Age of Wonders 4 Official Modding Guide

TRIUMPH STUDIOS

Introduction

Welcome to the official Age of Wonders 4 modding guide! This guide is intended to get you started with creating and publishing mods for Age of Wonders 4 using the PC version of the game and the Paradox Launcher.

The guide starts off with several chapters on overall modding knowledge such as how to create a mod package and the basics on using the different tools provided with a PC installation of Age of Wonders 4.

Later chapters will cover specific content types such as Tomes and Units. It's advised to read through the first few chapters and then navigate to the chapter talking about the specific type of content you'd like to create a mod for. Note that this guide may not cover all specific types of content you can mod for Age of Wonders 4.

Disclaimer: Modding Tools are provided as a courtesy to fans. They might have different system specifications from Age of Wonder 4, are not tech supported and are only available in English.

This guide is released alongside example content that can be used as reference or inspiration for your mods! These mods are also fully playable and can be applied to your game!

Tome of Penguins mod

Massive Maps and 12 Players mod

Potato Spiders mod

You can also access the direct files including some example content files here.

Modding Rules

Triumph Studios and Paradox Interactive do not condone malicious or offensive content in mods. But it is up to mod platform holders to report and/or take action against malicious or offensive mods (such as Steam Workshop or Mod DB) and we cannot be held responsible for the resulting experiences or damages caused by mods. We therefore ask modders to have fun but be responsible when creating mods and for players using mods to take this into consideration.

Below are some guidelines when creating mods:

- Only use the tools provided by Triumph Studios when creating mods, unless specified otherwise.
- It is recommended to use POedit for creating and modifying additional text into the game. A free trial version of the application is available online.
- It is recommended to use Maya for creating 3D assets for Age of Wonders 4 mods.
- We do not support modification of text provided by Triumph Studios or additional translations.
- Try and keep modifying existing RPKs, CLBs and TAM files to a minimum, as they decrease the compatibility of your mod with other mods and can go out of date whenever a patch or update releases that contain changes to the files you've modified.
- When testing your mod, check if the modified content is also not present when your mod is disabled.

- Using our Editor tools, you can look at existing content made by us (Triumph Studios), you can use this as reference to figure out how specific content is set up and made.

- When making or copying files, it is best practice to keep the same folder structure as in the game's installation folder. By default when creating a new mod, new folders replicating the original folder structure will be generated.

- As of this moment writing the guide. There is no mod support for Age of Wonders 4 on consoles, nor are the modding tools provided in those versions of the game.

Getting Started

A Mod in Age of Wonders 4 consists of an .ACP file which is a package file that keeps track of all files that the mod wishes to load in, and files such as .clb, .rpk and .tam that hold the content the mod contains.
Mods can both override the game's original files/content and/or add content on top of the game's original content.

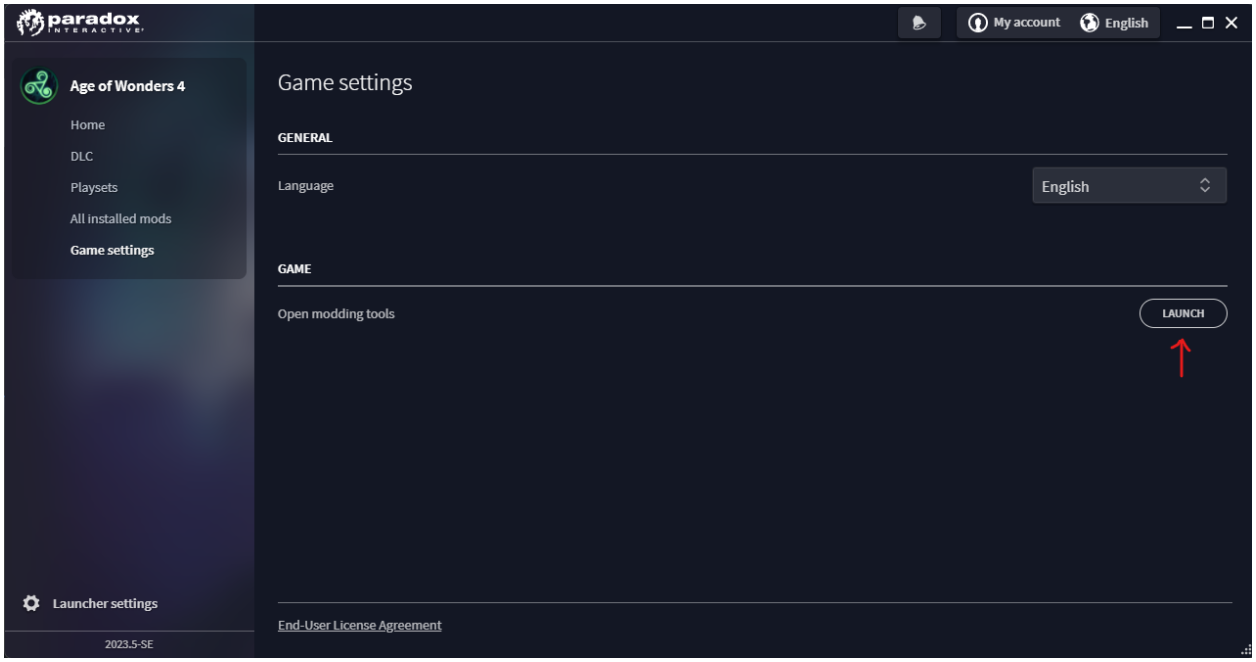
Using Package Manager

Package Manager will be the main tool used for creating and managing mod content. It is also used to launch other tools such as Resource Editor, Content Editor and Level Editor.

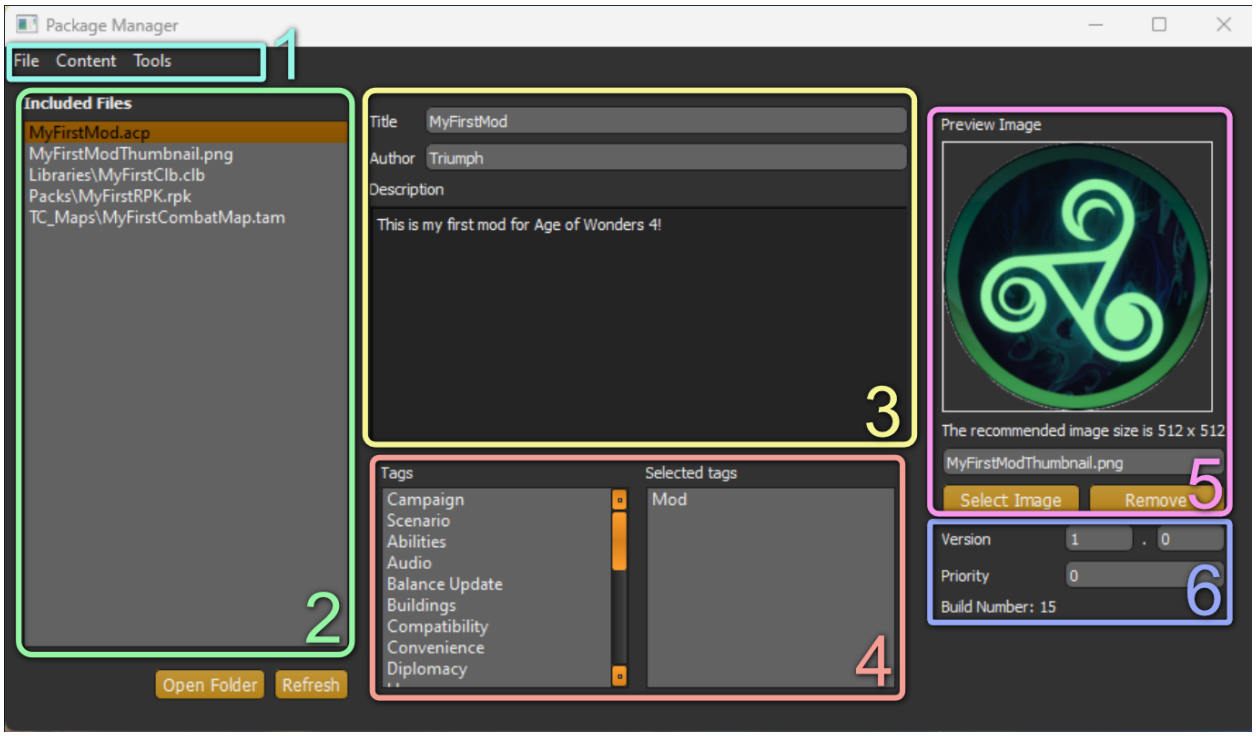
You can launch Package Manager by navigating to the installation folder of Age of Wonders 4 and finding *PackageManager.exe***.**

*Tip: when using **Steam**, you can easily find the installation folder by right clicking on **Age of Wonders 4** in your **Steam Library**, click on **Properties**, navigate to **Local Files** and click on **Browse…***

You can also open up the Package Manager via **Paradox Launcher**, by navigating to **Age of Wonders 4 -> Game Settings -> Open Modding Tools -> Launch**



Once opened up, let's look at the elements in the interface that makes up Package Manager:



1. The **Menu Bar** with 3 **Menu Categories** each with their own **Dropdown Menu**
    a. **File** - has all the functionality for creating, saving and loading Package files. A single Package file keeps track of all the changes and files a single mod contains.

    b. **Content** - Here you can link Packages (.acp) or Resource Packs (.rpk) that are necessary for this package to work. We also call this **Dependencies**. It helps generate errors when a mod could not be loaded in due to missing particular resources it depends on.
    c. **Tools** - from here you can launch the other tools: **Resource Editor** for **Resources** (.rpk files), **Content Editor** for **Art Assets** (.clb files), and **Level Editor** for **Combat Maps** (.tam files).

2. The **Included Files** list keeps track of all the files that are included in this mod. Whenever you place a new file in the mod's folder, the **Package** will pick it up to be used for the mod, and removing a file will also remove it from the Included Files list.
    a. The **Open Folder** button underneath the list allows you to navigate to the Mod's folder where you need to place files for the **Package** to detect them and include them in the mod.
    b. The **Refresh** button allows the **Package** to detect new or removed files in the Mod's folder and will update the **Included Files** list.

3. The **Title, Author** and **Description** are text boxes that can be used to name the mod, provide a description and credit yourself as the maker of your mod.

4. The **Tags** and **Selected Tags** list contain name tags to help identify what kind of content your mod will contain. Double clicking on any tag in one of the list will move it to the opposite list.

5. The **Thumbnail Image** of the mod will help create a visual icon for your mod. You can link an image file using the **Select Image** button. Or remove the currently selected image by clicking the **Remove** button.

6. The **Version** number of your mod will be used to detect newer versions of the same mod.
    a. If two mods with the same name are loaded, only the one with the highest **Version Number** will be loaded.

Creating a new Mod

Once you have opened Package Manager, you can create new mod by following these steps:

1. Click **File** -> **New**
2. Name your package with the name you want to give your mod.
3. Fill in any info about the mod as described in **Using Package Manager**
4. Save by either hitting **CTRL + S** or navigating to **File** -> **Save**
    a. Your Mod's files will be saved in *…\Users\[UserName]\Documents\Paradox Interactive\Age of Wonders 4\Mods\[Modname]*

The **Included File** list on the left will populate once we start adding actual content to the Mod! For now it should only contain *[modname].acp*

Publishing a Mod

Once you are ready to wrap up your mod for distribution. Load up your mod's .acp file in **Package Manager.**

Then, click **File** -> **Publish** and your mod will generate metadata, which is used when uploading your mod.

Navigate to the folder of your mod, by default this should be *C:\Users\%username%\Documents\Paradox Interactive\Age of Wonders 4\Mods*

You can Zip the folder of your mod and upload it, or upload it unzipped!

Testing a Mod

You can test a mod without publishing/zipping it. As long as the mod's .acp file and its content are in a separate folder in *…\Users\[UserName]\Documents\Paradox Interactive\Age of Wonders 4\Mods*

You can then make sure the mod is detected and loaded using **the Paradox Launcher:**

1. Launch the **Paradox Launcher** and navigate to the **Age of Wonders 4** launch window.
2. Underneath the Resume and Play button, a "**Playset**" dropdown can be found.
3. Open it and select "**Add new playset**" and give it a name.
4. You will be brought to the Playsets window with your new playset collected, press the "**Add Mods**" button.
5. **Tick the box** of the mod(s) you wish to test!
6. When you've selected the correct mods to test, click "**Add x to Playset**"
7. Navigate back to the Age of Wonders 4 home window, check if the right playset is selected, and hit **Play**! You'll now be playing the game with your mod enabled.

It is often handy to check the following things when testing your mod:

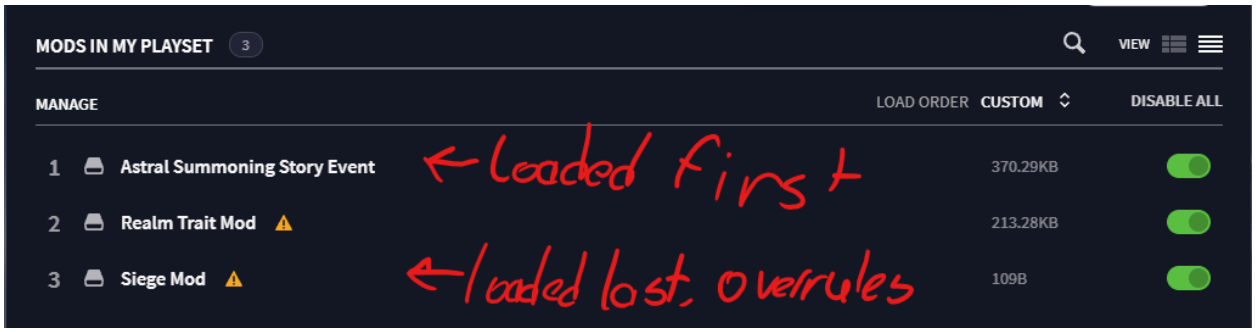
- Can you start a session successfully?
- Do the modded features/content show up and work as intended in game?
- Does any custom text of your mod display correctly?

Mod Compatibility and Load Order

When testing your mod with other mods, it can be that certain changes of your mod may be overwritten depending on what mod gets loaded last.

In **Paradox Launcher**, the list of mods in your **Playset** also depict the order which gets loaded, this can be a little confusing, because the load order is from **Top** to **Bottom** but this means that the **Bottom** Mod has the highest priority in overruling other mods.

Every time a new mod gets loaded, it will overrule previous mods if they modify the same existing content.



To adjust your load order, you can hover your mouse near the load order number next to the mod name. You can just click, hold and drag the mod up or down the list to adjust the order.

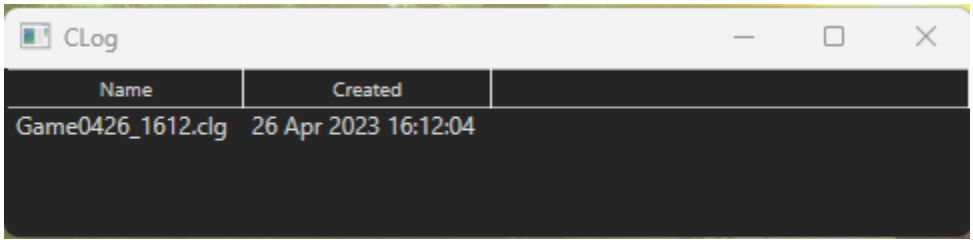
When testing your mod with other mods active, it's best to load your mod last.

Debugging with CLog

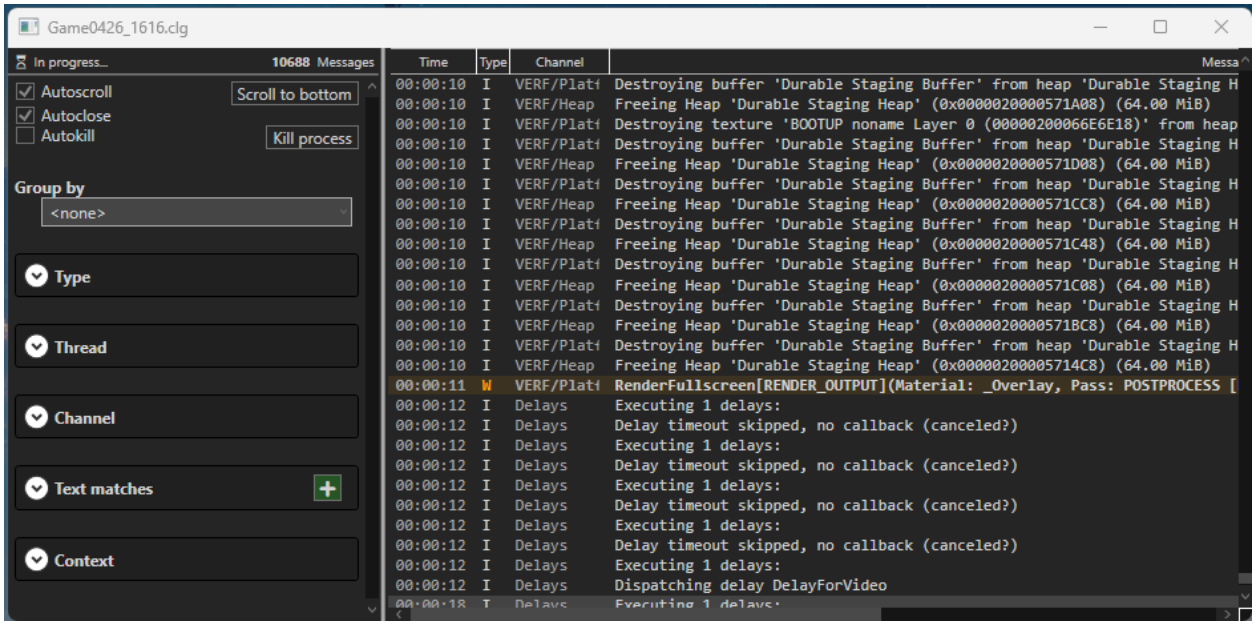
Clog is a tool that helps with generating logs of sessions in Age of Wonders 4. These logs contain information about what is happening in the background of the game. And often provide handy information on what might be going wrong when you're modding. CLog is always activated while one of the tools or the game is open, but its UI may be hidden. By opening it, it will provide you with active log tracking and also filtering a lot of log entries.

To open the UI of CLog, follow these steps:

1. Navigate to your local install folder, in case of **Steam**, this is something like *C:\Program Files (x86)\Steam\steamapps\common\Age of Wonders 4*
2. Find and open *CLog_UI.exe*
3. This should open up a **CLog** window! While no session is active, it will provide a list of logs that have already been stored. You can also find these yourselves in *C:\Users\%username%\Documents\Paradox Interactive\Age of Wonders 4\Logs*



4. You can click any of the *.clg files* in the list to open the log in CLog's UI. Alternatively, when starting the game with CLog open, it will automatically open and update the generated log for the new game, which should look something like this:



5. In here, you'll often be looking for particular entries to try and see if something you did caused problems. More on how to use filters and find functions to hone in on particular entries.
6. While having a tool open, a small active log will also be active in the tool's UI, these are stored in the *…/Logs* folder. They also show up in CLog's list and can be opened there.

While having CLog open, it's tricky to look for particular entries, luckily there are various ways you can filter the log:

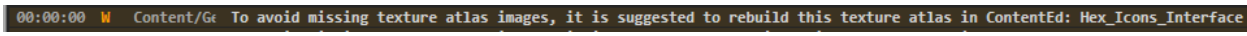
Using Type FIlters

- There are 3 types of log entries:
  - **Information**

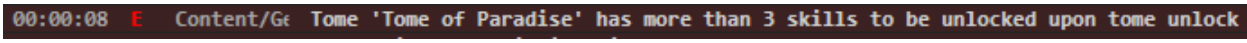
    ```
    00:00:01  I   Content/Ge  Content Registered: GAMECAMERA.RPK (H:7)
    ```

    - Shown in gray.

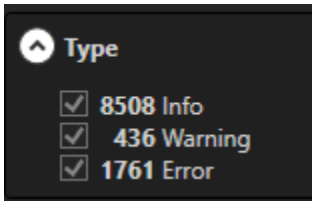
    - These are mostly notifications of particular events or processes in the game, these can often be ignored and thus turned off. As they often do not show up to inform you of anything gone wrong.
- **Warning**



    - Shown in yellow.
    - These often detect possibly problematic content or processes. But does not mean the mentioned content or process is actually causing any issues.
- **Error**



    - Shown in red.
    - These are the ones you'll often be looking for. They point out problems with content and processes. Such as broken resource links (dependencies that could no longer be loaded because the linked resource no longer exists or was moved) or settings in content that invalidates the content.



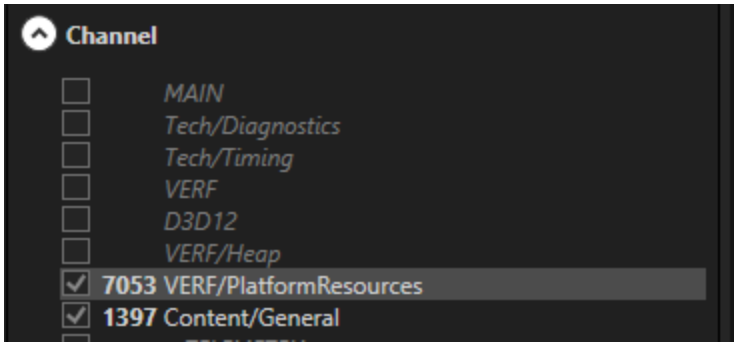
- In CLog, you can filter in/out these types by ticking or unticking the boxes, the number next to the boxes indicate the amount of entries of that particular type.

Using Threads

- This is usually not useful to modders, because it's mainly for programmers to check what processes are running in which thread. Best to ignore this for debugging purposes and leave them all ticked. It often does not matter in what thread your process or content runs.

Using Channels

- These are custom made filters by Triumph to categorize certain processes.
- Depending on what you're working on, you can switch certain channels off. But it's advised to usually keep *VERF/PlatformResources* and *Content/General* active.

