Checking “Stack” or “Srf Attach” will allow your part to attach in a stack or to a surface. Checking “Allow Stack”, “Allow Srf Attach”, etc. will allow *other* parts to stack attach or surface attach respectively. Currently (?) Allow Collision, Allow Dock, Allow Rotate, and Allow Root have no effect in game (check this).
    9. **Attach Nodes**: If Stack is checked above, then you need a “top” and a “bottom” node, if Srf Attach is checked above, then you need a “srfAttach” node. Note, node names are case sensitive and having a node is not enough by itself, you do also need the corresponding Attach Rule set true or the node will have no effect. Under Attach Nodes click the + button to add a blank node and configure as needed.
       1. **Node ID**: “top”, “bottom”, “srfAttach”, etc. (case sensitive!)
       2. **Node Type**: Select as appropriate. (Stack for Stack, Surface for Surface…)
       3. **Attach Method**: Select Fixed\_Joint for Stack and Hinge\_Joint for Surface Attach.
       4. Is Multi Joint: In general set to True for stack attach to help prevent noodle rockets.
       5. **Multi Joint Max Joint**: Set to 3 if you set *Is Multi* Joint to true?
       6. **Position**: Set as appropriate. Should be on the skin or outside of the part where you would expect to find it on your part in the VAB.
       7. **Orientation**: Set as appropriate. The Orientation vector should be a unit vector (length 1) pointing in the direction of the part that will attach to the node, so pointing away from your part.
       8. **Size**: Affects rigidity of your part. If your part is connected to another part with the same “size” node, then rigidity will be optimal, and otherwise it will be suboptimal.
       9. **Visual Size**: Set the same as Size.
       10. **Is Resource Crossfeed**: Set as needed for this node.
       11. **Is Rigid**: Set as needed for this node.
       12. **Rinse and Repeat**: Subsequent nodes created with the + button will inherit settings from the last node made, so this may accelerate the process as you just need to change the Node Id, Position and Orientation for new nodes that are similar to the previous created node.
13. **Add Module\_Color**: You need this to be able to paint your part with base and accent colors. As above, click Add Component and search for Module Color.
14. **Add Module\_Drag**: *All parts need this*. As above, click Add Component and search for Module Color.
15. **Add other modules as needed**. For example, if your part is an engine you’ll also need:
    1. **thrustTransform** object: Create an empty game object as a child of the root part and name it thrustTransform.
    2. **Throttle VFX Manager**: Configure as needed (?). No need to drag anywhere, it just need to be a component for the part.
    3. **Flameout VFX Data**: Drag this up to the Flameout VFX property in Module\_Engine.
    4. **Module\_Gimbal**: Configure details as needed (e.g., Gimbal Range and Gimbal Speed, etc.), then drag this up to the Gimbal property in Module\_Engine.
    5. **Module\_Generator**: Configure details as needed, then drag this up to the Alternator property in Module\_Engine.
16. **Apply Options**: Select the root part and in the Inspector window near the top on the Prefab line, click the Overrides dropdown and pick “Apply All”. If this option is not available, then you’ve got nothing you need to do here. Move along, move along.
17. **Save Part JSON**: Click Save Part JSON button at the bottom of the Core Part Data module. This will put the resulting part JSON in the Assets folder for your Unity project. You need to do this any time you’ve edited the Core Part Data module (or also a module it depends on?).
18. **Make Prefab**: Grab the root part object and drag it to the Unity project Assets folder.
19. **Add Part Icon**: Create an icon for your part that the game will use in the parts picker. This needs to be a PNG file with specific dimensions. It should conform to the style used by other parts in the game. However you do this, you need to name the file <part\_name>\_icon.png, and you need to drag that file into the Assets folder in Unity.
    1. **Convert Icon to Sprite**: Select the part icon in the Assets folder. In the Inspector window, click the pulldown menu for Texture Type and change this from Default to Sprite (2D and UI), then click Apply. If this is not done, your lovely icon will not display!
20. **Make Root Part Addressable**: Select the root part’s prefab in the Assets folder and in the Inspector window check the box for Addressable.
21. **Make JSON Addressable**: Select the part’s JSON in the Assets folder and in the Inspector window check the box for Addressable.
22. **Make the Icon Addressable**: Select the root part’s icon in the Assets folder and in the Inspector window check the box for Addressable.
23. **Configure Addressable Properties**: In the Addressables Groups expand the Default Local Group and find your part.
    1. **Group Name \ Addressable Name**: Change the information in the Group Name \ Addressable Name from “Assets/<part\_name>\*” to be just “<part\_name>\*”. So “Assets/<part\_name>.prefab” becomes “<part\_name>.prefab”, and so forth. You can leave the “Assets/” part of the path definition alone for each of these, that’s as it should be. The value for the Addressable Name needs to be the same as the file name it’s associated with and must not include any path parts. All of these need to be based on the Part Name established in the Core Part Data module.
    2. **Labels**: For the JSON set this to parts\_data. Leave it blank for the prefab. If parts\_data is not an option in the dropdown for Labels, then click Manage Labels, click the + button to add a new label, and set the Label Name to “parts\_data”. Click Save.
24. **Build Parts Pack Addressables**: In the Addressables Groups window, click the drop down for Build and pick New Build > Default Build Script. If you have any unsaved changes Unity will prompt you to save, and then it will build the addressables package for your mod.
25. **Copy Addressables to Plugin Folder**: If the Build step above was successful, the do the following.
    1. **Delete Old Stuff**: In Windows Explorer, navigate to your Base Plugin Folder’s addressables folder (<KSP2 Game Dir>\BepInEx\plugins\<your Mod Name>\addressables) and delete any content inside that folder. There may be four things: AddressablesLink (folder), StabaloneWindows64 (folder), catalog.json (file), and setting.json (file).
    2. **Copy New Stuff**: In Windows Explorer, navigate to your Unity project’s folder and from there to Library\com.unity.addressables\aa\Windows folder. If the build process above was successful, then there will be four things in this folder: AddressablesLink (folder), StabaloneWindows64 (folder), catalog.json (file), and setting.json (file). You will need to copy all of these to your Base Plugin Folder’s addressables folder.
26. **Launch Game and Test**!
27. **Rinse and Repeat for Additional Parts**