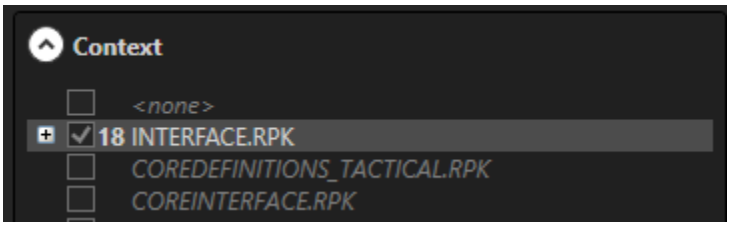


Using Text Matches

- In Text Matches, you can create Text filters that you'll more commonly use. By clicking the green **+** button, you can add a new text filter.
    - Fill in a **Name** for the filter
    - Fill in the **Expression** on which you want to filter with
    - Choose to use **Substring** which detects if the **Expression** is present anywhere, or use **Regex** for more specific searches using your **Expression**
- Once at least 1 Text Match filter exists, you can select it to use it. Or switch back to *None* to have it off. You can also edit any existing Text Match Filter by clicking the pen icon to its right.

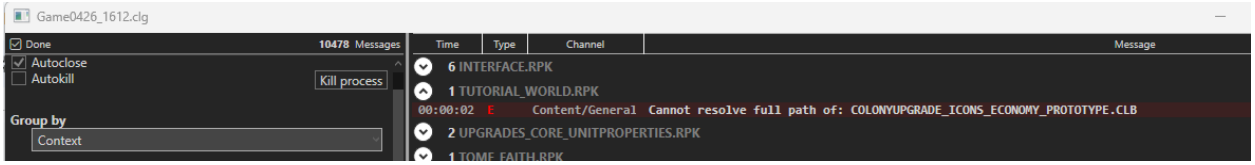
Using Context

- Context will filter out which exact content packages the errors originate from.
- Switching everything off until only one pack is left to check if a specific pack contains what error is not the best method here. Instead look at **Group By.**



Using Group By

- At the top of the filters is a **Group By** dropdown menu, by selecting a category there. It will group all the Log entries based on the category chosen.
- **Group By** *content* is especially handy in certain scenarios such as trying to track down if your modded pack(s) specifically generates any errors.
- You can open and close any group of entries by clicking the arrow icon on each group.



Using Find String

- By pressing **CTRL + F**, you can use a good old reliable find function to find a particular string.
- clicking the arrows up and down will cycle up or down in the log for the next entry matching your expression.

We hope this helps you find issues with your mod! Note that particular errors, such as **Scripts** not able to resolve, will not be logged in Clog.

Common types of errors to look for:

Broken Resource Links

- They can be seen as red links in our tools, or as *Broken Resource Detected!* In a log.
- This means a resource contains a link to another resource that has either been moved to a different pack, removed, or a pack dependency has been changed. Meaning that the linked resource can no longer be found.
- Note that **copying** a resource with CTRL + C and deleting the original will **break resource links** because the newly pasted resource is treated as a new resource and will have a different ID than its predecessor.
- To fix this:
    - If the pack dependencies might have changed, make sure to add the right *Resource Pack dependencies* in your packs by going to **Content -> Resource Packs** and making sure the right files are in the list.
    - If the resource was copied or removed, simply replace the link to the copied resource or *(none)*

Double Resource ID detected

- You'll usually get this as an Assert (pop up error) while loading the game.
- May be caused by a resource existing twice in different packs (such as copying an .rpk file in file explorer without removing resources or regenerating IDs).
- May be caused by Circular Dependencies (see below), usually when a lot of them are generated and they continue to generate until the game crashes.

Circular Dependency Detected

- You'll usually get this as an Assert (pop up error) while loading the game. This means somewhere there are Resource Packs linking to each other in a circular manner.

- This means that the game is stuck endlessly loading content, as Resource Pack 1 calls upon the game to load Resource Pack 2, and Resource Pack 2 calls upon the game to load Resource Pack 1. The game will never finish loading and crashes.
- To fix this, you can make sure to organize your packs dependencies in the **Content** menu in **Resource** and **Content editor**.
- If you're unsure how the circular dependency exists, you can check **Content** -> **Dependency Graph** in **Resource Editor** which lets you detect indirect dependencies. (Pack A loads Pack B which loads Pack C which loads Pack A again)

Editing Existing Packages

You can overwrite existing content easily in Age of Wonders 4. You don't have to be afraid to overwrite anything of the base game. When editing existing content, you automatically make a copy of the original .rpk, keeping your base game safe.

While you can change any resource in Age of Wonders 4, this guide will show a step by step example on how you can change the Pathfinder unit. The general principles will be the same for any resource.

Step by Step Guide

1. Open the **Package Manager**
2. Create a new package.
3. In the Package Manager, click on **Open Tools > Resource Editor**.
4. In the top left, click on **File > Open > Units > Units_Culture > Barbarian - Units**
5. Right click **Barbarian - T0 - Pathfinder - Unit** and click **Create Mod**.
6. While it is selected, scroll down to the **Properties** line under the Abilities and Properties header. Click the yellow cube to open the list of Properties.
7. Add the "*+25 HP"* property 4 times.
8. Save the package and close the Resource Editor.
9. The edited package gets automatically added to the Package Manager.
10. Save the package editor and publish the mod. Add the mod to your Playset, start a new game while playing the barbarian culture. If you select your Pathfinder scout, you will see it now has 100 extra HP!

Multiple Mods

If two mods edit the same resource, the one with the mod with the higher priority takes precedence. It is, however, possible to edit two different resources in the same package. So while two mods cannot both edit the Pathfinder unit, one mod can edit the Pathfinder, and a second mod can edit the Sunderer.

Editing Existing Content Libraries

We have gone over editing existing resource packages, however; you also have the ability to edit the underlying content that these resource packages use. For instance, the meshes and

materials used in the resources are all loaded through so-called *Content Libraries*. These have the *.clb* extension. They are located under the *Content/Libraries* directory within the root directory of the game. Within your mod directory, there's also a respective *Libraries* directory. This directory is used for when a mod wants to override libraries used in the game. *ContentEd* is the tool we use to **open**, **edit** and **save** these libraries.

Editing *Content* is usually not necessary and only required if you want to completely override how something in the base game looks. *Content* is purely visual and doesn't have any gameplay logic tied to it.

It is also worth noting that if you accidentally **save over** a base game's *.clb*, you would need to either:

- **Revert to a back-up** version of the file
- **Verify the Integrity** of your game files (*Steam*)

Some examples of *Content* are:
- Meshes & Models
- Textures
- Materials
- Animations
- Skeletal rigs

For this example, we'll be replacing the image of the bootup screen. Grab a nice image from somewhere, preferably in the *.png* format!

Step by Step Guide
1. Open the Package Manager
2. Create a new package (or open an existing one)
3. In the Package Manager, click on **Tools > Content Editor**
4. In *ContentEd*, click on **File > Open…** in the upper left hand corner
5. Navigate to your *Age of Wonders 4* root directory (not the mod directory!)
6. Go into **Content > Title > Libraries > Interface** and open *Bootup.clb*
7. Now, **Before doing anything**, make sure to create a back-up of the file, or simply go to **File > Save As…** and save it somewhere on your machine. (e.g. *Bootup_Original.clb*) This makes it easier to revert when something goes wrong
8. Go to **File > Save As…** and navigate to your mod directory.
9. Duplicate the directory hierarchy as it is in the original base game, starting from the *Title* directory
    a. In this case, the base game directory for *Bootup.clb* is in *Content\Title\Libraries\Interface*
    b. For this example, we would then save the modified *.clb* in *[ModName]\Libraries\Interface*, do this under the same name as the original file. The final path would be: *[ModName]\Libraries\Interface\Bootup.clb*

    c. If these directories **don't exist** yet, you can **create them** by right clicking in the File Save Dialog (under **New > Folder** in the context menu)
10. Click on the **TextureAtlasses** tab in this library
11. **Select** the *noname* Texture Atlas by left clicking on it in the list on the left
12. In the panel on the right of the content list, right click *BootupScreen2* and choose **Refresh From File**
    a. If you've already added the image beforehand, you can choose to **Refresh From Path**, which will simply refresh the image from its original location
13. Find your image that you want to replace the bootup screen with and click on the (now highlighted in red) **Build** button. It should change back to its original color when the atlas was rebuilt successfully
14. Press **CTRL+S** to save the pack, or go to **File > Save** in the upper left hand corner
15. Go back to the Package Manager and click on **Refresh**
16. You should now see the *.clb* being listed in the **Included Files** panel
17. Save within the Package Manager and publish the mod. Add the mod to your Playset, set the playset as active and boot up the game. You should now see your own image being the splash screen of the game!

Multiple content mods
As with resource packs, if two mods modify the same Content Library, the one with the highest priority takes precedence over the other. Whilst it's possible to combine multiple Resource mods, Content Libraries don't offer this flexibility. This is also why it's more favorable to add **new** libraries and link them appropriately in **existing or new** resource packs.

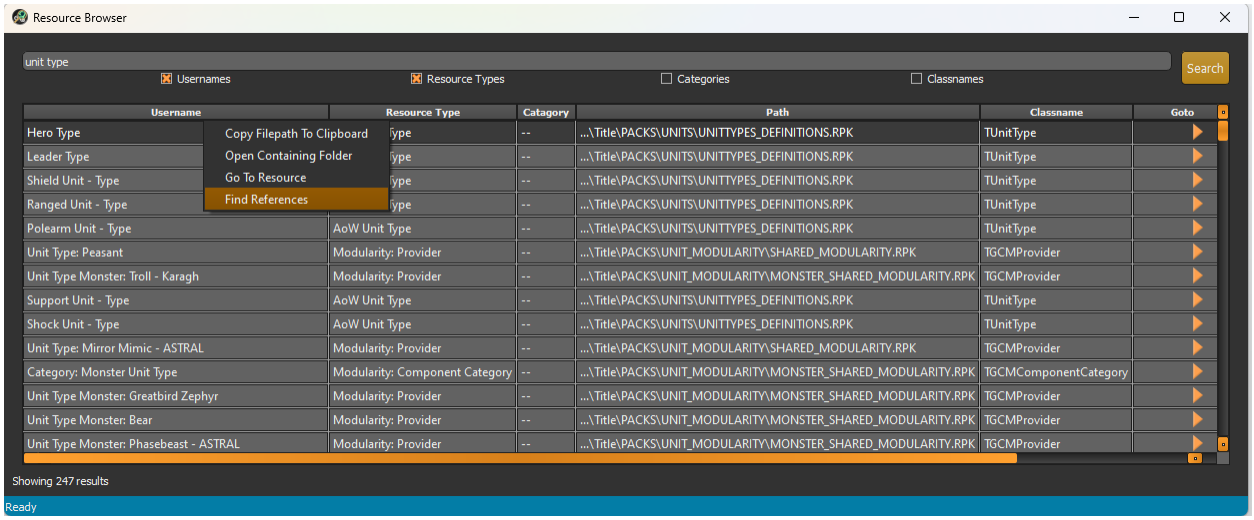
Resource Browser

If you want to edit a specific resource, or if you want to make a new mod and want to see how the base game does it, you can use the Resource Browser.

To access Resource Browser, load in your mod in Package manager and then in Package Manager, go to **Tools -> Resource Browser**

Once the Resource Browser has finished loading, you can type in any **Expression** in the search bar which will search all resources that match the filter in either their name or their resource. You can also:

- Open the resource by clicking the **Go To Resource** option or Go To button.
- Find all resources that **Find References**, which will change the current filter to get all resources that have a link to this resource.



Adding New Content

Adding Packages

1. Start a new project in the **Package Manager** and open the **Resource Editor**.
2. In the resource editor, click on **File > New**. You have now created a new pack.
3. On Category, click New, give your category a name and click Ok.
    a. Categories are necessary for any resources to be put in a pack. Otherwise it does not matter in which category resources are located. They exist purely for organization purposes.
4. Save the package in the location of your mod. By default this should be in: Documents\Paradox Interactive\Age of Wonders 4\Mods\YourModName\Packs.
5. In the Package Manager, click **Content > Packages > Add**, then find your package you want to add and do that here.

Adding Resources

1. Left Click on an empty line on the left. The line will become yellow to indicate it has been selected.
2. In the middle panel under the main tab, you will see **Type:** and **Name:**. Behind Type is a dropdown menu. This lists all the resources in the editor. Select which one you want and click on **New**.
    a. When the dropdown menu with all the resources is selected you can start typing for the resource you are looking for.
3. Change the resource from *noname* to whatever you want.
    a. It is recommended to use the format *[Content Name] - [Resource Type]* so you can find the resource easily. Example: *Fireball - Tactical Unit Targeter*.

Adding Content Libraries

Apart from being able to edit existing content libraries, you also have the option to add new *.clbs*. This is useful to not interfere with content from the base game and gives mods more flexibility to use their content in many different ways throughout the resource packages.

For this example we're going to be replacing the city structures on the *Strategic Map*, for a multitude of *Cultures*, through our *Modularity* system.

We're going to need the following pieces of content:

- A mesh (in the *.fbx* file format, see **FBX Notes**)
- A texture (preferably in the *.tga* or *.png* file format)

You can of course create these yourself, however; we have also included some example files that you can use for this process. These are provided alongside the guide online.

The relevant files are:

- *Castle_Mesh.fbx*
- *Castle_Diffuse.png*
- *Castle_Normal.png*

We're going to be replacing the *Tier I City Structure* of both the *Feudal* and *Barbarian* Cultures.

Step by Step Guide

1. Open the Package Manager
2. Create a new package (or open an existing one)
3. In the Package Manager, click on **Tools > Content Editor**
4. To start off, immediately go to the upper left hand corner and choose **File > Save**
5. Save the file under a name of your choice, in the *Libraries* directory within your mod's folder hierarchy (e.g. *Castle_Meshes.clb*)
6. You can see some default tabs of the resources that can be imported. If you want to enable tabs for different types of resources, you can do so under **Options > Change Features**. For this example we won't need the *Models* feature, so go ahead and disable it by selecting the *Models* entry under *Selected:* and moving it to the right of the list by clicking the **>>** button
7. You should now be in the *Meshes* tab. If you are not, **select** the tab by clicking on it
8. Now, click on the **Import** button to start importing meshes
9. **Navigate** to the *Castle_Mesh.fbx* file in the example files directory we have provided (or use an .fbx of your choice!). Open the file. It's worth noting that *.dae* is still supported as a file format, but it's preferable to use *.fbx*. If you run into importing errors, make sure your mesh is both **triangulated** and that it has a **material**
10. **Switch** to the *Textures* tab by clicking on it

11. Again, click on the **Import** button to start importing textures
12. **Navigate** to the *Castle_Diffuse.png* and *Castle_Normal.png* files we have provided, or use some textures of your own. The normal is not required, but we're leaving it here for completeness sake.
13. You can choose to **rename** the texture files, which we will be doing in this case. You can either select the new content and press **F2** to rename it, or change the name in the input box on the right hand side (in the panel with all the properties). In this case we'll rename the entries to *Castle_Diffuse* and *Castle_Normal* respectively.
14. Select the *Diffuse* texture and change the *Color Space* property to **sRGB**. This is required for proper colors on your diffuse texture. The normal map should remain in the **Linear** color space.
15. Next, **switch** to the *Materials* tab
16. While having *Default* selected under the *Material Type:,* click on the **New Material** button
17. **Rename** the new *noname* material to something like *Castle_Material*
18. **Copy** the content by pressing **CTRL+C** while having the content selected
19. **Select** the row below the currently selected content and paste a copy by pressing **CTRL+V**.
20. **Rename** this copy to *Castle_Material_Red*
21. Under the *Diffuse Map* separator, **change** the *Library* property of the *Diffuse Map* to *Default { NAME_OF_YOUR_CLB }*
22. Next, select the imported *Castle_Diffuse* texture under the *Resource* property of the *Diffuse Map*
23. Do this same process for the *Normal Map* property, but select the *Castle_Normal* texture that we've imported earlier.
24. **Go back** to the *Meshes* tab and select the previously imported mesh
25. **Expand** the *Default Material* property in the right hand side panel
26. Under the *Library* property, **select** *Default { NAME_OF_YOUR_CLB }* again
27. **Change** the *Resource* to the *Castle_Material* material we've created earlier. You should now see the material being applied to your mesh in the viewport.
28. **Right-click** the *Meshes* tab and choose **Lock Viewport**. This makes it so that the viewport stays open as if we're in the *Meshes* tab, while navigating through the other tabs. To unlock the viewport, **Right-click** on the locked viewport tab and choose **Unlock Viewport**.
29. **Rename** the imported mesh (using the same rename principles as mentioned above) to *Castle_Feudal*.
30. **Copy** the content by pressing **CTRL+C** while having the content selected
31. **Select** the row below the currently selected content and paste a copy by pressing **CTRL+V**.
32. **Rename** this piece of content to *Castle_Barbarian*
33. **Change** the material of this copied mesh to the *Castle_Material_Red* material we've created earlier
34. While **keeping** the *Castle_Barbarian* content **selected**, **switch** back to the *Materials* tab
35. **Select** the *Castle_Material_Red* content
36. Under the *Colorize* separator, **click** the white square next to the *Colorize* property

37. In the dialog that appears, **click** on the white square again to open a color picker
38. **Pick** a nice color for your material. You should be able to see the material update in realtime as we have *locked* the viewport of the mesh with the material that we're editing.
39. **Save** the content library by either pressing **CTRL+S** or going to the upper left hand corner and choosing **File > Save**. There's an entry in the log under the viewport if saving was successful.

We have now successfully created a *Content Library* that we can use throughout our *Resource Packages*! Of course, this content will not appear in the game yet until we have linked it up somewhere; so let's do that now.

Step by Step Guide

1. In the *Package Manager*, go to **Tools > Resource Editor**
2. In the upper left hand corner, go to **File > Open**
3. Within the root directory of the base game, **navigate** to *Content\Title\Packs\Strategic\Architecture* and open *Architecture_Feudal_Strategic.rpk* and *Architecture_Barbarian_Strategic.rpk* by **multi-selecting** them (**CTRL + click**)
4. In the opened pack, in the upper bar, go to **Content > Libraries**
5. **Click** the **Add** button and find the *Castle_Meshes.clb* content library we've created earlier (in your Mod's *Libraries* directory). Open it.
6. **Click OK**, the resource pack should now refresh
7. Now, **change** the *Category:* to *Modularity: City*
8. **Select** the *Component: City Core T1* resource
9. **Right-click** the resource and choose **Create Mod**
10. **Select all meshes** under the *Mesh* tab in the right hand side panel and **delete** them by pressing **Del** on your keyboard (or **Right-clicking** and choosing **Delete**)
11. **Right-click** the *0th* entry in this same list and choose **New** (or press **CTRL+N**)
12. Next, **expand** the *Mesh* property on the right and find the mesh in our created content library (**Library =** *Default { CASTLE_MESHES }*, **Resource =** *Castle_Feudal*)
13. **Save** the pack (**CTRL+S**)
14. **Repeat steps 4 to 13** for the *Architecture_Barbarian_Strategic* resource pack

It's time to test our mod! **Boot up** the game with the proper *Playset* of your mod. **Create a new faction** using the *Feudal* culture and **start a new session**. Your city should be replaced with the castle mesh. If you start a game using the *Barbarian* culture, you will ideally see the mesh with the different material that we've created earlier.

FBX Notes

We use the *.fbx* file format (as well as *.dae*, but it has been deprecated during development) to import meshes into our *Content Libraries*. As *FBX* can have problems with parity between different applications that interpret it, we wanted to elaborate a bit on how you can make sure that your exported meshes are correctly imported into our engine.

Some things worth mentioning:

- We have a *right-handed*, *DirectX*-based, coordinate system
- In our engine, the axes are defined as follows:
    - **+Y-axis**: *Up axis*
    - **+Z-axis**: *Forward* axis
    - **+X-axis**: *Right axis*
- We support the following from the *FBX* specification:
    - **Meshes**
    - **Material groups**
    - **Skeletons/rigs**
    - **Skinning** (with a maximum of 4 *weight influences*)
    - **Animations**
    - **Vertex attributes** (e.g. *color*, *custom vertex normals*, 2 *UV sets* – of which the 2nd is for overlays, etc.)
    - **Pivot points**
- Some of the things that we ***do not*** support:
    - **Lights**
    - **Cameras**
    - **IKs** (should be baked into the animation keyframes)
    - **Animation Curves** (curves should be baked into keyframes)
    - **Transformations** (these should be applied/baked into meshes or animation)
    - **Materials/shaders** (we set up materials from within our engine)
    - **Quads** (Triangulate your meshes; the importer *should* do this automatically)
    - **Quaternion integration of animation keyframes** (these should be resampled as *Euler*)

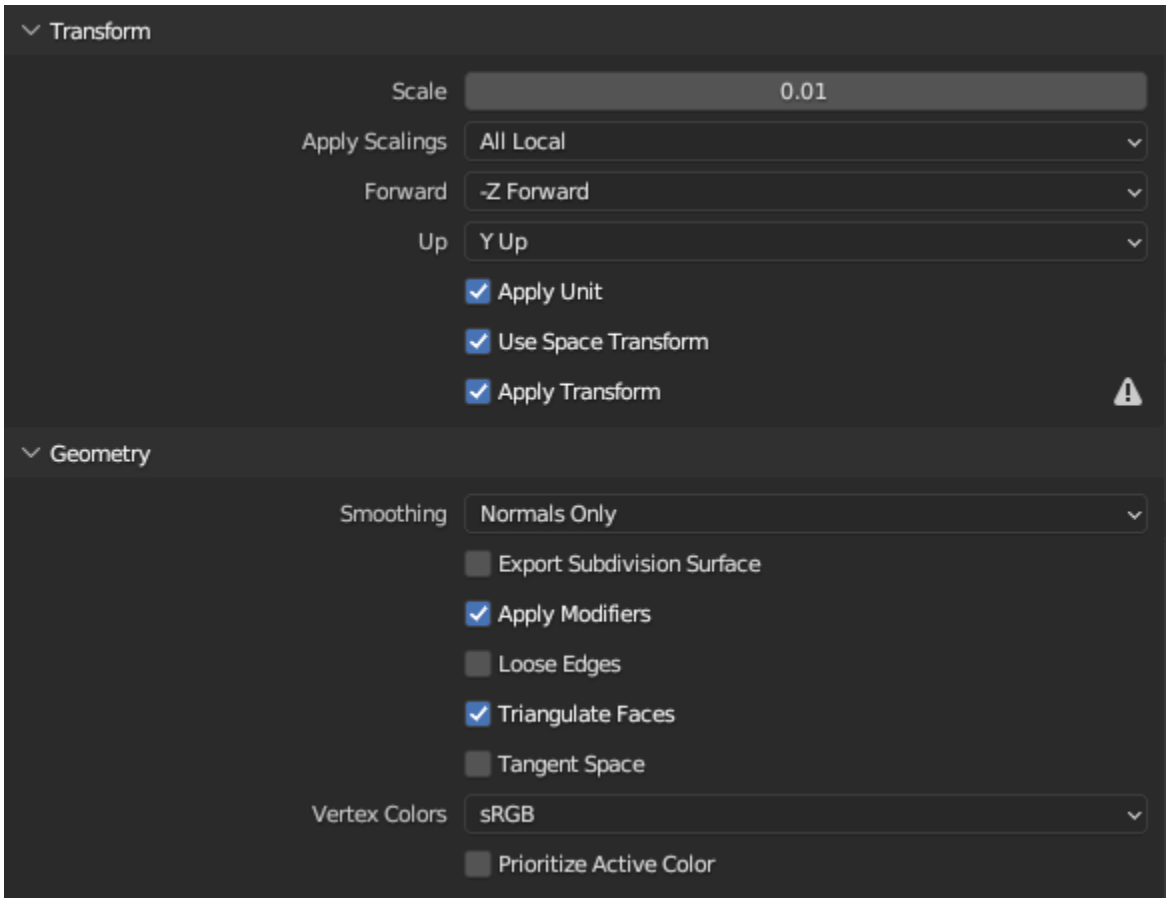
Blender has been used to test out modding compatibility, but anything that exports to the *.fbx* file format can be used. During development, *Autodesk Maya* was used, which has some implications in regards to the orientation of bones in a skeleton when compared to *Blender*.

Preparing For Export - Regular Meshes

To export a mesh from *Blender* there are some things to keep in mind:

- Make sure all transformations are applied
- Make sure your pivots are set correctly
- Align your meshes as follows:
    - **- Y-axis** = Z-axis (*forward*) in our engine
    - **+Z-axis** = Y-axis (*up*) in our engine
    - **+X-axis** = X-axis (*right*) in our engine
- A material is *required*, as we use them to split up meshes on a per-material basis
- Make sure your units are set to *meters*

When you are ready to export your mesh, go to **File > Export > FBX (.fbx)**. You can now save the file somewhere, but before you do; make sure the settings are as follows (other settings might work – especially if you follow your own orientation rules, but these settings have been used for testing given the alignment mentioned above). These settings were available in *Blender 3.5*, but they are also applicable for earlier versions.



The *Bake Animation* section should not be relevant for non-animated meshes. Furthermore, what is included in the export is up to yourself; do note we only support *Meshes* and *Armatures* for export. *Vertex Colors* can also be changed to be linear if required.

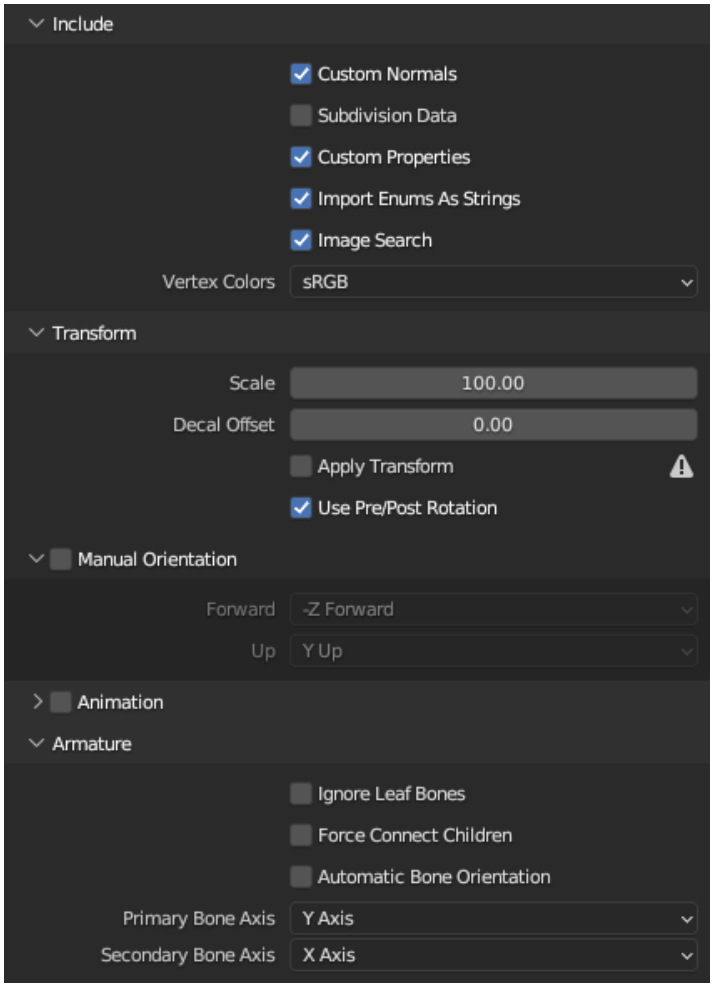
Importing - Rigs

In our example files we have included some importable *.fbx* files for the different *rigs* that we use in the game. This means you won't have to completely re-create the skeletal hierarchy in your 3D editing software if you were to export *skinned* bones.

It's important that the hierarchy is exactly the same in the .fbx file as that it is expected to be in the game. Our mesh to bone mapping is *index-based*, so bones need to be in the same place in the hierarchy for animation skinning to work. Furthermore, we do some filtering of our rigs through how their names are written out in the .fbx file (e.g. the _F_ and _K_ prefixes). It is *possible* to create your rigs from scratch, but it is not recommended as this means you would

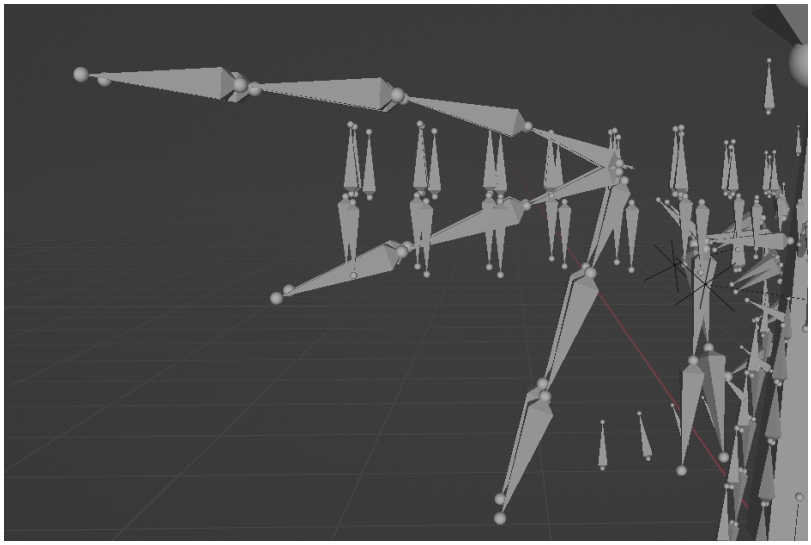
also have to go through the process of setting up our animation systems from scratch. The idea here is to re-use existing rigs and animations for when you only want to swap out some meshes in your mod. Again, it is possible to create a new rig, animations and meshes that are skinned to it, but we currently don't have any support documentation for this.

To import a rig, go to **File > Import > FBX (.fbx)** and navigate to the appropriate file of the rig that you want to import. Use the following settings:

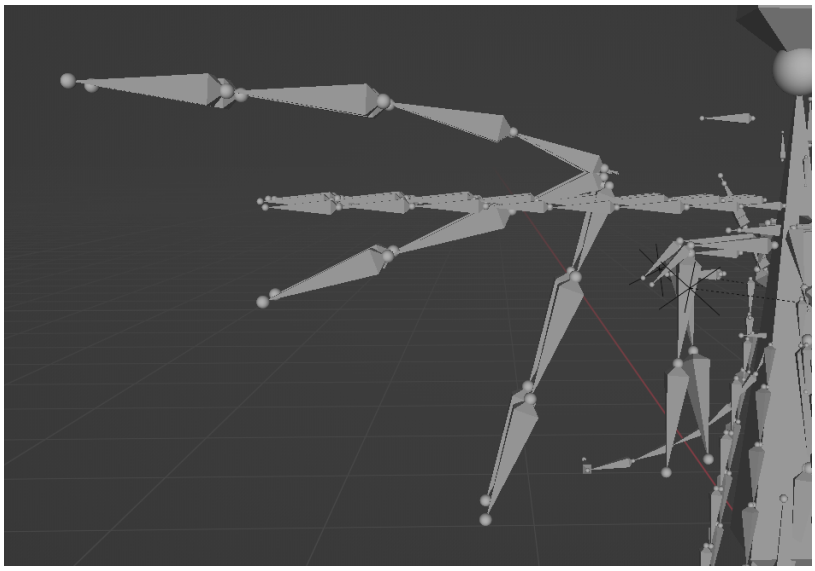


The most important part here is that *Automatic Bone Orientation* & *Ignore Leaf Bones* remains unchecked under the *Armature* section. This makes sure that the bone orientations stay as-is.

Lastly, it's worth mentioning that these rigs are not very useful to use for animation in *Blender*. The bone orientations of our rigs don't have a *leading axis* – we purely look at the orientation of a bone in the bind pose and deduce the bone length from child to parent. In *Blender*, the leading axis of a bone is always the local *Y-axis*. This means that the visualization in the viewport of *Blender* might look a bit misleading, but this is unfortunately a limitation we have to deal with. When using *Autodesk Maya*, the bones will import correctly as-is.
As we're only storing skinning information about bones, this should be workable enough; but for animation it's practically impossible.



*Example of the bone orientations of Humanoid_Rig when imported **correctly***

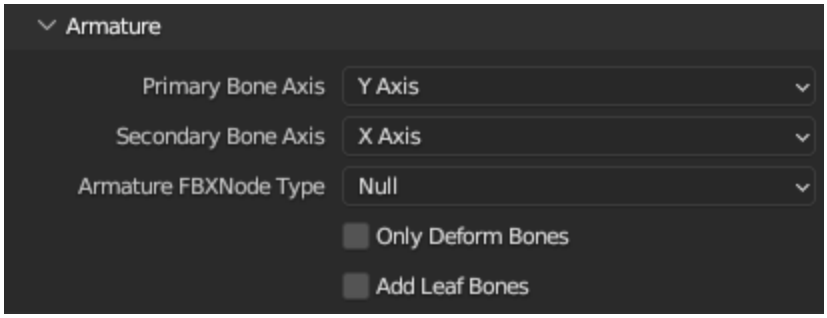


*Example of the bone orientations of Humanoid_Rig when imported **incorrectly***

Preparing For Export - Skinned Meshes

After skinning meshes to bones, you can now export these meshes and the respective armature that goes along with it.

The export settings are exactly the same as for regular meshes; however, the armature settings should remain like this:



Be sure you have included both *Armature* and *Mesh* in the *Include* section!

FBX - Validation

If you want to validate whether your skinned mesh was exported correctly, try and follow the **Adding Leader Customization Items** section using your new *.fbx*. If you want to validate a regular mesh, try out the **Adding Content Libraries** section!

Modularity System Overview

Throughout the game, we use the so-called *Modularity* system. For example, the guide on **Adding Content Libraries** shows you how you can set up a *City* structure for both the *Barbarian* and *Feudal* culture, while sharing the same mesh content, but using different materials for each. Be sure to check out that guide as well!

Modularity - Design

The idea behind the *Modularity* system is that we can re-use our content and combine them in different ways. Flexibility is key here. In a way the system can do very generalized processing of visuals. There are no hard dependencies on gameplay logic. For instance, all of the humanoid units in our game share the same rig, but they're scaled differently through so-called *Rig Modifiers*. These modifiers are applied in the *Modularity* system based on the unit's *Providers*. However, these *Providers* only tell us what a unit is supposed to look like, they're not strictly linked to the unit. You could create the unit visually while not having any information about the Unit's gameplay data. We keep track of all the visuals we want to present in a *Modularity Object*.

Some examples of these providers are – but are not limited to:

- **Skin**
- **Culture**
- **Unit Type**
- **Gender (where applicable)**
- **Customization**

Modularity - The Modularity Object

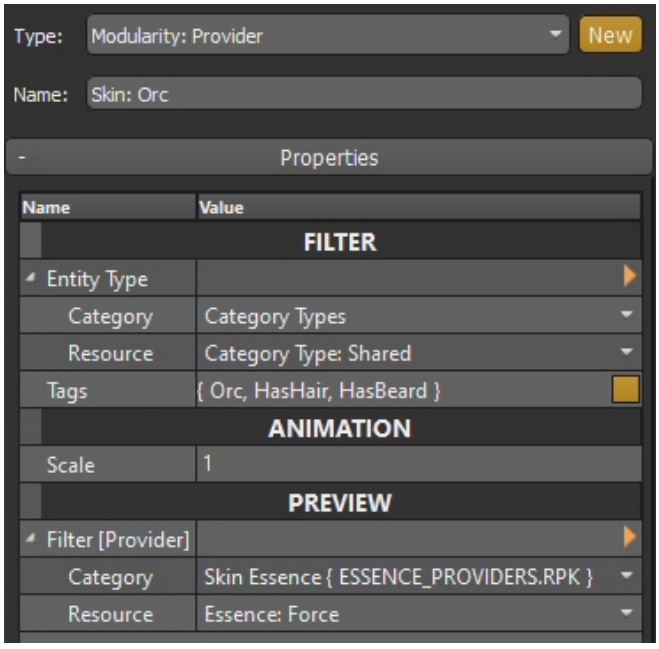
To bring the visuals together, we have a generalized object that a lot of objects (not all) in our game use. The properties that are read out from gameplay logic are variable based on what type of object we're dealing with. For instance, we do quite a bit of *Modularity* logic in *Units*, but way less so in *Structures*. To keep track of what we want to show, we store so-called *Providers* in the *Modularity Object*. These *Providers* are linked to a respective *Modularity Object* in the resource packs.

Modularity - Providers

These are resources of the *Modularity: Provider* resource type. *Providers* are the "modular pieces" to our system. For instance; a *Feudal Toad Pikeman* unit's *Providers* could look like this:

- **Culture: Feudal**
- **Unit Type: Pikeman**
- **Skin: Toad**
- **Essence: Stout**
- **Gender: Male**

Providers inject so-called *Tags* into the *Modularity Objec*t. These can be used later by *Components* to filter certain parts of the visuals. Here is an example of the **Skin: Orc** *Provider*:



As you can see, whenever this *Provider* is added; the "Orc", "HasHair" and "HasBeard" *Tags* are included into the *Modularity Object*. In this example, we for instance make sure that during customization, hair and beard sliders can be used to customize the leader. Of course, this is just an example; we use *Tags* in a lot of different ways. It boils down to giving us a generalized way to **conditionally enable and disable content**.

Modularity - Entity Types

These types (*Modularity: Entity Type* in resource packs) are used to distinguish between what sort of object we're targeting with our *Providers*. The **Culture: Feudal** provider is not only used in Units, but also in Cities. If there are *Components* that are unlocked through **Culture: Feudal**, we only want to allow these if they're of the same *Entity Type* as the original object (e.g. We don't want to include a city's particle effects that are specifically unlocked through the **Culture: Feudal** provider if we're dealing with a Unit object). Quite often we use the **Category Type: Shared** as *Entity Type*. This ignores this type of filtering functionality.

How this works is that all *Providers* for an object are gathered, after which the first *Entity Type* we encounter becomes the **main** *Entity Type*. If another *Provider* unlocks components of a type that is not the same as the main type (and not shared), they will be **disabled**. The **order** in which *Providers* are added determines their **priority**. If for instance you add two *Providers* that have a different *Entity Type*, the *Entity Type* of only the first *Provider* is used, filtering out the other.

As you can see in the example screenshot on the previous page, there is an *Entity Type* property within *Providers*. This is only here to determine the main type as mentioned above, further filtering happens in the *Components*.

Modularity - Components

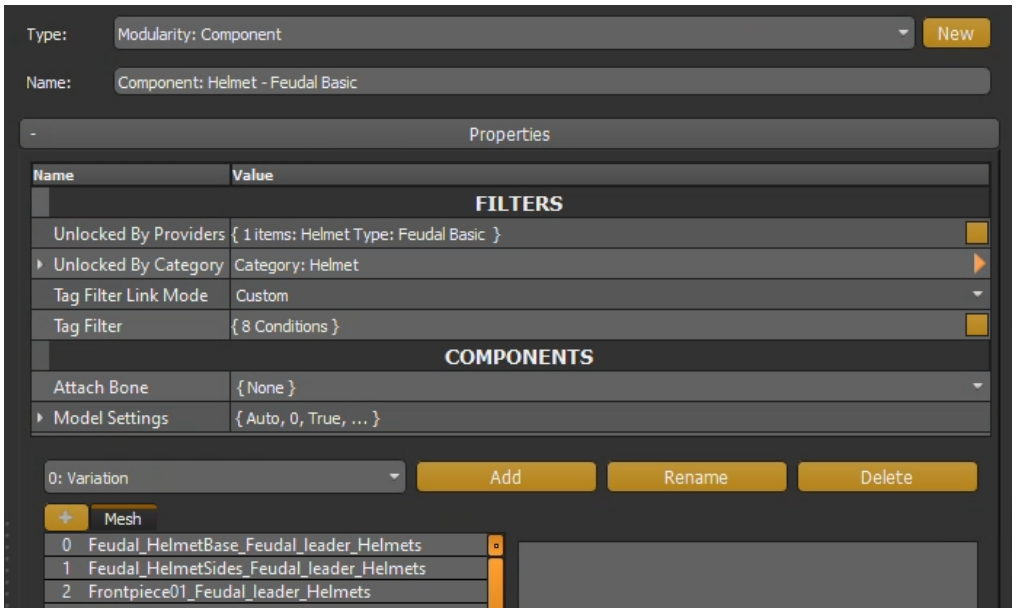
These are the types of resources (*Modularity: Component* in resource packs) that add the final visuals/visual dependencies to the *Modularity*-based object.

The different types of visuals that can be included in a component are the following:

- **Mesh**
- **Particle effects**
- **Animation data (rig, rig modifier, animation layer)**
- **Icon data (for rendering portrait icons)**
- **Decals**
- **Sound effects**
- **Colliders**
- **General variables (mostly unused!)**
- **Attachment nodes (for particles on weapons, for e.g. enchantments)**

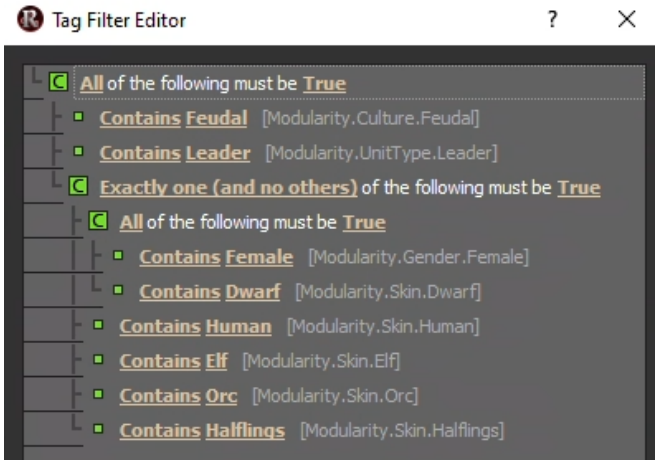
Components contain a *Component Category* (*Modularity: Component Category* in resource packs), which are used to determine what *Entity Type* the component applies to and by which *Providers* the *Component Category* is unlocked. *Component Categories* are then used to determine which component takes **precedence** over which. If two *Components* are unlocked, but they share the same category; either one or the other is picked **randomly**. This can be used to give small variances to the *Modularity* object you're building. If the *Entity Type* of the category doesn't match the **main type** of the currently processed object, the components are **disabled**.

Components - Examples
Here is an example of a component for a *Feudal Helmet* (*Content\Title\Packs\Unit_Modularity\Culture_Feudal_Modularity.rpk* > **Component: Helmet - Feudal Basic**) customization option:

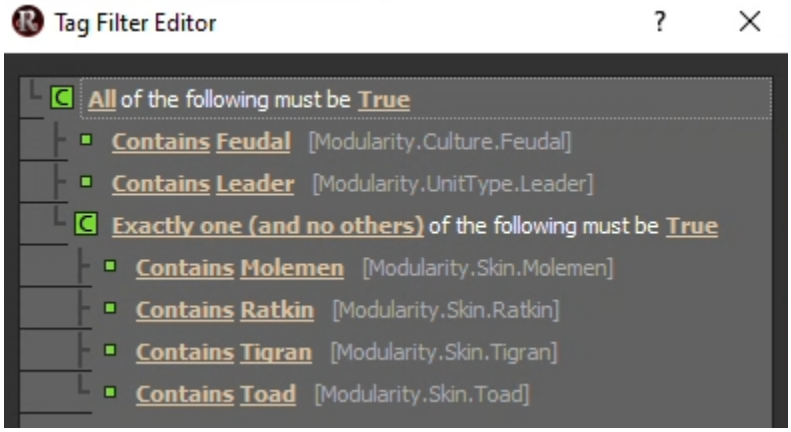


As you can see, the helmet has its own *Provider*. If we add **Helmet Type: Feudal Basic** to our *Modularity* object, we'd get the meshes under "Mesh" added to our visuals. Again, this only happens if the *Entity Type* of the **Category: Helmet** matches the original main type of our object. (In this case, **Category Type: Unit**) If we add this *Provider* to a *City*, the helmet will not appear.