## Blender Basics for Beginners

1. If you’re making an engine, put an empty single arrow at the world origin facing in the direction your engine will fire, then move it along the axis of the engine so that it’s at the exit of the engine. Name this object thrustTransform, and keep in mind that size does not matter.
2. Make sure all the part objects in your blend have the same material, and that the material has a good name. The material you use for this can be a default material, nothing special is needed.
3. Make sure any Boolean modifiers are applied, but array modifiers can be left unapplied.
4. Make sure you’ve got a good UV Map (use UV Smart Project) or an excellent one (UV Smart Project followed by Pack Islands). Recommended island spacing for both is 0.003
5. When exporting your FBX for use in Substance Painter or any other tool, select only those objects that are actually part of your part and then be sure to check the box for Limit to Selected Objects

## Texturing Tips

1. Make sure the *Kerbal\_Space\_Program\_2\_Parts\_Paintable.spt* file is in your “Documents>Adobe>Substance Painter>Assets>templates” folder. If the templates folder doesn’t exist yet, then create one under Assets and drop that file in it.
2. Launch Substance Painter and create a New project.
   1. In the project creation dialog pick the *Kerbal\_Space\_Program\_2\_Parts\_Paintable* template
   2. Select your FBX file
   3. Make sure the Document Resolution is sufficient for your texture maps (highly recommend 4096! Any less and painted on bolts look like crap)
   4. Click Open.

A screenshot of a computer

Description automatically generated

1. Make sure the UV Map looks OK in Substance Painter. If it doesn’t, you need to go back to Blender and re-unwrap then export a new FBX and return to step 2 above.
2. Use the File > Import resources… menu to bring up the Import Resources dialog (shown below) and click on the Add Resource button to bring in the *KSP2\_Part.spsm* file. Repeat this step for the *Kerbal Space Program 2 - Standard Damaged.spexp* file. Both of these should go into your library as you’re going to need them with every part you make.
   1. Use the Add Resources button to find the resource you want to import
   2. Select the file and click the Open button
   3. Repeat steps (a) and (b) above for as many things as you would like to import
   4. Select the destination to Import your resources to (hint: library ‘your\_assests’ for these!)
   5. Click Import