Components - Tag Filter
A *Tag Filter* can be specified to make sure that the *Component* only shows up under specific circumstances. The *Tag Filter* of this specific resource looks like this:



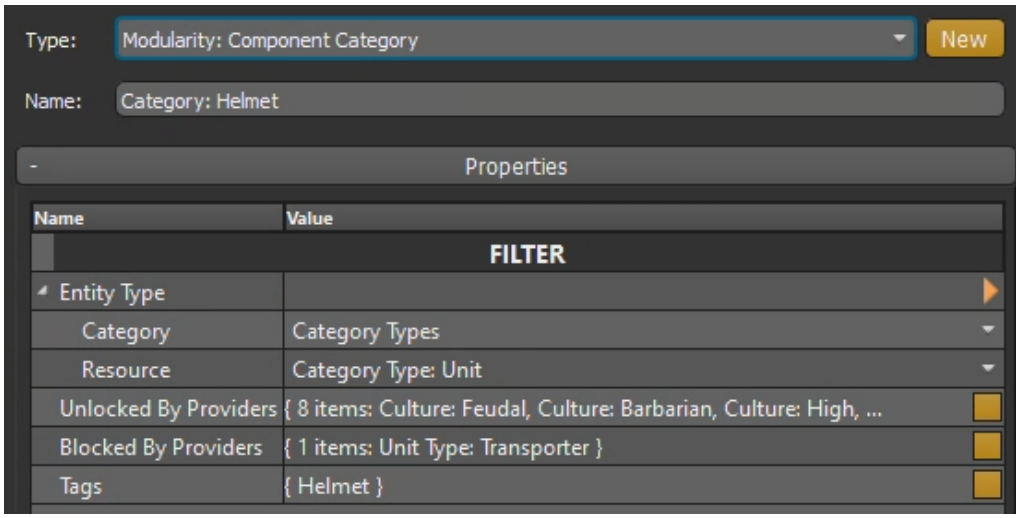
A similar resource (**Component: Helmet - Feudal Basic no nose and sides** in the same pack) then has exactly the same set up, except for the *Tag Filter* and *Meshes* it provides.



This means all of our Feudal Leaders – that are either a *Moleman*, *Rat*, *Tigran* or *Toad*, will use the "*no nose and sides*" version. *Female Dwarves*, *Humans*, *Elves*, *Orcs* and *Halflings* will use the **original helmet**. However, both of these can simply be added through the **Helmet Type: Feudal Basic** *Provider*. The system will automatically resolve the filters.

Components - Component Categories

Taking a look into the **Category: Unit** *Component Category* reveals the following:



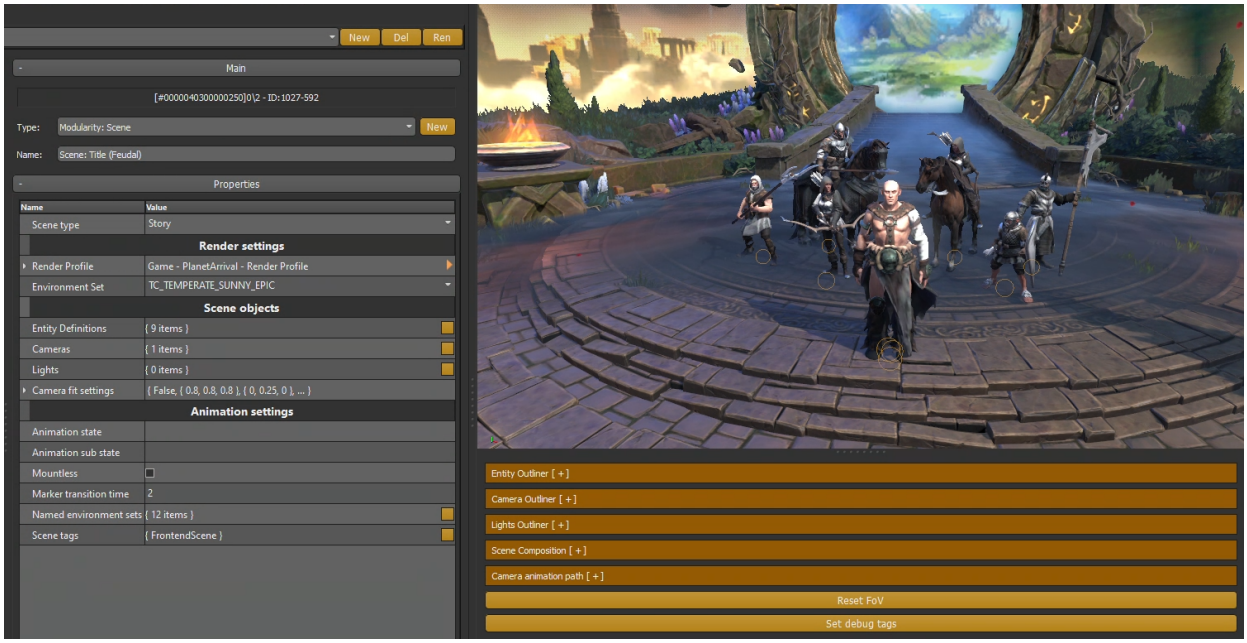
As you can see, it is only applicable for the **Category Type: Unit** *Entity Type*. Furthermore, *Unlocked By Providers* determines which *Providers* unlock this respective category. Not only can we **include** *Providers*; *Blocked By Providers* makes sure that even though a specific *Provider* has the correct *Entity Type* linked, it is still filtered out. (In this case, we don't want the helmet to display when on a boat – we don't show any units on boats at the time of writing this guide). Lastly, you can push in more tags into the *Component Category* if need be. In this case, we add the extra *Helmet* tag.

Modularity - Scenes

There's a multitude of methods to determine how your *Components* interact with each other, but *Scenes* are probably the easiest way to check if everything is working correctly in your *Modularity* setup.

Scenes are specifically used in-game for presenting screens with multiple *Modularity Objects*. These objects receive their *Providers* from code or from resources. We won't be going into too much detail on how to use these *Scenes* for their in-game purpose, but they're also a very powerful **debug tool**.

For this example we'll be opening the *Content\Title\Packs\Frontend_Scene.rpk* pack. **Select** the *Scene: Title (Feudal)* resource, you should see something like this:



You can navigate the *Scene* with the default camera controls. If you want to **Move**, **Scale**, or **Rotate** an *Entity* you can do so by selecting it in the **Entity Outliner** or by simply clicking the orange circle at the *Entity*'s pivot point. A *Gizmo* will appear.

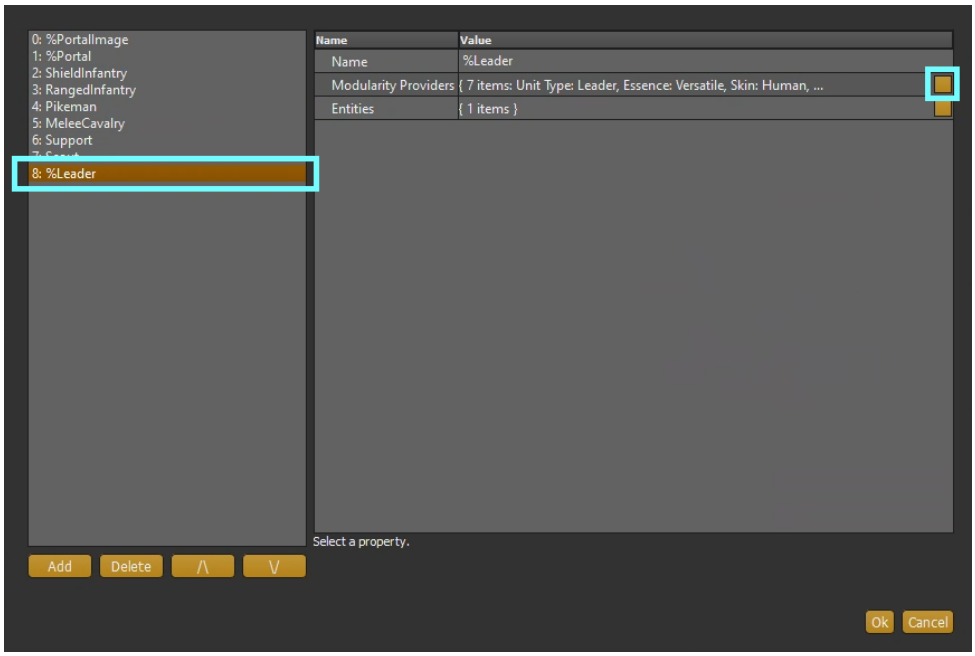
With the *Gizmo* open you can cycle through the different transformation modes with the following keys:

- **T** = Translation, offset the entity positionally
- **E** = Scale, change the size of the Entity
- **R** = Rotate, change the orientation of the Entity

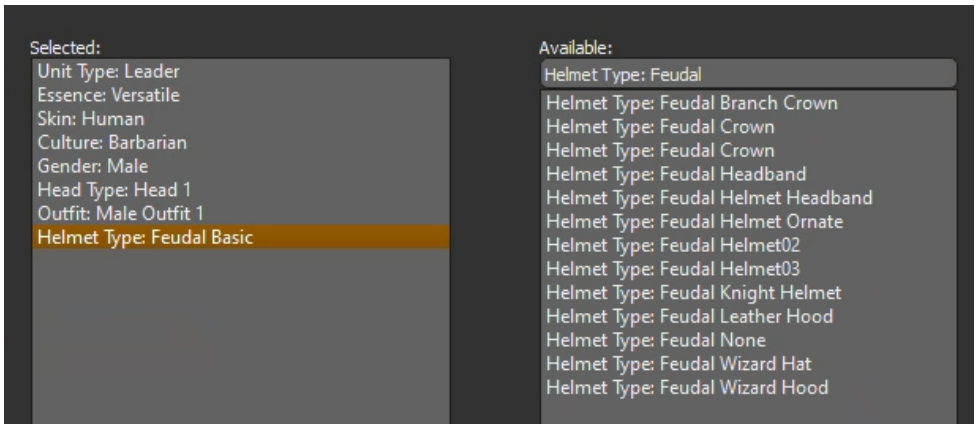
Furthermore, you can press **B** to view all bones of each *Entity* in the view (not very performant!). There's of course a lot to do here, but what we're mostly interested in is viewing the Modularity objects that we've created or are yet to create. Please feel free to fiddle around with the tool!

Here's a step-by-step guide on how you can test your *Providers*:

1. Open the **Entity Definitions** panel by clicking on the yellow/orange square to the right-hand side of it.
2. **Select** the *%Leader Entity Definition* and open its *Modularity Providers*



3. **Search** for *Helmet Type: Feudal Basic* and add it to the list



4. As you can see (if you click **OK** twice to exit this interface), we still do not get a helmet. That is because the helmet is for the *Feudal* culture, not for *Barbarians*. So our *Tag Filter* doesn't pass.
5. **Go to** the *Modularity Providers* of *%Leader* again, **remove** the *Culture: Barbarian* provider and **add** the *Culture: Feudal* provider instead. You should now see the armor and helmet change!

You can of course create a *Scene* from scratch, just be sure to link the correct resource packs containing your *Providers*!

Modularity - Linking Providers

If a system supports our *Modularity* system, you usually have the ability to **link** providers to a respective resource in the resource packs. In the *Scenes* example above we linked different *Providers* to the preview entities of the *Scene*. While we will not see these changes in the game, this is exactly how you would also change the *Providers* of a different resource. However, there are also circumstances in which *Providers* are gathered from settings resources and then combined into one *Modularity Object*. This is the case for *Units*, for instance.

A big difference between how *Modularity* resources and regular resource links work, is that *Modularity* resources are not technically linked together. *Providers* know what *Entity Types* they unlock, they don't know about their *Components*. Instead, *Components* know what *Providers* unlock them. In the end, we only link *Providers*; you **cannot** link *Components*. It is a bit difficult sometimes to keep an overview of this, but the *Resource Browser* can be a powerful tool to find where *Providers* for *Components* are defined.

Adding Text

If you want to add new things like units or abilities into the game, you will also need to add new text into the game so that those units and spells have distinct names and descriptions. For handing text in Highlander we recommend the usage of POEdit.

Create a POT Template file

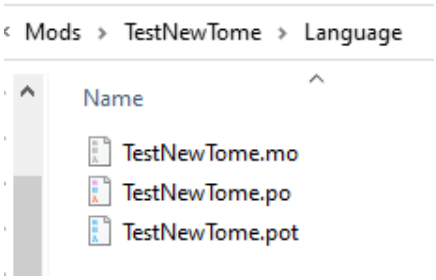
1. Create an empty text file such as Notepad.
2. Add the following lines:
    a. *msgid "[IDNAME]"*
    msgstr ""
    b. The first line is the text ID that is used to identify which text should be used. The second line is what the text will be that is used when you link the TextID.
    c. If you want more than 1 string, simply add another 2 lines "msgid "[IDNAME]"" and "msgstr """ for each extra string.
3. Save the file as *[MODNAME].pot*
    a. You can also change the extension to **.pot** after saving it as a text file.

This file is the template that you will use whenever you need to add new text IDs

Tip: Triumph usually formats string IDs as follows: [rpkname@resourcename@element], which can look something like:
"UNITPROPERTIES@POISONED@DESCRIPTION"
This allows you to quickly identify where the string should end up and what its purpose is.

Create PO Project

1. Launch **POEdit**
2. Click on **Create new…** (Create new translation from POT template)
3. Select the Template made in the previous step
4. Save the po file in the Languages folder of your mod as *[MODNAME].po*
5. Start or refresh your package manager. You will see three files show up: a **.po** file, a **.mo** file and a **.pot** file.
6. Make sure the .mo file is named *[MODNAME].mo*.



This file is the actual localisation project that you will use after adding text IDs to add new text entries.

Adding ID's

1. Right click the .pot file and open it with a text editing program like Notepad.
2. Add the text IDs with empty message strings like you did before:
    a. msgid "MOD@PACK@RESOURCE@NAME"
    msgstr ""
3. Save and exit.
4. Open the **.po** file using **POEdit**.
5. Click on **Translation > Update from POT file** and double click the **.pot** file.
6. The new text IDs are now added.

Adding Text to IDs

1. Open the **.po** file using **POEdit**.
2. Click on one of the text IDs you want to add text to.
3. At the bottom is a text box for translations. Insert your text here.

Adding Text IDs to Resources

1. Whenever a resource needs text, it will have the display **{ Not Set }**.
2. Click on the yellow cube to open a window called **Localized String Dialog**.
3. In the empty text bar at the bottom, add the text ID, for example: MOD@PACK@RESOURCE@NAME.

Adding the MO to your Mod

Lastly, to link your text to the project, do the following:

1. In the Resource Editor, go to **Content > Languages > Add** and add the **.mo** file.
2. Do this for each *.rpk file* in your mod that you want to link your text to.

Adding a New Tome (Example Mod)

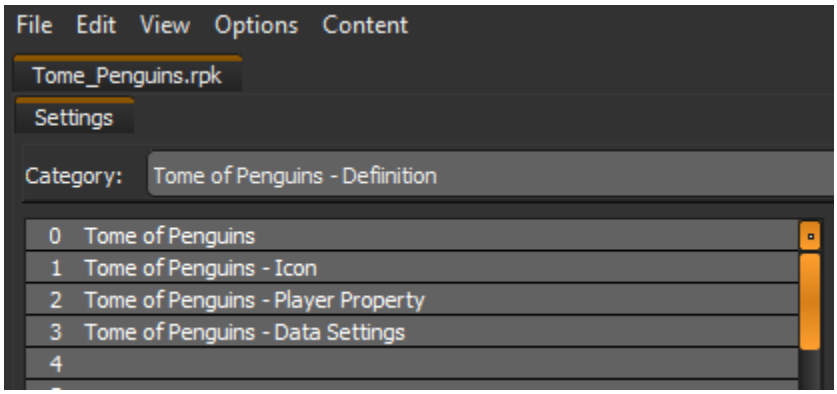
For this guide, we will be making an example Tome that you can download called the Tome of Penguins and is available on Steam Workshop and online alongside this guide.

All these steps will assume you will be making a Tome of Penguins, but the specific names or resources used can be anything you want.
Make sure to read up on Adding Text.

Tome Setup

1. In the package manager, open the resource editor and make a new package, save it as *Tome_Penguins*.
2. Add the package to the Package Manager via **Content > Resource Packs > Add**.
3. Go to Content > Resource Packs > Add, and add Core_Research found in *Title\Packs\Research*.
    a. This contains all the information all Tome packs need to know about such as research cost and icons.
4. Add a new **category** and name it *Tome of Penguins - Definition*.
5. Add the resource **AoW Skill Grid: Group** and name it *Tome of Penguins*
6. Add the resource **AoW Skill Grid Group: List Data Settings** and call it *Tome of Penguins - Data Settings*

    a. Add the Tome of Penguins to the **Skill Group: Global** and **Frontend Skill Groups**. This is what actually makes the Tome of Penguin show up as a possible Tome to choose during gameplay and during faction creation.
7. Add the resource **Icon (Interface)** and name it *Tome of Penguins - Icon*.
    a. Click on **Image > Resource > Resource** and select **Tomes**.
    b. Click on **Sub Resource** and select **Tomebook_Chaos_Pandemonium**.
8. Add the resource **Player property** and name it *Tome of Penguins - Player Property*. This can be used as an identifier to unlock certain content associated with the Tome.
9. There are a bunch of settings in this resource that need to be filled in in the **Tome of Penguins**.
    a. **Source Name**: *Tome of Penguins*
    b. **Gameplay Description**: *The forbidden Tome of dire Knowledge, containing the foulest spells and the most horrid creatures.*
    c. **Icon Link**: Tome of Penguins - Icon
    d. **Icon Opened Link**: Tome - Chaos: Opened
    e. **Glyph Image A**: Tome - Chaos: Glyph A
    f. **Glyph Image B**: Tome - Chaos: Glyph B
    g. **Initial Player Properties:** Link to **Tome of Penguins - Player Property**.
    h. **Tier**: Tome - Tier I
    i. Right click **Hero Affinities > New > Chaos - Affinity Definition**, put the affinity amount to 2. (Despite the name, this is the affinity given to the Empire, not to Heroes)



You can test the mod now by publishing and enabling the mod. In faction creation, the Tome of Penguins should show up as one of your starting Tomes.

Adding Research Skills

Research Skills are the pieces of content you can unlock via Arcane Research using Knowledge. This includes research from Tomes, Cultures and general research. All work using the same setup.

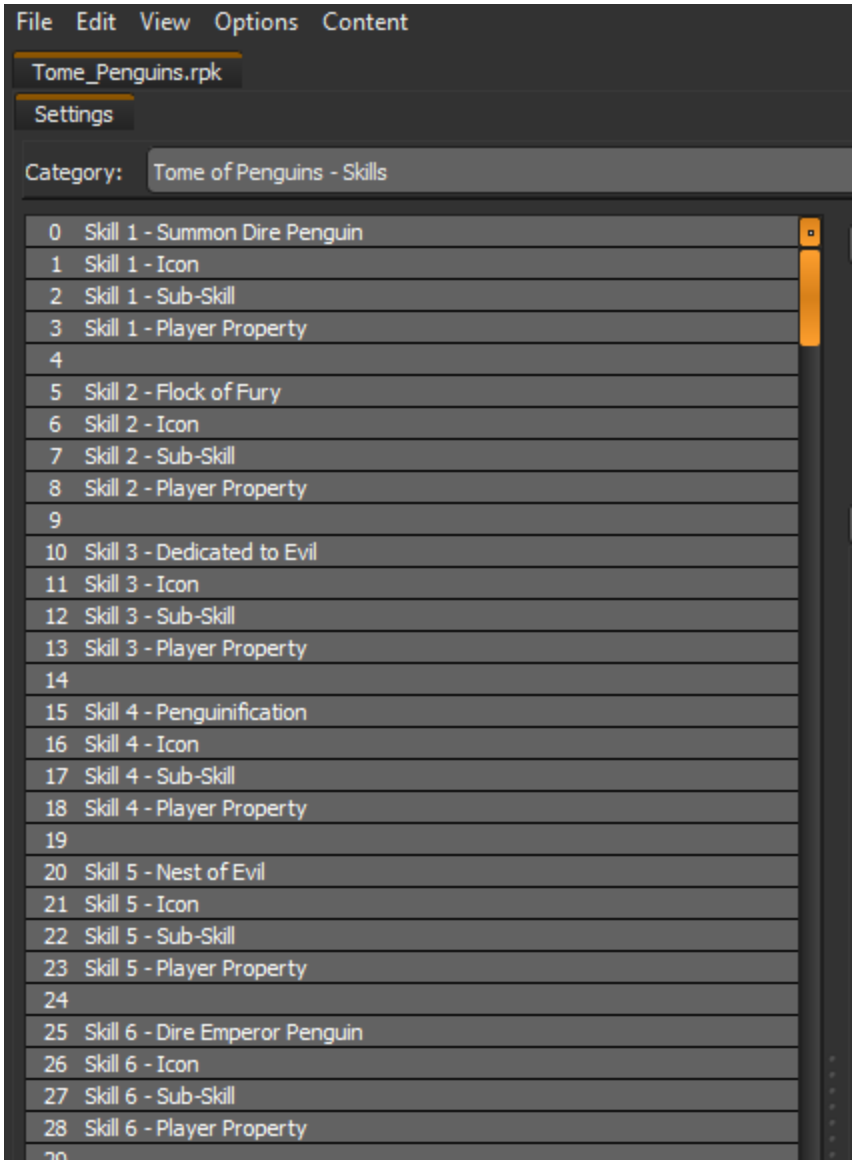
1. In the Tome_Penguins.rpk, add a new **Category** and name it *Tome of Penguins - Skills*.
2. Add the resource **AoW Skill Grid: Skill** and name it *Skill 1 - Skill*.
    a. **Screen Name**: Name used for the skill. This must always be filled in or it will cause errors!
    b. **Screen Description**: The description used for the skill.
    c. **Screen Type Link**: This is displayed when looking at the skill to inform the player what type of skill it is, it has no effect on gameplay and can be left empty.
    d. **Tier**: This determines the Knowledge cost of the skill.
    e. **Group Link**: This links the skill to your Tome. So link this to *Tome of Penguins*.
    f. **Requisite List**: These are hidden identifiers used for logic such as discounts.
    g. **Can be Starting Skill**: Choose a single skill in your Tome and set this to True. When this Tome is chosen as a starting Tome. That skill will already be unlocked.
    h. **AI Priority Resource Link**: Used for AI to determine how valuable this skill is. By default, set this to **Research AI Priority - Medium**. The skill won't work if this is empty.
3. Add the resource **Icon (Interface)** and name it *Skill 1 - Icon*.
    a. Choose any icon.
    b. In **Skill 1 - Skill**, go to **Large Icon Link** and link this icon.
4. Add the resource **AoW Skill Grid: Sub-skill** and name it *Skill 1 - Sub-Skill*.
    a. **Type**: This determines how the UI displays this skill. This needs to match up or the skill won't work.

        i. **Operations** are used for all Spells, including Unit Enchantments and Transformations. And for Siege Projects.
        ii. **Colony Upgrade** is used for City Structures.
        iii. **Unit Unlock** is used for Units.
        iv. **Global Bonus** is used for most other things like passive bonuses.
    b. In **Skill 1 - Skill**, go to **Sub Skills** and link this sub skill.
    c. Multiple Sub-skills can be given to unlock multiple things with a single skill.
5. Add the resource **Player Property** and name it *Skill 1 - Player Property*.
    a. In **Skill 1 - Sub-Skill**, go to **Player Properties** and link this Player Property.
    b. This property is the identifier for a player that will actually unlock content. Everything before this is just to give this player property to the player via research.

Now if you test the mod, there will be a single skill displayed in the Tome. You can research it and get the player property **Skill 1 - Player Property**. This is an invisible property and currently unlocks nothing.

A tome should consist of 4 to 6 skills. Every 4 skills a player unlocks a new tome. So giving fewer than 4 skills will result in a possible dead end of your research. Having more than 6 skills will result in UI and balance issues.

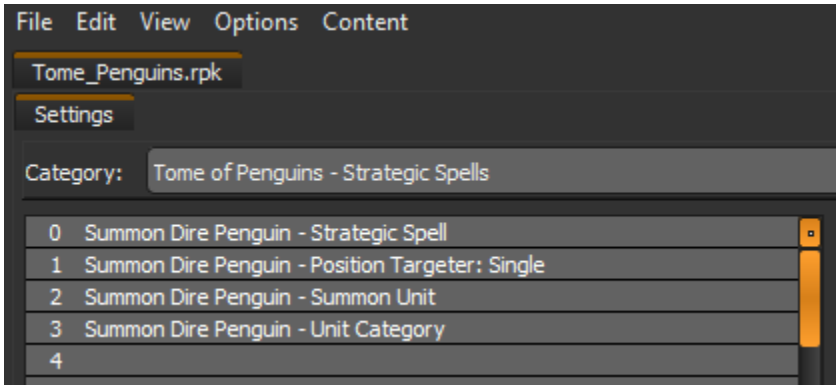
6. Repeat this process 5 times to get a total of 6 skills for this tome.
    a. **Tip**: You can copy paste the setup of 4 resources a few times, then rename and relink everything. So *Skill 2 - Player Property* will be given by the *Skill 2 - Sub-skill* which is given by the *Skill 2 - Skill*.
    b. **Tip 2**: You can search and replace resource names using the CTRL + SHIFT + F shortcut. It will only apply to all the resources currently selected (those that turn yellow).
    c. For the rest of this example tome, the skills will be as follows:
    Skill 1: Strategic Spell - Summon Dire Penguin
    Skill 2: Tactical Spell - Flock of Fury
    Skill 3: Unit Enchantment - Dedicated to Evil
    Skill 4: Transformation - Penguinification
    Skill 5: City Structure - Nest of Evil
    Skill 6: Unit - Dire Emperor Penguin


File Edit View Options Content
Tome_Penguins.rpk
Settings
Category:
Tome of Penguins - Skills
0 Skill 1 - Summon Dire Penguin
1 Skill 1 - Icon
2 Skill 1 - Sub-Skill
3 Skill 1 - Player Property
4
5 Skill 2 - Flock of Fury
6 Skill 2 - Icon
7 Skill 2 - Sub-Skill
8 Skill 2 - Player Property
9
10 Skill 3 - Dedicated to Evil
11 Skill 3 - Icon
12 Skill 3 - Sub-Skill
13 Skill 3 - Player Property
14
15 Skill 4 - Penguinification
16 Skill 4 - Icon
17 Skill 4 - Sub-Skill
18 Skill 4 - Player Property
19
20 Skill 5 - Nest of Evil
21 Skill 5 - Icon
22 Skill 5 - Sub-Skill
23 Skill 5 - Player Property
24
25 Skill 6 - Dire Emperor Penguin
26 Skill 6 - Icon
27 Skill 6 - Sub-Skill
28 Skill 6 - Player Property

Adding a Strategic Spell

1. In the Tome_Penguins.rpk, add a new **Category** and name it *Tome of Penguins - Strategic Spells*.
2. Add the resource **AoW Special Operation: Instant** and name it *Summon Dire Penguin - Strategic Spell*.
    a. **Screen Name**, **Screen Description**, **Screen Type** and **Large Icon Link:** Can use the text and icons as the skill.
    b. **FX Sequencer**: The PFX that play when this spell is cast. Select **Casting SC Chaos**.
    c. **Event > Targeters**: Links to targeters used for this spell. All spells will need at least 1 targeter.
    d. **Required Player Properties**: By default, all spells are available to everyone. So make sure to add a property here that players need to unlock. Link **Skill 1 - Player Property**.

    e. **Requisites**: Identifiers used for effects such as discounts. Add **Strategic Spell**, **Strategic: Summon Spell** and **Chaos Affinity Content**.
    f. **AI Priority Resource Link**: Set to **Strategic Operation AI Priority - Medium**.
    g. **Cost Data Link**: This defines the casting point and mana cost to cast this spell. Set this to **Strategic Summon Spell Tier 1 (60)**.
3. Add the resource **Magic - Strategic Position Targeter: Single** and name it *Summon Dire Penguin - Position Targeter: Single*. Targeters determine what hexes or units are affected by a spell.
    a. **Block Enemy Units**: Set to **True**.
    b. **Movement Checker Unit Link**: Set to **Unit - Land Movement Template**.
    c. In **Summon Dire Penguin - Strategic Spell**, you can now link this resource to the **Events > Targeters**.
4. Add the resource **Magic - Strategic Position Effect: Summon Unit** and name it *Summon Dire Penguin - Summon Unit*.
    a. In **Summon Dire Penguin - Position Targeter: Single**, link this resource to the **Effects**.
5. Add the Resource **AoW Unit Category** and name it *Summon Dire Penguin - Unit Category*.
    a. **CategoryName**: Add the following text: SUMMON_PENGUIN_CAT
    b. **Sets**: Add the following text: *PENGUIN*.
    c. In **Summon Dire Penguin - Summon Unit**, link this resource to the **Unit Category Link**.
6. Make sure the research skill has the **Research: Strategic Summon** and **Research Affinity: Chaos** requisites and the sub-skills Type setting is set to **Operation**.

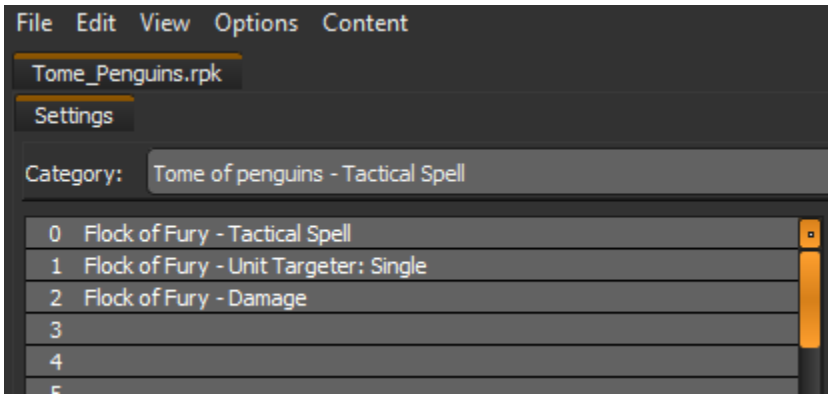


You can test the mod. Choose the Tome of Penguins in faction creation, prime the spell and cast it.

Adding a Tactical Spell

1. In the Tome_Penguins.rpk, add a new **Category** and name it *Tome of Penguins - Tactical Spells*.
2. Add the resource **AoW Special Operation: Combat** and name it *Flock of Fury - Tactical Spell*. A lot of the setup for this will be similar to adding a Strategic Spell.
    a. **Screen Name**, **Screen Description**, **Screen Type** and **Large Icon Link:** You can use the same text and icons as the skill.
    b. **FX Sequencer**: The PFX that plays when this spell is cast. Select **Casting TC Chaos**.
    c. **Event > Targeters**: Links to targeters used for this spell. All spells will need at least 1 targeter.
    d. **Required Player Properties**: By default, all spells are available to everyone. So make sure to add a property here that players need to unlock. Link **Skill 2 - Player Property**.

    e. **Requisites**: Identifiers used for effects such as discounts. Link **Tactical: Debuff Spell**, **Tactical: Direct Damage Spell** and **Chaos Affinity Content**.
    f. **AI Priority Resource Link**: Set to **Strategic Operation AI Priority - Medium**.
    g. **Cost Data Link**: This defines the casting point and mana cost to cast this spell. Set this to **Tactical Operation Tier 1 (10/10)**.
3. Add the resource **Magic - Tactical Unit Targeter: Single** and name it *Flock of Fury - Unit Targeter: Single*.
    a. **Effects:** Add **Blinded - Effect Property** and **Bleeding - Effect Property** .
    b. **Friendly Fire Type**: Set to **Enemies Only**.
    c. In **Flock of Fury - Tactical Spell**, you can now link this resource to the **Events > Targeters**.
4. Add the resource **Magic - Tactical Unit Effect: Damage** and name it *Flock of Fury - Damage*.
    a. **Damage > Physical** set the damage value from 0 to 15.
    b. In **Flock of Fury - Unit Targeter: Single** link this resource to the **Effects**.



You can test the mod. Choose the Tome of Penguins in faction creation, research Flock of Fury, go into manual combat and cast the spell on an enemy.

Adding a Unit Enchantment

1. In the Tome_Penguins.rpk, add a new **Category** and name it *Tome of Penguins - Unit Enchantments*.
2. Add the resource **Aow Special Operation: Sustained - No Map Marking** and name it *Dedicated to Evil - Unit Enchantment*. Sustained Operations are spells that have a continuous effect that can be canceled by the player.
    a. **Screen Name**, **Screen Description**, **Screen Type** and **Large Icon Link**: You can use the same text and icons as the skill.
    b. **Required Player Property**: By default, all spells are available to everyone. So make sure to add a property here that players need to unlock. Link **Skill 3 - Player Property**.
    c. **Requisites**: Identifiers used for effects such as discounts. Link **Strategic Spell**, **Strategic: Unit Enchantment**, **Chaos Affinity Content**.
    d. **AI Priority Resource Link**: Set to **Strategic Operation AI Priority - Medium**.
    e. **Cost Data Link**: This defines the casting point and mana cost to cast this spell. Set this to **Unit Enchantment Tier 1 (70)**.
    f. **Duration**: Determines how long the sustained spell is active for. Set this to **-1** to make it last until canceled by the player.
    g. **Is Unit Enchantment**: Set this to **True**.
3. Add the resource **Magic - Player Targeter: Self** and name it *Dedicated to Evil - Player Targeter: Self*. This allows a spell to target the player that cast it.
    a. In **Dedicated to Evil - Unit Enchantment**, you can now link this resource to the **Events > Targeters**.
4. Add the resource **Magic - Player Effect: Property** and name it *Dedicated to Evil - Player Effect: Property*. This allows temporary player properties to be added to a player.

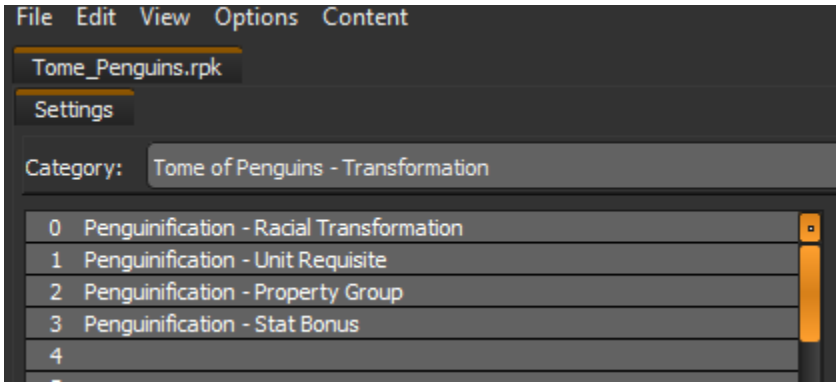
    a. In **Dedicated to Evil - Player Targeter: Self**, you can now link this resource to the **Effects**.
5. Add the resource **Magic - Player Property: Unit Upgrade** and name it *Dedicated to Evil - Player Property: Unit Upgrade*. This causes all the units under the player to be affected by the linked Unit Upgrade.
    a. In **Dedicated to Evil - Player Effect: Property**, you can now link this resource to the **Player Properties**.
6. Add the resource **AoW Unit Upgrade** and name it *Dedicated to Evil - Unit Upgrade*. This allows us to give properties to units with specific requisites. And it allows us to charge an upkeep for each affected unit.
    a. **Requisite Filter > Required Requisites**: Link to **Shock Unit - Requisite** and **Fighter Unit - Requisite**. If this is empty it will be applied to all owned units. You can add any requisite you want here.
    b. **Upkeep Link**: Link to **Unit Enchantment Upkeep - Cheap (2 per unit)**.
    c. In **Dedicated to Evil - Player Property: Unit Upgrade**, you can now link this resource to the **Player Properties**.
7. Add the resource **AoW Unit Property: Group** and name it *Dedicated to Evil - Property Group*. Property groups are used to store multiple properties. The advantage of using this as opposed to linking separate properties in the Unit Upgrade is that you can more easily decide what properties are displayed, what text is used and it is generally easier to change later.
    a. **Screen Name**: Use the same as the spell.
    b. **Screen Description:** You can optionally write a different text to differentiate between the spell and the property it gives. Or use the same one as the spell.
    c. **Large Icon Link**: Use the same as the spell.
    d. **List**: These are the actual properties the unit will get. Add **Vicious Killer - Property** and **Base Melee Strike - +2 Physical Damage Unit Property**.
    e. **Display Children**: Set this to **False**. To avoid duplicate mentions of the same property. We do not want Vicious Killer to appear separately in the unit panel.
    f. In **Dedicated to Evil - Unit Upgrade**, you can now link this resource to the **Properties**.

You can now test the mod. Choose the Tome of Penguins in faction creation, research Dedicated to Evil, make sure you have a shock or fighter unit (summon some Dire Penguins) and cast the Unit Enchantment.

Adding a Transformation

1. In the Tome_Penguins.rpk, add a new **Category** and name it *Tome of Penguins - Transformation*.
2. Add the resource **Aow Special Operation: Racial Transformation** and name it *Penguinification - Racial Transformation*. This is the operation that allows you to apply race transformations. It works very similar to strategic spells.
    a. **Screen Name**, **Screen Description**, **Screen Type** and **Large Icon Link**: You can use the same text and icons as the skill.
    b. **Required Player Property**: By default, all race transformations are available to everyone. Make sure to add a property here that players need to unlock. Link **Skill 4 - Player Property**.
    c. **Requisites**: Identifiers used for effects such as discounts. Link **Strategic Spell**, **Strategic: Transformation**, **Chaos Affinity Content**.
    d. **AI Priority Resource Link**: Set to **Strategic Operation AI Priority - Medium**.
    e. **Cost Data Link**: This defines the casting point and mana cost to cast this spell. Set this to **Race Transformation Tier 1 (150)**.
    f. **Transformation Tags**: If you want these Transformation to be used by node defenders, open the Tag Set Editor, right click and add **MinorTransformation** and **ChaosTransformation**.

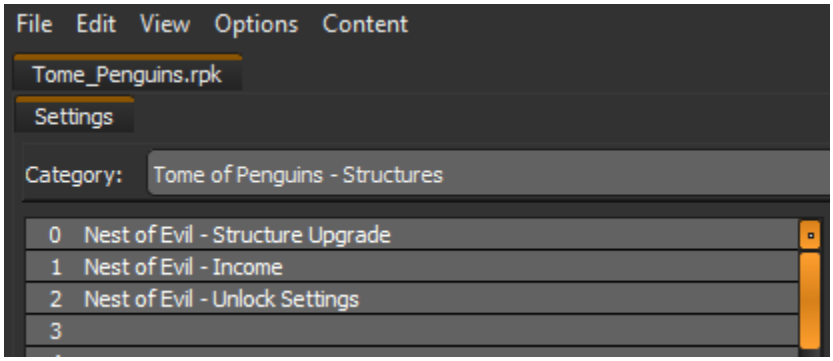
    g. **Combat Value Bonus**: Modifier on a unit's strength rating. This is used for AI and humans to evaluate the strength of a threat. Set this to 0.2.
3. Add the resource **AoW Unit Requisite** and name it *Penguinification - Unit Requisite*.
    a. **Screen Name**: Use the same name as the skill.
    b. In **Penguinification - Racial Transformation**, you can now link this resource to the **Requisites to Give**.
4. Add the resource **AoW Unit Property: Group** and name it *Penguinification - Property Group*.
    a. **Screen Name**, **Screen Description**, **Screen Type** and **Large Icon Link**: You can use the same text and icons as the skill.
    b. **List**: The list of properties the units will get. Add **Intimidating Aura - Auto Spell**.
    c. **Display Children**: Set to false. The Transformation itself will already be displayed.
    d. In **Penguinification - Unit Requisite**, you can now link this resource to the **Property Link**.
5. Add the resource **AoW Unit Property: Stat Bonus** and name it *Penguinification - Stat Bonus*.
    a. **Screen Name**, **Screen Description**, **Screen Type** and **Large Icon Link**: You can use the same text and icons as the skill.
    b. **Damage Reduction**: You can set defensive bonuses for all damage types. For now go to **Frost** and change the value to **3**.
    c. In **Penguinification - Property Group**, you can now link this resource to the **List**.
6. You can add visually changes using Modularity Providers. Check out Modularity for more information.



You can now test the mod. Choose the Tome of Penguins in faction creation, research Penguinification, and cast the transformation.

Adding a City Structure

1. In the Tome_Penguins.rpk, add a new **Category** and name it *Tome of Penguins - Structures*.
2. Add the resource **Structure Upgrade: Standard** and name it *Nest of Evil - Structure Upgrade*. This is the city structure itself.
    a. **Screen Name**, **Screen Description**, **Screen Type** and **Large Icon Link**: You can use the same text and icons as the skill.
    b. **Cost Data Link**: The defines the gold and production cost to build the structure in the city. Set this to **Cost Tier 1**.
    c. **AI Priority Resource Link**: Set to **Strategic Operation AI Priority - Medium**.
3. Add the resource **Colony Property: Income** and name it *Nest of Evil - Income*.
    a. **Right-Click** on **Income Flat** and select **Gold**.
    b. Do the same for **Mana**.
    c. Set the values for both to **10**.
    d. **Multiply by Alignment**: Set this to True. It will open some new options.
    e. **Allow Negative Multiplier**: Set this to false.
    f. **Alignment Type**: Set this to **Warmonger LVL 1**. This is a link to the alignment resources. It does not matter which Warmonger level you pick.
    g. In **Nest of Evil - Structure Upgrade**, you can now link this resource to the **Colony Properties**.
4. Add the resource **Property: Unlock Settings** and name it *Nest of Evil - Unlock Settings*.
    a. **Property**: Link to the **Skill 5 - Player Property**. This now means that when a player gains that player property, all other resources linked under **Unlocks** will also be given to that player.
    b. **Structure Upgrades**: Add the **Nest of Evil - Structure Upgrade**.

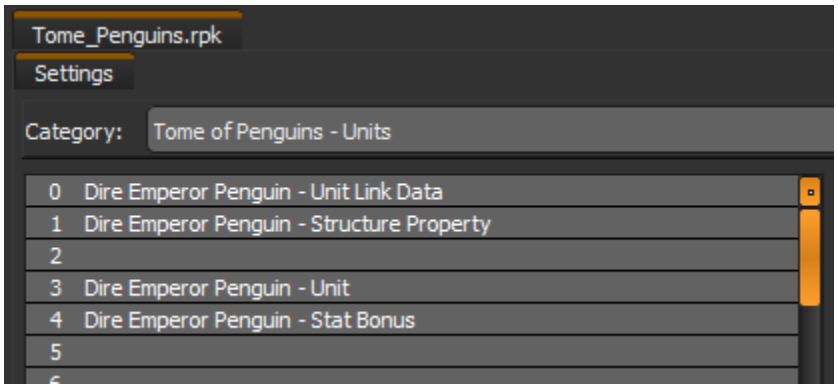


You can now test the mod. Choose the Tome of Penguins in faction creation, research the Nest of Evil, and construct it in your city. You can get evil alignment with the cheat DEMONEN.