A screenshot of a black screen

Description automatically generated

1. Use the process in Step 4 above to bring in any textures you want to use. There are many you can peruse and download for free here: <https://substance3d.adobe.com/community-assets>
   1. Note: When importing textures, you can either put them in your library so that you can use them in multiple projects, or you can put them directly in your project if you don’t want them in your library. Either way works.
   2. You can repeat this step any time, and as often as needed, so don’t worry if you don’t have all the textures you need at the start. It’s also possible to make textures within Substance Painter
2. Apply the KSP2 Part material to your whole object and once done delete the default later that Substance Painter provided.
3. Locate the Bake button (looks like a croissant) in the upper right of Substance Painter and click it 
   1. Uncheck the option to make an Opacity Mesh Map
   2. Click the large blue Back Selected Textures button
   3. When baking is done, click the Return to Painting Mode button
4. Expand the folder by the KSP2\_Part smart material in the Layers panel
5. Expand the folder by the Base Material smart material within the KSP2 Part
6. Search for and select the materials you want to paint your part within the Assets panel.
   1. Drag a material from the Assets panel onto the Base Material in the Layers panel.
   2. Right click on that material and add a black mask to it, making sure the resulting mask is selected (there will be a blue border around it)
   3. Pick the Polygon Fill tool from the tool bar and then select a option from Triangle Fill, Polygon Fill, Mesh Fill, and UV Chunk Fill.  Use the triangle to paint tris, the square (polygon) for quads, the cube (mesh) for entire objects, and the checkerboard (UV chunk) for connected faces on a side of surface.
7. Create a simple material: Example white boron nitride ceramic.
   1. Click the Add Fill Layer button (paint can) 
   2. Drag the new fill layer onto the Base Material folder.
   3. In the Propertied panel under Materials, leave only color, roughness, and metal selected.
   4. In the Properties panel set the base color to what you need (white in this case)
   5. Set the metallic slider to 1.0.
   6. Set the roughness slider to 0.3.
   7. Add a black mask to the fill layer
   8. Paint like a pro
8. Add “Nails”: This is where you add various surface details via the height map, and works for painting on nuts, bolts, fasteners, etc. All of these can be added as “alphas” like this
   1. Import your alpha(s)!
   2. In Layers, select the black mask for Nails in the Heightmap
   3. Select Radial Symmetry (for applications around radially symmetric parts like I've got here)
   4. Set your X, Y, Z point about which things will be symmetric, where Y is the vertical. (Switch the Show/Hide Manipulator on to help see where this point is in your model)
   5. Set Mirror Y, the count you want, and the angular span the radial pattern should follow (360 for all the way around)
   6. With the brush tool active, select your alpha
   7. Move your cursor onto the part and see the red dots where the "Nails" will be placed.
   8. Adjust the size of the "Nail" using [ and ] to make it smaller or larger respectively
   9. Click where you want the first "Nail" to appear, this will place them all.
9. Add “Stickers”: This is where you add signs, stickers, decals, etc., where the sticker has a uniform color (like a radiation symbol, etc.)
   1. Import your alpha(s)!
   2. In Layers, under Base Material, add a fill layer for the sticker
      1. Uncheck all material properties for the layer except color, metal, and rough
      2. Set the fill color to be what you want the sticker to look like
      3. Move the layer to be above any other Base Material layer that impacts the part(s) you’re going to apply stickers to (or it will be overwritten and you won’t see your stickers!)
   3. Select Radial Symmetry (for applications around radially symmetric parts like I've got here)
   4. Set your X, Y, Z point about which things will be symmetric, where Y is the vertical. (Switch the Show/Hide Manipulator on to help see where this point is in your model)
   5. Set Mirror Y, the count you want, and the angular span the radial pattern should follow (360 for all the way around)
   6. With the brush tool active, select your alpha
   7. Move your cursor onto the part and see the red dots where the "Stickers" will be placed.
   8. Adjust the size of the "Sticker" using [ and ] to make it smaller or larger respectively
   9. Click where you want the first "Sticker" to appear, this will place them all.