Unlocking a Unit

You can have a look at the example mod to see how the unit is set up. Or otherwise follow the guides for creating new units.

1. In the Tome_Penguins.rpk, add a new **Category** and name it *Tome of Penguins - Structures*.
2. Check out the guide Creating a New Unit to see how you make a new unit. In the example mod, the added unit is a Dire Emperor Penguin. Which is a Tier 2 Shock unit with mostly the same setup as the normal Dire Penguin.
   (Don't forget you can always copy-paste resources between packs.)
3. Check out the guide Making a unit recruitable to see how you can make the Dire Emperor Penguin available for recruitment.

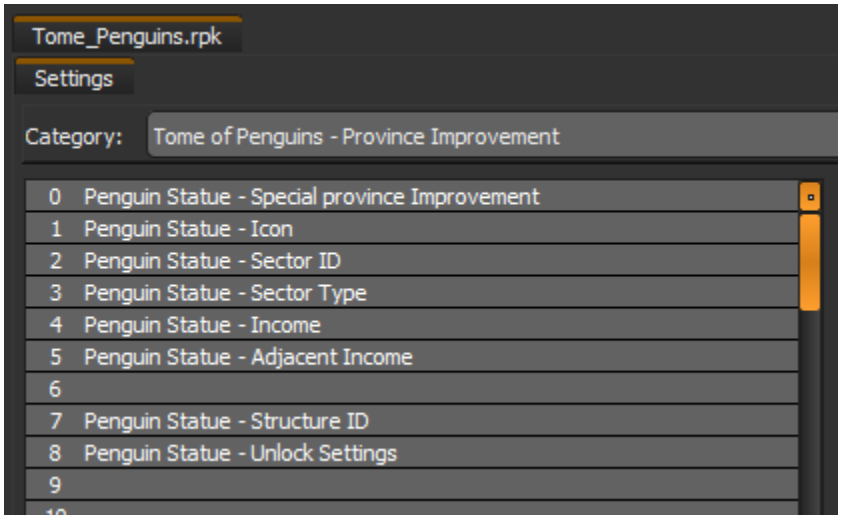


Adding a Special Province Improvement

Special Province Improvements function very similar to City Structures.

1. In the Tome_Penguins.rpk, add a new **Category** and name it *Tome of Penguins - Province Improvements*.
2. Add the resource **Structure Upgrade: Standard** and name it *Penguin Statue - Special Province Improvement*.
    a. **Screen Name**, **Screen Description**: Make some new text IDs and attach them here.
    b. **Requirement Description**: Link to this text ID CITYUPGRADES_STRATEGIC@PROVINCE_IMPROVEMENT@REQUIREMENT
    c. **Icon Link**: Add a new icon and link it to the **CityUpgrades_LeaderStatue**.
    d. **Modularity Provider**: This is used for the visualization of the province improvement on the world map. Link to **Structure Improvement: Leader Statue**.
    e. **Cost Data Link**: The defines the gold and production cost to build the structure in the city. Set this to **Cost Tier 2**.
    f. **Structure Properties**: Add the resource **Override: Improvement_Conduit**. This overrides the tactical combat map that will be used when fighting on this structure.
    g. **Sector Properties**: Add the resource **Conduit - Sector ID**. This will be used for adjacency bonuses.

    h. **Must have Structure Properties**: Link to **Town Hall Tier 2 - Base City Level - ID**. This ensures that the Special Province Improvement is unlocked by a Tier 2 Town Hall like all the others.
    i. **AI Priority Resource Link**: This determines the value the AI sees for this structure. Set to **Strategic Operation AI Priority - Medium**.
    j. **Sector Improvement**: This determines whether it is a structure built in the city or a Special Province Improvement. Set this to true.
    k. **Requisites**: This is used as an identifier for other effects such as scaling income per X. Add the requisites **Conduit - Requisite** and **Special Province Improvement - Sector Requisite ID**.
3. Add the resource **Sector Property** and call it *Penguin Statue - Sector ID*. This is just used as an identifier.
    a. In the **Structure Upgrade** link this in the **Sector Properties**.
4. Add the resource **Sector Type** and name it *Penguin Statue - Sector Type*. This is used to make it display correctly on the world map.
    a. **Name**, **Description**: Add the same IDs as the Structure Upgrade.
    b. **Requirements** > **Required All Properties**: Link to the **Penguin Statue - Sector ID**.
5. Add the resource **Colony Property: Income** and name it *Penguin Statue - Income*. This can give income to a city in a number of different ways.
    a. Right click on **Income Flat** > **New** > **Mana**. And set the mana value to 10.
    b. In the **Structure Upgrade** link this in the **Colony Properties**.
6. Add the resource **Sector Property: Income** and name it *Penguin Statue - Adjacent Income*.
    a. **Override Screen Name**: Set to **False**. This will automatically link the name of this structure wherever this income is displayed in breakdowns.
    b. Right click on **Income Flat** > **New** > **Unit Production**. And set the Unit Production value to 7. This is the Draft resource, but in resources it is called Unit Production.
    c. **Multiply by Adjacent Sectors** > **Required One Properties**: Add **Farm - Sector ID**. This will now multiply the income by the amount of adjacent provinces that count as farms.
    d. In the **Structure Upgrade** link this in the **Sector Properties**.
7. Add the resource **Structure Property** and name it *Penguin Statue - Structure ID*. This is used as an identifier we can give to players.
    a. In the **Tome of Penguins** (AoW Skill Grid: Group) link this resource to the **Initial Player Properties**.
8. Add the resource **Property: Unlock Settings** and name it *Penguin Statue - Unlock Settings*. This is used to unlock a variety of things and is also used for the City Structures and Cultures. Whoever has the property automatically also unlocks all the other things.
    a. **Property**: Link to **Penguin Structure - Structure ID**.
    b. **Structure Upgrades**: Link to **Penguin Statue - Special Province Improvement**.

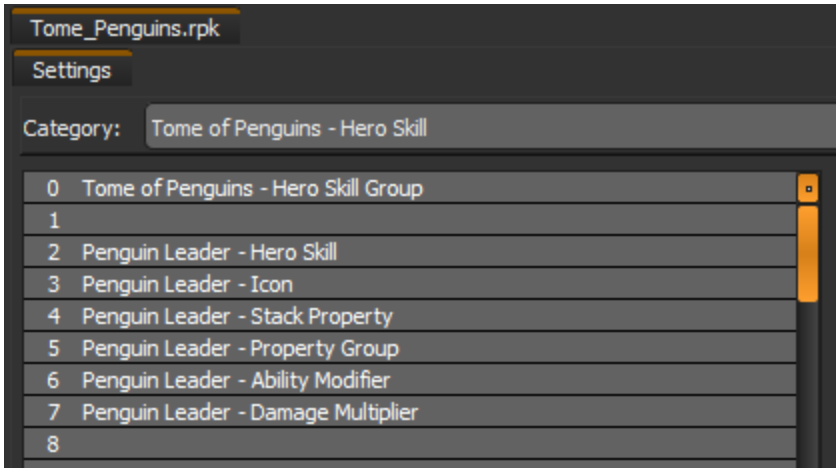


You can now test the mod. Choose the Tome of Penguins in faction creation. Get to 3 population using the cheat FREEPOP when having your city selected and annex a province. Then make a Tier 2 Town Hall using the REMBRANDT and PHILIPS cheats. Now you can build the Special Province Improvement.

Adding a Hero Skill

1. In the Tome_Penguins.rpk, add a new **Category** and name it *Tome of Penguins - Hero Skill*.
2. Add the resource **AoW Hero Skill Group** and call it *Tome of Penguins - Hero Skill Group*. This allows multiple Hero Skills to be connected to a single Hero Skill Group. This will be used to link the Tome and the Hero Skill.
    a. In the **Tome of Penguins** (AoW Skill Grid: Group) link this to the **Hero Skill Group Link**.
3. Add the resource **AoW Hero Skill**: Normal and name it *Penguin Leader*.
    a. **Screen Name**, **Screen Description**: Make new text IDs and attach them here.
    b. **Icon**: Add an icon resource under the Hero Skill, choose an icon and link it here.
    c. **AI Priority Resource Link**: Set this to **Hero Skill AI Priority - Medium**.
    d. **CategoryLink**: This determines under what category it falls for the purpose of progressing through the Hero Skill levels. Set this to **Support - Hero Skill Category**.
    e. **Level Link**: Determines how many Hero Skills of the same category you need to have to unlock this. Set it to **Novice - Hero Skill Level**.

    f. **Group Link**: This is the last thing needed to link the Tome and the Hero Skill. Whenever you get that Hero Skill Group, you will get all Hero Skills that have it linked. Set this to **Tome of penguins - Hero Skill Group**.
4. Add the resource **AoW Unit property: Stack property** and call it *Penguin Leader - Stack Property*. This allows you to have a unit give a property to other units in the same stack.
    a. **Screen Name**, **Screen Description**, **Icon**: Link the same as in the Hero Skill.
    b. **Requisite Filter** > **Forbidden Requisites**: Add the **Hero Requisite**.
    c. **Requisite Filter** > **Required Requisites**: Add the **Animal Requisite**.
    d. **Affects Self**: Set this to **False**.
    e. **Stack leader Only**: Set this to **True**.
    f. In the **Penguin Leader - Hero Skill** link this to the **Properties**.
5. Add the resource **AoW Unit Property: Group** and name it *Penguin Leader - Property Group*.
    a. **Screen Name**, **Screen Description**, **Icon**: Link the same as in the Hero Skill.
    b. **Display in Unit Panel**: Set to **False**.
    c. In the **Penguin Leader - Stack Property** link this to **Unit Property**.
6. Add the resource **AoW Unit Property: Ability Modifier** and name it *Penguin Leader - Ability Modifier*.
    a. **Screen Name**, **Screen Description**, **Icon**: Link the same as in the Hero Skill.
    b. **Display in Unit Panel**: Set to **False**.
    c. **Required Requisites**: Add **Melee Weapon**, **Ranged Weapon** and **Magic Weapon**. This determines which abilities get the modifier. We just added all attack abilities.
    d. In the **Penguin Leader - Property Group** link this to **List**.
7. Add the resource **AoW Ability Modifier: Damage Multiplier** and name it *Penguin Leader - Damage Multiplier*.
    a. **Screen Name**, **Screen Description**: Link the same as in the Hero Skill.
    b. **Display in Damage Previews**: Set to **False**.
    c. **Display in Ability Tooltips**: Set to **False**.
    d. **Multiplier**: Set to 1.2, which is a +20% increase.
    e. In the **Penguin Leader - Ability Modifie** link this to **Modifiers**.



You can now test the mod. Choose the Tome of Penguins in faction creation. Select your leader and cheat HAUER. You can not choose the Penguin Leader Hero Skill in the Support category. Add some Penguins (or other inferior animals) and see in their unit panels that their damage increases.

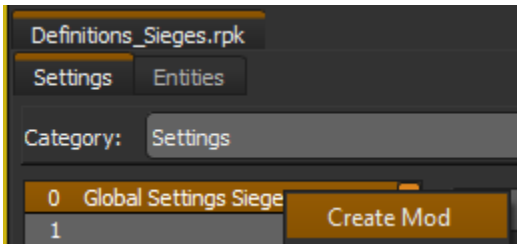
And that concludes this section on creating your own Tome. Time to experiment!

Sieges

The siege system itself is set up via resources, meaning it is moddable. However, due to the Siege Settings existing in a singular resource. Multiple siege system mods may conflict with eachother.

To mod the Siege System, follow these steps:

1. In your mod, open *Definitions_Sieges.rpk* of the base game's *.rpk* files.
2. Go to the **Settings** category, and mark the **Global Settings Siege** as a mod.



3. Within this resource, there are various options available to you to mod:
    a. **Combat Threshold** and **Progress Rate** are deprecated and should remain on 0
    b. **Required Units Amount** dictates the amount of units within an army is required to initiate and continue a siege.
    c. **Attacker Unit Properties** contians a list of **Unit Properties** given to besieging units during a siege (including, during the siege battle)

d. **Defender Unit Properties** the same as **Attacker Unit Properties**, but for defending armies inside the city.
e. **Colony Properties** contains a list of **City Properties** that a city gains while a city is under siege.
f. **Defender Unit Damage** is the amount of proportional damage defending units inside the city will take each turn a city is under siege. Note that regeneration is not turned off by default, you could turn this off by giving a **Unit Property** in **Defender Unit Properties** that blocks regeneration.
g. **Base Siege Fortification Damage** is the default amount of damage dealt to **Fortification Health** of a City under Siege every turn.
h. **Hero** and **Unit Tier 1** to **5 Siege Fortification Damage** is the amount of damage is dealt to **Fortification Health** for each hostile unit of that tier (or hero) that is adjacent to the besieged city. This counts for *every* unit that fits the tier or hero requirement.
i. **Base Slege Projects Slots** is the amount of Siege Project slots every player starts with. Note that we grant an extra starting slot by default via a **Player Property**.
j. **Max Siege Project Slots** is the maximum amount of slots a player can ever have, even if **Base Siege Projects Slots** or **Player Properties** would grant more. It will be clamped to this value.
k. **Defensive Upgrade Identifier** is a link to a Structure Property used to identify which Structure Upgrades should show up in the Siege Interface
l. **Under Siege PFX** links to a PFX that will play as long as a siege is ongoing, it will play on the center of the city.
m. **Siege Modularity Provider** links to a Modularity Provider added to the City while a siege is ongoing.
n. **Siege Unit Property** links to the Unit Property that identifies a required unit to start a siege. At least 1 Unit must have this property in an army to be able to start a siege (next to having enough units as defined in **Required Units Amount)**
    i. It is better to instead give the units you wish to be able to siege this property instead of linking something else here.
    ii. If you want all units to be able to siege, you can simply unlink it by linking *{none}*

Take a look around *Definitions_Sieges.rpk* for more on modding specific rules or content of the siege system!

Siege Projects

Siege Projects are granted to players as long as the pack they're in is loaded in and conditions (Required/Forbidden Player Properties) are met. Siege Projects themselves are easy to setup. But it's Siege Projects that bring siege engine units that might be a little more difficulty, more on that can be found near the end of this chapter.

To set up a Siege Project follow these steps:

Create a new *.rpk* file and create a new category within it.

Create a new **AoW Siege Project** resource

a. **DataStoreTag** can be filled in so you can reference this Siege Project as a hyperlink in text.

b. **IsDefault** should be set to false, if set to true, will automatically assign this project to any empty Siege Project slot

c. **Screen Name** link the name of the Siege Project

d. **Screen Description** link the description of the Siege Project, note that costs and Fortification Damage are automatically shown in the Siege Interface and do not need to be shown in the Description.

e. **Large Icon Link** links to an **Icon (Interface)**

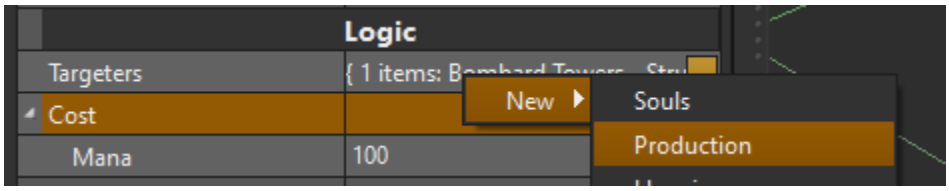
f. **Targeters** here is where you link **Targeters** that will trigger when the Fortification Health on the City under siege reaches 0. Note that all Targeters will be centered on the City core

i. This means it's often wise to affect entire **domains** or **provinces** if you want to do fancy World Map effects.

ii. You can use City/Structure targeters to target the city itself, this is often how we add **Combat Properties** for during the siege battle!

g. **Cost** here you can define resource costs that need to be paid upfront to undertake the Siege Project. You can add new resources to the cost by right clicking on it and selecting **New** -> select any resource you want.

i. You can then fill in an amount in the newly added resource cost.



h. **Siege Health Damage** is the amount of Fortification Damage this Siege Project will contribute each turn.

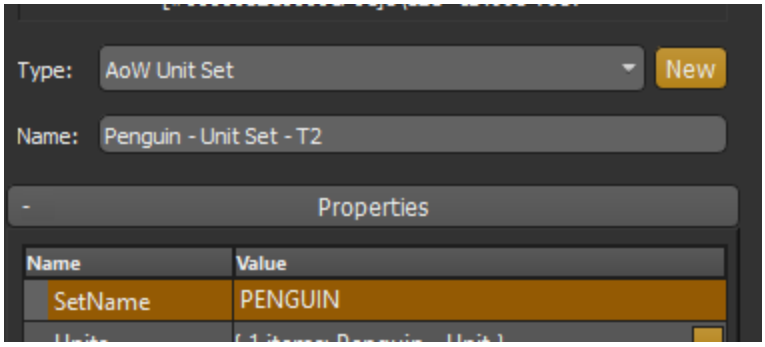
i. **Can Stack** when set to true, will allow you to assign this Siege Projects multiple times in a single siege!

j. **Required/Forbidden Player Properties** contain lists of Player Properties that are used as filters to know which players should have access to this Siege Project.

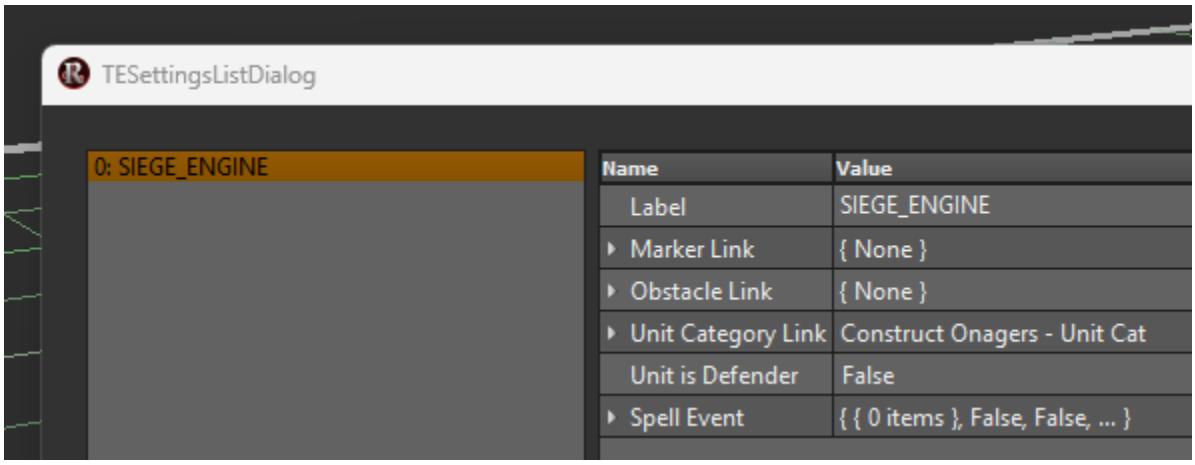
And there you go! Your Siege Project should now be available to players as long as the *.rpk file* containing the resource is loaded in and the conditions are met!

If you want to add in a **Siege Engine Unit** to your **Siege Project**, follow these steps:

1. Create a **AoW Unit Category** resource and in its **Sets**, type in any Unit Set's Set Name. Unit Set strings can be found in **AoW Unit Set -> SetName**



2. Create a **Structure Property: Tactical Defense** resource, open up its **Defense Data** list
3. Click **Add**, a new entry in the list called (null) should appear, select it.
4. Fill in the label as *SIEGE_ENGINE*
    a. This is the text label used for the markers that spawn in Siege engine units during Siege combat. All Siege Maps have dedicated markers to spawn them in specific locations.
5. Set the entry's **Unit Category Link** to the **AoW Unit Category** you created. And set **Unit is Defender** to false. Press **Ok** to save the changes and close the Defense Data list.



6. Create a new **Magic - Structure Effect: Add Property** resource. In this resource, link the new **Structure Property: Tactical Defense** in the **Structure Properties** list.
7. Create a new **Magic - Structure Targeter: Single** resource. In this resource, link your **Magic - Structure Effect: Add Property** in the **Effects** list.
    a. Make sure to check if **Friendly Fire Type** isn't set to *Own Stuff Only*,*Friendlies Only* or *Nothing*.
8. Link your **Magic - Structure Targeter: Single** in your **AoW Siege Project**'s **Targeters** list.

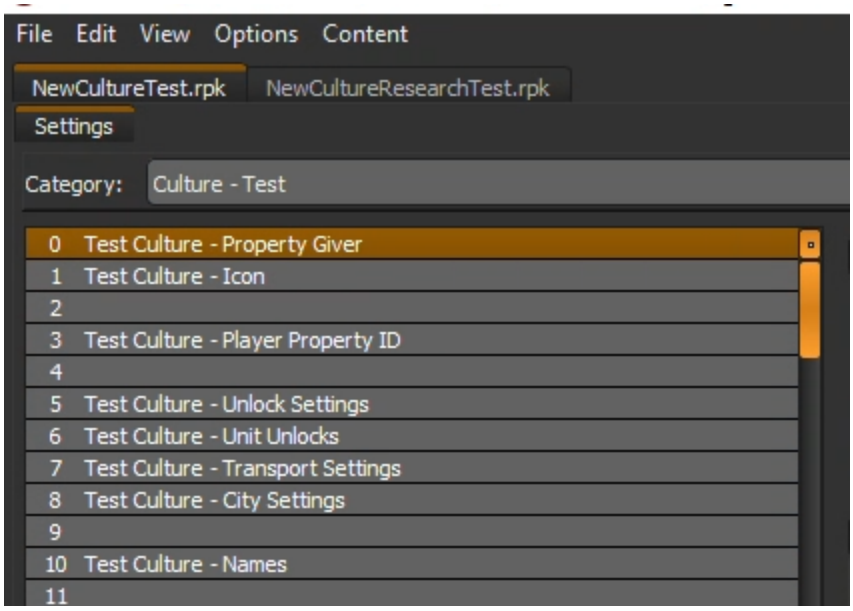
Your Siege Engine Siege Project is now ready!

Adding a Cultures

Culture Setup

1. Create a new package.
2. In **Content > Resource Packs** add the following packs:
    a. UNIT_MODULARITY.RPK
    b. CULTURE_DEFINITION.RPK
    c. SHAREDMODIFIERS_UNITPROPERTIES.RPK
    d. UPGRADES_BASEABILITY_UNITPORPERTIES.RPK
    e. EQUIPMENT.RPK
    f. TRAITS_BACKGROUND_CHARACTERS.RPK
3. Create a new **Category**.
4. Add the resource **AoW Faction Property Giver**.
    a. **Type**: Set to **Culture**.
    b. Add a **Screen Name**. And optionally a **Description**, **Lore** and **Bio Description** and **Icon**.
    c. **Provider**: This is used for the default visuals of the faction. Set this to any of the existing cultures.
    d. **Hero Affinities**: Despite the name, this is the affinity given to both the player empire *and* the race (and therefore also Heroes). Right click to add an affinity type. You can add multiple affinities here. Then change the value(s) from 0 to 1 or 2.
5. Add the resource **Player Property**.
    a. In **AoW Faction Property Giver**, you can now link this resource to the **Player Properties**.
6. Add the resource **Property: Unlock Settings**.
    a. **Property**: Link to the **Player Property**.
    b. **Structure Upgrades**: Link to any City Structures or Special Province Improvements that the culture needs. Note that this does not include City Structure replacements.
    c. **Leader Backgrounds**: This is the list of equipment you normally choose from in faction creation. You can link to a Shock, One handed + Shield, Ranged, Orb and Staff Equipment Background here. Note that these must have the AM identifier. (Standing for Ascended Mortal/Champion)
    d. **Hero Backgrounds**: These are for Heroes with this culture you recruit later during gameplay. You can just link to the same backgrounds as for Leaders.
7. Add the resource **AoW Culture Unit Link Data**.
    a. **Culture Link**: Link to the **AoW Faction Property Giver**.
    b. **Units**: Link the units that belong to this culture. To learn how to create units, check out Creating a new Unit.
8. Add the resource **AoW Culture Transport Settings**. This defines the boats the culture will use.

    a. **Culture Link**: Link to the **AoW Faction Property Giver**.
    b. **Movement Requisites**: Link to **Water Movement Requisite**.
    c. **Extra Properties**: Link to the following properties:
       Naval Units - Group
       Embarked - Penalty
       Water Unit Vision Boost - Unit Property
       Default Transport Movement Settings Override - Property
    d. **Provider**: Link to **Unit Type: Transporter**.
9. Add the resource **City Culture Settings**.
    a. **Culture Link**: Link to the **AoW Faction Property Giver**.
    b. **Culture Property Link**: Link to the **Player Property**.
    c. **Defender Settings**: This determines the unit compositions that are used when armies are spawned on the world map. This is used for your starting army and for the armies that Free Cities use to defend their nodes.
    d. **Defenders: UnitPool Category Name**: Add an unique identifier for this. This needs to be filled in.
    e. For the other defender options, you can see how another culture is set up. By default, the Spawn Strength is set to Weak and the defenders increase in Tier and number the stronger it is.
    f. **Reinforcement Settings**: This is a fallback system when a Ruler has no units left. You can copy the settings from the previous step.
    g. **Garrison Spawn Strength:** Set to **Disabled**.
    h. **Forward Base Garrison Spawn Strength:** Set to **Disabled**.
10. Add the resource **AoW: Name**.
    a. **Type**: Set to **Culture**.
    b. **Culture**: Link to the **AoW Faction Property Giver**.
    c. **Unlocks**: All of these options need to have at least 1 text ID linked to it. You can make your own names or link to IDs used by other cultures.



You can now test your modded culture by launching the game, creating a new game and race and choosing your culture.

Culture Research

The research provided by cultures functions the same way as the research given by Tomes.

1. Create a new .rpk and save it in the packs folder.
2. Follow the guide on Tome Setup, with the following exceptions:
    a. Set the **Tier** of the tome to **Tome - Default Tier**.
    b. Do not add the **AoW Skill Grid Group: List Data Settings**, since we will directly link the research in the culture. It should not show up as a possible Tome choice during gameplay.
3. Add any research skills you want. How to make those can be found in the chapters of Adding a New Tome.
4. Go back to the modded culture.rpk.
    a. Go to **Content > Resource Packs > Add** and link the new modded culture research.rpk you made.
    b. In the **AoW Faction Property Giver**, you can now link the **Skill Grid: Group** to the **Skill Groups**.

Culture City Structures/Special Province Improvements

To add a new City structure or Special Province Improvement:

1. In the **Property: Unlock Settings,** go to the **Structure Upgrades.** You can link to any City Structure or Special Province Improvement you want the culture to have.

2. To learn how to make new City Structures check out Adding a City Structure in the Tome section of this guide.
3. To learn how to make new Special Province Improvements check out Adding a Special Province Improvement in the Tome section of this guide.
4. Unlike the default game, it is easier to just put the City Structures and Special Province Improvements directly in the modded culture.rpk. And for organization purposes you can give them their own category.

You can also override the default city structures with cultural unique variants. This is done via the **City Culture Settings**.

1. Right Click the **Culture Structure Upgrades** and select the City Structure you want to replace.
2. Expand the newly created line. Link the City Structure you want to replace it with.
3. Cultures usually also replace the Town Hall II, Town Hall III and Town Hall IV with cultural unique versions. But this is not required.

Culture Hero Skills

To learn how to make new Hero Skills check out Adding a Hero Skill.

1. In **AoW Faction Property Giver**, you can now link this resource to the **Hero Skill Group Link** instead of linking it to the Tome.

Units and Heroes

Creating a new Unit

It is recommended to look at existing units and find one that is similar to what you want to achieve. You can copy the unit, paste it in your mod and use it as a base to work from.

1. Add the resource **AoW Unit**.
2. **Figure Settings**: This determines things like the size of the units and how many members the unit has and in what kind of formation they are standing.
3. **AI Priority Resource Link**: This determines how the AI decides to recruit units. By default always put this on **Unit AI Priority - Medium**.
4. **Tier Resource Link**: This determines the Tier of the unit. This influences the base HP, defense, resistance, status protection, upkeep and XP requirements. It will also be used as an identifier for many other effects.
5. **Type**: This is the unit role, such as Shock or Polearm units. Many have default properties that are given automatically according to their unit role.
6. **Subtype**: Optional identifiers that are used for animals, elementals or fiends for example. They also contain properties that are inherently given. They are also used for identifiers for conditional effects.
7. **Is Player Race Unit**: If this is turned on, the unit will be a racial unit and benefit from race transformations. It will also cause their visualization to try and use the modularity system to create a racial variation of the unit. Check out Modularity for more information.
8. **Summon Type**: This determines when a unit counts as 'being summoned'. They will be given the Magic Origin property and require Mana for their Upkeep. By default leave this on **When Summoned**.
9. **Requisites**: Requisites are used as identifiers and they can give unit properties. They are one level above properties, as such a property cannot give a requisite. Generally, a requisite is permanent, and won't be taken away during gameplay, but it can be given.
10. **Preferred Ability Link**: Link the active ability that is used for the main attack of the unit.
11. **Properties**: You will link all passive and active abilities here.
    a. All units have one of the properties that are formatted as follows: **Armor (Average) Resistance (Average) HP (Average) - Stat Bonus**.
    b. While HP scales with the unit tier, the damage the unit deals does not. Add Base Strike damage modifiers as appropriate.
12. **Defense Mode Link**: For most units, link the default **Defense Mode** here. Normally only shield units use the **Defense Mode - Shield Wall** and only support units use the **Defense Mode - Warding**.
13. **Upgrades**: Upgrades can be used to give properties using some extra logic like replacing other upgrades or requiring upkeep. You always want to add one of the movement upgrades here: **Land Movement - Unit Upgrade**, **Floating Movement - Unit Upgrade**, **Flying Movement - Unit Upgrade** or the **Swimming Movement - Unit Upgrade**.

14. **Screen Name**: Link to the ID of the name of the unit.
15. **Provider**: Add a Modularity Component to ensure the unit has visuals.
16. **Required Structure Property Link**: If you want a player to be able to recruit this unit in cities, add a **Structure Property** here. Then you want to give the structure property to the player via Research Skills, Cultures or other sources.
17. **Cost Data Link**: This determines the Gold and Draft cost to recruit the unit.

Make a unit recruitable

1. Add the resource **AoW Culture Unit Link Data**. Despite the name, this is not exclusively used for cultures. Any unit linked in here becomes immediately available for recruitment for everyone!
2. Add the resource **Structure Property**. This is just an identifier we will give to colonies.
    a. In the **AoW Unit** > **Required Structure Property Link** you can now link this property. This means that the unit will no longer show up for everyone, but only for those with this Structure Property.
3. To unlock via Research: In the **AoW Skill Grid: SubSkill**, Instead of using as Player Property, you can link to the **Structure Property**.

Alternatively, you can give this structure property via City Structures, Special Province Improvements, race traits or anywhere else where you could give a structure property.

Creating a Passive Property

Unit Properties
There are various unit properties that add different passive effects to a unit. They need to be added in **AoW Unit > Properties**.

1. **Screen Name**: Always add an ID here, even if you won't display this property. Otherwise the resource will not work.
2. **Screen Description**: Add a description via a text ID here.
3. **Large Icon Link**: Link the icon this property will use.
4. **Display in Unit Panel**: If this is set to false, this property will not show up in the unit panel.
5. Display Counts: Only relevant if the property can be given multiple times, which by default is only used for specific Status Effects.
6. **Display Type**: When set to primary it will be displayed in the Unit Panel. Secondary will display it smaller and at the top, on the same level as the unit types. Important will display it both in the Unit Panel but also in the bottom panel when selecting the unit in combat. By default this is used for damage responses.

Example Unit Properties

- **AoW Unit Property: Stat Bonus**: Give extra HP, Defense, Resistances and/or Vision.
- **AoW Unit Property: Status Resistance**: Specifically give Status Resistance.

- **AoW Unit Property: Magic Effect Shield**: Used for damage response. To either damage the attacker or inflict them with status effects.
- **Aow Unit property: Auto Spell**: This allows you to add new targeters to the unit when a certain trigger takes place, such as **End of Turn**, **Any Spell Cast** and **On Friendly Unit Dies**. The targeters can target the unit itself or all adjacent enemies for example.

Ability Modifiers
Some passives affect a unit's abilities. Such as increasing damage in certain scenarios or applying extra effects to attacks.

1. Add the **Resource AoW unit Property: Ability Modifier**.
2. Add a **Screen Name** but otherwise you generally won't need to add a description or icon since you want to display the Ability Modifier inside of the ability.
3. **Modifiers**: Add the Ability Modifiers that will be applied.
4. **Requisite Filter**: This filter allows you to tell the system on what abilities the modifiers can be applied on. All abilities have requisites such as **Melee Strike (Base)**, **Ranged (Base)** and **Magic Weapon (Base)**. The non-basic attacks (often abilities with cooldowns) use the same naming convention but do not include the suffix **(Base)**.

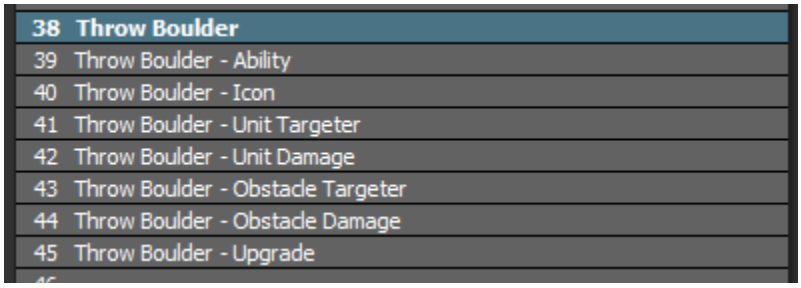
Example Ability Modifiers

- **AoW Ability Modifier: Damage Multiplier**: A damage multiplier on the base damage that has the option to only be conditionally applied for attacking enemies with specific requisites or properties, Attacks of Opportunities, Flanking, Hexes Moved, Crits, First Strike and whether it is your turn.
- **Aow Ability Modifier: Extra Effect**: You can add status effects the attack will inflict here. It has the same filters.
- **AoW Ability Modifier: Extra Targeter**: You can add extra targeters such as a Circle, Line or Caster with entirely new effects.

Creating an Active Ability

1. Add the resource **Aow Unit Property: Ability**.
2. **Combat behavior settings**: This is used for the animation and PFX for the ability. You can often link a behavior that starts with Melee or Magic that fits.
3. Add the **Screen Name**, **Screen Description**, **Large Icon Link** as appropriate.
4. **Ability Type**: Determines how many action points the ability uses.

5. **Accuracy Settings Link**: Sets the range and accuracy of the attack with the amount of accuracy falloff per hex. For melee attacks just use **Melee - Can't Miss**. Most buffs and debuff abilities use the **Can't Miss** accuracy settings with a specific range.
6. **Requisites**: The identifiers for this ability. You can identify this as a basic ability, an active ability or whether it is a support or summon for example. These determine what other effects from Unit Enchantments, buffs or debuffs can be applied on this ability.
7. **Cooldown**: Non-basic active abilities often have a cooldown of at least 1.
8. **Blocked When Engaged**: You want to set this to true if you want a spellcasting or ranged unit to be unable to use this ability in melee.
9. **Event Type**: Make this match the type of ability this is. Cast spell is used for magic attacks or buffs.
10. **Event > Targeters**: Add at least 1 targeter here. Often this will be the **Magic - Tactical Unit Targeter: Single**. In the targeter you can add **Effects** such as **Magic - Tactical Unit Effect: Damage** or you can link to status effects you want to apply..
11. **Primary Effect Link**: This is used for displaying effects in the UI. Link to your primary effect such as tactical damage or the status effect.
12. **Allow Move and Use**: This automatically moves the unit in range to use the ability. For melee this is often true, but for ranged abilities this is normally set to false.

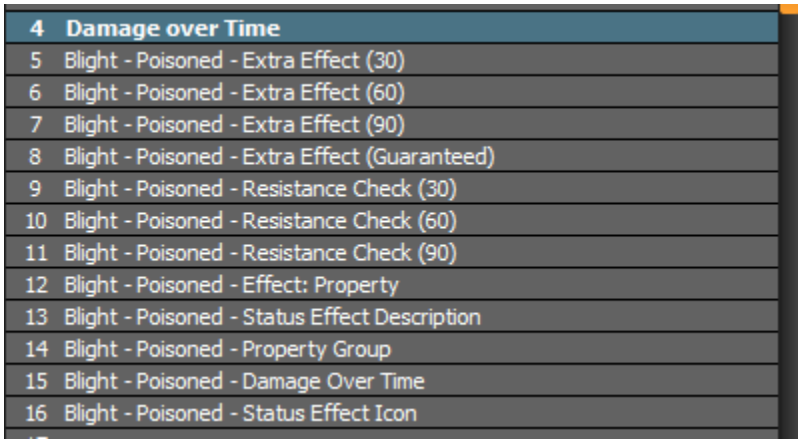


Creating a Status Effect

A status effect is technically a property you give to units via effects. These can be applied via Unit Abilities or via Spells. For spell only status effects you can skip step 1.

1. Add the resource **AoW Ability Modifier: Extra Effect**: This is given to a unit to apply the status effect. This can then be given to a unit ability directly or via a Unit Enchantment or Race Transformation.
    a. Add an **Effect Name** and **Effect Description**.
    b. **Conditional Logic**: If you have multiple effects you don't want to be used and displayed multiple times (for example multiple versions of Poisoned with a different chance to apply) you can make a group here and add a priority.
    c. **Targeting**: You can add extra logic for applying the Status Effect here, but that will normally be done in **Magic - Status Effect Description**.
    d. **Effects**: This will link to the effect that will add a property. Or you can put a resistance check in between.

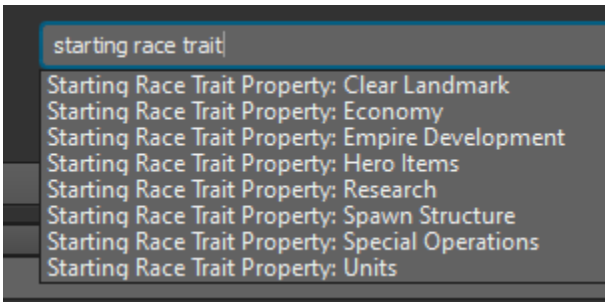
2. Optionally, add the resource **Magic - Tactical Unit Effect: Resistance Check**: This will allow the recipient the chance to resist the effect using their Status Resistance.
    a. **Strength**: Every point represents a 10% base chance to inflict the status effect.
    b. **Success Effect Link**: This will link to the effect that will add a property.
    c. In **AoW Ability Modifier: Extra Effect** link this to the **Effects**.
3. Add the resource **Magic - Tactical Unit Effect: Property**: This bridges the effect and the property.
    a. In **AoW Ability Modifier: Extra Effect** or link this to the **Effects**.
    b. Or in **Magic - Tactical Unit Effect: Resistance Check** link this to the **Success Effect Link**.
4. Add the resource **Magic - Status Effect Description**. This is the actual status effect.
    a. **Property Link**: This will link to the property that will contain what the status effect does to the unit.
    b. **Tooltip Text**: Add this text ID: PROPERTY@GAIN_PROPERTY
    c. **Num Turns**: This value is how many turns this status effect is active for.
    d. **Type Link List**: This is no longer currently used in Age of Wonders 4.
    e. **Target Property Filter > Forbidden Properties**: This is where immunities are placed to determine what units will not be affected by this status effect.
    f. **Blocked String Effect**: You can link a text ID here explaining why the status effect does not work.
    g. **Opposing Property Links**: This is used for the dispelling system. Whenever one of the status effects linked here would be applied while this status effect is present, the two instead cancel each other out and both are removed.
    h. **AI Is Positive**: Set this to false if the status effect is negative. Otherwise the AI won't use abilities or spells using this status effect properly.
    i. **AI Allow Dispel**: Set this to false if the status effect should not be allowed to be dispelled.
5. Now link a property or make a property for this status effect to link to the **Property Link**. To learn how to make properties, check out Creating a Passive Property.



Adding Race Traits

Body Traits

1. In the pack you will put your traits in, go to **Content > Resource Packs** and add RACETRAITS_WORLD>RPK and FORMTRAITS_WORLD.RPK.
2. Add the resource **Aow Trait**.
    a. **Type**: Set this to **Body**.
    b. **Screen Name**: Link to a text ID.
    c. **Screen Description**: You can describe the effect for the form traits here.
    d. **Lore**: This is used to give a quick pitch on why you would want this trait.
    e. **Screen Bio Description**: This is used to create the race bio.
    f. **Effect Descriptions**: Instead of the Screen Description, you can list the bonuses separately here. This could be useful if you need to use the same effect between multiple traits. (And for us developers it was useful for translations.)
    g. **Incompatible Race Traits**: This is meant for Society Traits to be mutually exclusive.
    h. **Incompatible Forms**: This is meant to make Body and Mind Traits limited to certain Forms.
    i. **Affinities**: Right Click to add affinity or an alignment shift to this trait.
    j. **Player Properties**: All linked player properties will be given to the player that has chosen this trait in faction creation. It does nothing for players who later get a race with this trait.
    k. **Unit Properties**: All linked properties will be given to the units that belongs to this race.
    l. **Colony Properties**: All linked properties will be given to all cities of that race.
    m. **Starting Race Trait Properties**: These make use of a category of resources that all start with **Starting Race Trait Property**. They are used to give starting bonuses for the player that chooses these traits during faction creation.



    n. **Unit Upgrades**: All linked upgrades will be given to the units that belongs to this race.
    o. **Mount Link**: You can make an **AoW Faction Mount** resource and link it here to make your race benefit from an exotic mount.

Mind traits

Mind Traits are mostly the same as the body traits but with the following differences:

1. Set the **Type** to **Mind**.
2. **Starting RMG Theme**: This forces the player to have some of the specified theme around them. This is used for adaptations.
3. **Starting RMG Overlay**: This forces the player to have some of the specified overlay around them. This is used for adaptations.

Society Traits

Society Traits are mostly the same as the body traits but with the following differences:

1. **Incompatible Cultures**: This is meant to make Culture Traits limited to certain Cultures.

Adding Realm Traits

Realm Traits are the modifiers that can be added to realms in Age of Wonders 4. Changing the world map generation and altering the player experience by adjusting some of the rules in the world map. A lot of the existing realm traits can be used to check what kind of modifiers are possible!

Tip: to get inspired or look at example traits, look at Empire_Core.rpk where most of the base game's realm traits are set up!

Realm Traits are **Player Empire: World Trait** resources, which can hold multiple links to settings that are meant to define or override specific parts of RMG.

- **Name, Description** and **Icon** to display and describe the realm trait in Realm Trait selection.
- **Category** which dictates what slot this Realm Trait will occupy.
- **Sub-Category**, which will automatically create or join the realm trait in a sub-header inside of realm selection, such as "Free City Modifiers"
- **Is Default** which when set to *True* will make this realm trait automatically picked when creating a new Realm. You usually want to keep this set to *False*.
- **Mutually Exclusive** which is a list of linked **Player Empire: World Traits** that cannot be picked in the same realm as this Realm Trait. It helps eliminate thematic mismatching or helps prevent mixing of Realm Traits that are incompatible.
- **Resource Pack File** where you can type out the exact filename of a *.rpk* file that will be loaded in only when the realm trait is active. In such *.rpk* files you can often put custom content like **Trigger Scripts**.
    - It is important these *.rpk* files do not get loaded in by default, as some content will automatically be picked up by the game and applied even when the realm trait was not chosen.
- **Type** which has several options to inform the resource what kind of content can be linked directly to the realm trait (in addition to, or instead of, the **Resource Pack File**)
    - **User Settings** choosing this will allow you to link several **User Setting** resources, which are used for RMG.
    - **Houses** setting is deprecated, no need to use this.
    - **Inhabitants** choosing this will allow you to link several **Player Empire: Inhabitants** resources, which can dictate premade AI empires that should appear in the realm.

To set up a new realm trait, follow these steps:

1. While having your mod loaded in the Package Manager, open the Resource Editor and create a new .**rpk** file.
2. **Save** the new *noname.rpk* in the Mod's */packs* folder*,* name it something like *"Realm Trait [realm trait name] Definition"*
3. In this new *.rpk* file*,* create a **category**, it can be called *"Realm Trait Definition"*

4. In here, create a **Player Empire: World Trait** resource. Name it something like *"Realm Trait: [name]"*
5. Link a string to the **name**, **description**, and link an **icon** (link to an **Icon: Interface** resource). See Adding Text on how to create custom text for this realm trait. Realm Traits that miss these may not show up in the game.
6. Make sure to also set its **Interface Stuff -> Category** correctly to make sure it shows up in the right realm trait slots.
7. Make sure to save the *.rpk* file by hitting CTRL + S.
8. Don't forget to load the new *.rpk* file by adding it to the **Content** list in your mod's *.acp* file!

You're done! The new realm trait should now show up in the list of available realm traits in the right slot! And its content will only be activated when said realm trait is active in a session!

Geography Traits

Geography traits are **Realm Traits** that alter the shape and distribution of the land and oceans of the world that are generated by **RMG** when starting a new game. The shape of the world will impact the flow of expansion and movement of armies. Only one **Geography trait** slot exists, so you cannot combine multiple in a single realm.

To set up a new Geography trait, follow these steps:

1. Set up a new **Realm Trait** using the steps in Adding Realm Traits.
2. Create a **Player Empire: User Setting** resource in your *RealmTrait[name]Definition.rpk* file, set its **Type** to **Map Type**
3. Now create a **RMG Map Type Definition** resource, and populate it with settings you would like it to have.
    a. To gain access to some of the premade settings to link in your **RMG Map Type Definition**. Go to **Content -> Resource packs** and **Add** *RMG_Definitions_Strategic.rpk, RMG_Sectors_Strategic.rpk* and *RMG_Strategic.rpk* packs to the **required content** list of your **Realm Trait's** *.rpk* file.
4. Go to your **Player Empire: User Setting** resource, in the **Map Type** link, link up your **RMG Map Type Definition**
5. Go to your **Player Empire: World Trait** resource, set its **Type** to *User Settings* in the **Entries** list, link up your **Player Empire: User Setting.**
6. Make sure to save the *.rpk* file by hitting CTRL + S.

Your new Geography trait is now set up and ready to be picked in Realm Customization!

Clime Traits

Clime traits are Realm Traits that define the distribution of Climes and Overlays, creating a very thematic realm such as a tropical realm filled with Tropical and Desert climes, devoid of arctic.

To set up a new Clime trait, follow these steps:

1. Set up a new **Realm Trait** using the steps in Adding Realm Traits.
2. Create a **Player Empire: User Setting** resource in your *RealmTrait[name]Definition.rpk* file, set its **Type** to **Map Geography**
3. Now create a **RMG Geography Template Override** resource.
4. In this new resource, set **Override Theme Weights** and **Override Overlay Weights** to *true*. You can also set any other overrides to *true* you wish.
5. You can now populate the **Theme Weights** and **Overlay Weights** with numbers to indicate the composition of climes/overlays you wish to have the **RMG** use.
    a. The number scale does not matter, as the game will add all numbers together and use that to determine the relative amount of a clime or overlay. But for ease of use, try to use **percentages.**
6. Go to your **Player Empire: User Setting** resource, in the **Map Geography** link, link up your **RMG Geography Template Override.**
7. Go to your **Player Empire: World Trait** resource, in the **Entries** list, link up your **Player Empire: User Setting.**
8. Make sure to save the *.rpk* file by hitting CTRL + S.

Your new Clime trait is now set up and ready to be picked in Realm Customization!

*Tip: You can also include custom content like scripts in Clime traits to, for example, attach Combat Enchantments to all provinces of a particular type! For how to do this, check out Misc Traits**!***

Inhabitants Traits

Inhabitants traits are Realm Traits that change the army compositions of independents and infestations in the realm. To fit a more thematic challenge when conquering the realm.

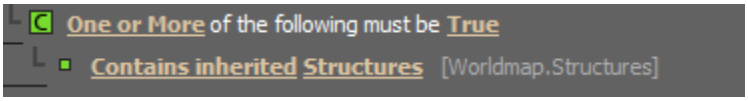
To do this, we'll be needing **Trigger Scripts** and **Tag Filters** that will find and override or infiltrate **Unit Pools** of locations on the world map.

- Infiltrating **Unit Pools** means adding on top of existing Unit Pools, such as adding Dire Penguins as a valid unit to all structure guards!
- Overriding **Unit Pools** means discarding the previously used Unit Pool and replacing it with one of your own choosing. When overriding, make sure there's enough units of varied tiers in the new unit pool.

To set up a new Inhabitants trait, follow these steps:

1. Set up a new **Realm Trait** using the steps in Adding Realm Traits.

2. **Create** a new *.rpk* file, **save** it in your mod's *packs* folder and name it something like "*RealmTrait[name]Content.rpk*".
3. In this new *.rpk* file, create a **Trigger Script**, a **Tag Filter**, and a **AoW Unit Pool** resource.
4. In the **AoW Unit Pool**, define the units and/or unit enchantments and transformations you wish to infiltrate or override existing Unit Pools with!
5. In the **Tag Filter**, specify the filter for world map locations that you wish to change the guarding units of.
    a. If you want them to affect all structures and pickups, use a filter with *One or More of the following must be True: Contains Inherited Structures [Worldmap.Structures]*



    b. If you want to filter more specifically, filter with tags that are part of *Modularity -> Structures*, such as *Modularity -> Structures -> MagicIngredient* or *Modularity -> Structures - > Ocean*.
6. Open the **Trigger Script,** and make sure the triggered [EVENTS] is *A structure spawns units for the first time* if you want to affect structure guards and/or *A pickup spawns units for the first time* if you want to affect pickup guards.
7. In the **Trigger Script's** [ACTIONS]. Use either
    a. **Structure - InfiltratePools** if you want to infiltrate the pools of structures, only use this if *A structure spawns units for the first time* is present in [EVENTS]
    b. **Structure - OverridePools** if you want to override the pools of structures, only use this if *A structure spawns units for the first time* is present in [EVENTS]
    c. **Pickup - InfiltratePools** if you want to infiltrate the pools of pickups, only use this if *A pickup spawns units for the first time* is present in [EVENTS]
    d. **Pickup - OverridePools** if you want to override the pools of pickups, only use this if *A pickup spawns units for the first time* is present in [EVENTS]
8. If you use BOTH *A structure spawns units for the first time* and *A pickup spawns units for the first time.* Make sure to add an IsValid, Structure[Triggering] check to make sure it's either a Structure or a Pickup that triggered the event. Below is an example of such a check:

```
[EVENTS]
  A structure spawns units for the first time
  A pickup spawns units for the first time
[CONDITIONS]
[ACTIONS]
  [IF/THEN/ELSE]
    [IF]
      Bool[IsValid, Structure[Triggering]] == Bool[True]
    [THEN]
      Structure[Set, Megafauna - Unit Pool, Structures & Pickups Tag Filter (Everything), Structure[Triggering]]
    [ELSE]
      Pickup[Set, Megafauna - Unit Pool, Structures & Pickups Tag Filter (Everything), Pickup[Triggering]]
```

9. In the InfiltratePools and OverridePools [ACTIONS], link the **AoW Unit Pool** and **Tag Filter** you've made. And set the target to be **Structure[Triggering]** or **Pickup[Triggering].**

10. Hit **OK**, make sure the **Trigger Script** has **Enabled** and **Active** ticked, but has **Deactivate After Triggering** not ticked.
11. Save the *.rpk* file by hitting CTRL + S.
12. Go back to the *RealmTrait[name]Definition.rpk* file and select your **Player Empire: World Trait** resource.
13. In the **Content File -> Resource Pack File**, type the exact filename of your Realm Trait's content *.rpk* file. If you followed the instructions, it should be called "*RealmTrait[name]Content.rpk*".
14. Save the *.rpk* file by hitting CTRL + S

Note: Do not link the new *RealmTrait[name]Content.rpk* file in your mod's **Required Content** list. It will instead be dynamically loaded when the realm trait is chosen! Else the **Trigger Scripts** will always be active and affect every realm you play with the mod active!

Your new Inhabitants trait is now set up and ready to be picked in Realm Customization!

Presence Traits

Presence Traits are Realm Traits that add one or more specific rulers and empires to the map as a special opponent for the other players to deal with. These special rulers and empires often come with extra starting conditions for them and a Quest for all normal players to defeat the special ruler/empire as a victory condition.

These traits can be very complex. And can use things like **Trigger Scripts**, **Story Events**, **Quests**, and **RMG** resources to create a modified starting empire for the special ruler.

To set up a new Presence Trait, follow these steps:

1. Set up a new **Realm Trait** using the steps in Adding Realm Traits.
2. Create a new **Player Empire: Inhabitants** resource, in this resource you can set up a bunch of settings:
    a. **Alias** fill in a string that will be referenced in scripts to get a reference to this ruler. It's suggested to use Capitals and underscores, EG: *CUSTOM_REALM_TRAIT_RULER_1*
    b. **External Income** which will be extra permanent income for this ruler which can increase the difficulty of conquering this ruler.
    c. **Leader** which contains a link to a **Lord Definition** resource to define the Ruler and its race.
    d. **Leader Locked** which when set to *false* disables **customization** of the ruler. This is not relevant for AI controlled Rulers.
    e. **Leader Level** is the number of **levels** the ruler starts out with at the start of the game. AI's will automatically spend skill points.
    f. **Team** can assign a team from A to D. Rulers with the same team at the start of the game will be locked in an unbreakable **Alliance** with one another.

g. **Controller** can assign an AI difficulty or even a human player as the controller of this Ruler.
h. **Controller locked** which when set to *false* disables the ability for the player to change the **AI difficulty** in the **Advanced Settings** of the realm when creating a realm with this **Realm Trait** active.
i. **Combat Advantage** which determines the assigned combat advantage setting to the ruler. It can modify the damage that is inflicted by and taken by units from the Ruler's empire.
j. **Starting Skills** where you can link Research Skills the ruler will start unlocked with.
k. **Hero Spawning Enabled** which when set to *false*, will have the Ruler not spawn/recruit any heroes apart from their Ruler.
l. **Allow Houses to Match Race** which when set to *false*, will have no houses spawn that are inhabited by the Ruler's race.
m. **Lord Selection List** where you can link multiple **Lord Definition** resources, it will pick a random one from the list to use when the **Realm Trait** is picked in the game.
n. **Allowed Forms** and **Allowed Cultures** which contain lists of **AoW Faction Property Giver** resources, which should be **Form** or **Culture** Properties in the right lists. This will be used to generate/pick a Ruler if **Lord Selection List** or **Leader** are empty.

3. In your Realm Trait's *.rpk* file, go to **Content -> Resource Packs** and **Add** *lords.rpk* to the **Required Content** list.
    a. This will be very handy to link premade leaders, or else have resource dependencies set up to create your own.
4. If you want to create your own custom ruler/empire, then create a **Lord Definition** Resource, link it to the **Player Empire: Inhabitants` Leader**, check out Making/Sharing Premade Factions for how to create custom Rulers/Factions.
    a. If you will be using an already made premade leader, link it to the **Player Empire: Inhabitants` Leader**
5. If you wish to include additional content, such as **Trigger Scripts, Events** and **Quests**, you'll need to create a new *.rpk* file and type its exact filename in your **Player Empire: World Trait -> Content File -> Resource Pack File.**
    a. In this new *.rpk* file, put your custom content. Remember that you can use the **Player Empire: Inhabitants -> Player Settings -> Alias** string to reference a custom ruler in scripting.

Note: Presence Traits are quite complex and prone to bugs when not set up correctly. If you're unsure how to make parts of it work, check out the *Empire_Core.rpk* which contains all the base game Realm Traits. Additionally, when loading in extra content such as **Trigger Scripts**, do not link the new *RealmTrait[name]Content.rpk* file in your mod's **Required Content** list.

Your new Presence trait is now set up and ready to be picked in Realm Customization!

Misc Traits

Misc Traits are **Realm Traits** that modify the gameplay experience in the realm either in the world map or in combat. Which otherwise would not fall within the previous categories. They often provide smaller twists and modifiers that you'd want to combine with other **Realm Traits**. Such as **Combat Enchantments** or Increasing abundance in particular **locations** in the world map.

Misc Traits often load in content *.rpk* files with **Trigger Scripts** to enable their effects. For this example we'll be adding a **Combat Enchantment** to all combats in the world.

To set up a new Misc Trait, follow these steps:

1. Set up a new **Realm Trait** using the steps in Adding Realm Traits.
2. **Create** a new *.rpk* file, **save** it in your mod's *packs* folder and name it something like "*RealmTrait[name]Content.rpk*".
3. In this new *.rpk* file, create a **Trigger Script** resource.
4. In this new **Trigger Script**, make sure **Enabled, Active** and **Deactivate After Triggering** is ticked.
5. Go into the Script and add *Time - NewRound* in [EVENTS]
6. Add *Sector - AddGlobalProperty* in [ACTIONS] and link the **Combat Property** resource you'd like to use.
    a. Go to **Content -> Resource Packs** and **Add** *CombatProperties_Tactical.rpk* in the **Required Content** list to gain access to a lot of the base game's Combat Properties.
    b. Alternatively, you can create your own **Combat Property**.
7. Hit **OK** to confirm the script and save the *.rpk* file by hitting CTRL + S.
8. Go back to the *RealmTrait[name]Definition.rpk* file and select your **Player Empire: World Trait** resource.
9. In the **Content File -> Resource Pack File**, type the exact filename of your Realm Trait's content *.rpk* file. If you followed the instructions, it should be called "*RealmTrait[name]Content.rpk*".
10. Save the *.rpk* file by hitting CTRL + S

Your new Misc trait is now set up and ready to be picked in Realm Customization!

Making/Sharing Premade Factions

Premade factions/rulers are **Lord Definitions** that can be used for multiple purposes, depending on its setting:

- Show up in Front End to be selected when a player selects their Ruler.
- Not show up in Front End, but be able to show up as an AI Ruler
- Be used in certain custom scenarios, such as in **Presence Trait** as shown in Adding Realm Traits -> Presence Traits

To check out how our own Premade leaders were set up, look at *Lords.rpk*.

To set up your own new Premade Faction/Ruler, follow these steps:

1. Create a new *.rpk file* and name it something like "*[modname]Lords.rpk*" and create a Category called something like *Lord Definitions*.
    a. To gain access to a lot of resources that **Lord Definition** resources want to use, go to **Content -> Resource Packs** and **Add** *Lords.rpk* in the **Required Content** list.
2. Create a new **Lord Definition** resource, this is where your custom premade faction will be defined.
3. In the new **Lord Definition**, there are several settings you should set up to make your custom faction to be valid:
    a. **Lord Data -> Essence** should link to *Versatile Essence - Faction Property Giver*
    b. **Lord Data -> Skin** dictates the form of the ruler, should link to any form Property Giver, such as *Humans Form - Property Giver*
    c. **Lord Data -> Culture** dictates the culture of the ruler, should link to any culture Property Giver, such as *Feudal Culture - Faction Property Giver.* Should be the same as the ruler's race, and should even link to a culture if the ruler is a wizard king!
    d. **Customization Items**, where you can link up customization items to adjust the appearance of your custom Ruler.
    e. **Fill Customization Defaults,** which when set to true, will use randomized customization options when there are valid customization items missing in the **Customization Items** list
    f. **Identification Set** which determines the empire's icon and colors, it holds an **IconSet** for the empire's icon, **ForegroundSet** for secondary color and **BackgroundSet** for primary color.
    g. **Name Link** which links to a **Hero Name** resource, you can either link an existing one or make your own.
    h. **Title Link** which links to a string that should be the Title of the ruler, such as *Chaos Prince* or *High Priestess*
    i. **Race Name Link** which links to a **Race Name** resource, you can link to an existing one or make your own.
    j. **Race Bio Link** which links to a string that describes the Ruler's Race, leaving it empty will use the generated Race Bio based on the race's traits.
    k. **Type Template** which links to a **Lord Type Template** resource, you can either link to an existing one or make your own. If you wish to make a Champion, select one of the *AM* Lord Type Templates, if you wish to make a Wizard King, select one of the *WK* Lord Type Templates.
    l. **Leader Tags** where you can link tags to identify this particular ruler. Sometimes used for Scripting or Events.
    m. **Tech Specialization Link** should remain empty.
    n. **Faction Culture** dictates the culture of the ruler's race, this should link to any culture Property Giver.
    o. **Faction Essence** should link to *Versatile Essence - Faction Property Giver*

p. **Faction Skin** dictates the form of the ruler's race and should link to any form Property Giver, such as *Humans Form - Property Giver.*
q. **Faction Customization** list which can force particular customization options to appear on the ruler's race, such as hair colors or armor colors.
r. **Faction Race Traits** where you should link the **Society Traits** you wish to give the ruler's race. Linking race traits together that normally are incompatible can lead to either of them working incorrectly!
s. **Body Trait Override** where you can set the ruler's race's **Body Trait**. If you leave it empty, it will pick the default option of the race's chosen **Form**.
t. **Mind Trait Override** where you can set the ruler's race's **Mind Trait**. If you leave it empty, it will pick the default option of the race's chosen **Form**.
u. **Player Chosen Mount Link** where you can link a mount for the race's units. You can leave this empty for the default horse, or if you've linked a trait that grants a special mount.
v. **Initial Tome** where you can link the tomes this ruler starts with. The default is to pick a single Tier 1 tome if you're making a ruler that's exposed in the premade ruler list.
w. **Hero Properties** which contains a list of unit and hero properties the ruler starts out with. Normally you'll want to link one of the *Equipment Background Traits* of the correct ruler type. Such as *Equipment Background Trait - WK - Spirit Staff* for a wizard king that starts out with a spirit staff.
x. **AI Profile** which should link to *Default Profile*
y. **Personality Trait (Optional)** where you can define the ruler's diplomatic personality. If you leave it empty, one will be randomly chosen each time the ruler appears under AI control.
z. **Can Be Starting Lord** which when set to true can make the ruler appear as an AI faction.
aa. **Can Be Edited** which when set to true will allow the player to adjust the appearance of this ruler if they come under control of this ruler.
bb. **Available in Game Lobby** which when set to true will show this ruler in the Premade Ruler list when picking a new ruler to start a game with.
cc. **Display Order** which will dictate the order of this ruler when displayed in the premade ruler list. Only works if **Available in Game Lobby** is set to true.
dd. **Tags** which are tags to identify this Lord Data set. These are not given to the Ruler unit itself, for that, use **Leader Tags**

When all these have been set up with valid data, your ruler is ready to be used for multiple purposes, either as a premade ruler, as a special AI ruler, or for Presence Traits!

Adding Leader Customization Items

To add customization items for use in the *Faction Creation Appearance* step, we need the following:

- A *Content Library* containing all the meshes, textures and materials we want to have as customization items
    - The mesh requires to be **skinned** to a **bone** within the respective **rig** of the leader type we're creating a customization item for. More on that in the **FBX Notes** section
- A *Modularity* setup to be able to use these meshes in the game

For this section we also have some example files again. These are provided alongside this guide. The relevant files are:

- *Block_Head.fbx*
- *Block_Head_Diffuse.png*

Create a Content Library containing the above. We need a mesh, with a material; using the supplied texture. If you don't know how to do this yet, check out the **Adding Content Libraries** section. We're saving the content library under the *Libraries* directory in the mod's folder hierarchy, under the name *Block_Head_Meshes.clb*.

Of course, you can also use your own meshes, which you can set up through the **Preparing For Export - Skinned Meshes** section under the **FBX Notes**. For now let's keep things simple and implement the *Block Head* helmet customization we have provided.

Step by Step Guide

1. In the *Package Manager*, go to **Tools > Resource Editor**
2. First of all, **immediately save (CTRL+S)** the package somewhere in your Mod's *Packs* directory. We're saving the pack under the *Block_Head_Modularity.rpk* file name.
3. Create a new *Category* by clicking on **New** next to the *Category:* dropdown menu
4. Name it *Default* and click **OK**
5. Go to **Content > Resource Packs** in the upper bar
6. Click the **Add** button and **navigate** to *Content\Title\Packs\Unit_Modularity* in the base game's directory hierarchy. **Open** the *Shared_Modularity.rpk* file and then do the same for the *Hero_Customization_Categories.rpk* file, in the same directory.
7. Click **OK** again, the pack should **refresh**
8. Do the same for **Content > Libraries** and choose the *Block_Head_Meshes.clb* that you have saved out earlier. The pack should **refresh** again.
9. **Select** the 0th entry and change the *Type:* dropdown under *Main* to *Modularity: Provider*. Click on **New** next to it.
10. **Rename** the resource to *Helmet Type: Block Head*
11. **Change** the *Entity Type* property as follows: **Category** = *Category Types*, **Resource** = *Category Type: Unit*
12. **Click** the yellow/orange square next to the *Tags* property
13. A new dialog should open, **right-click** anywhere in the window and choose **Add**
14. **Search** for *RemoveHair* and **select** it. Then, click **OK** and **OK** again.

15. **Select** the empty resource entry at **index 1** and change the *Type:* dropdown under *Main* to *Modularity: Component*. Click on **New** next to it.
16. **Rename** the resource to *Component: Block Head*
17. **Click** the yellow/orange square next to the *Unlocked By Providers* property
18. **Search** for *Helmet Type: Block Head* and add it to the list, then click **OK**
19. **Expand** the *Unlocked By Category* property and change the fields as follows:
    Category = *Categories*, **Resource** = *Category: Helmet*
20. Now **click** the yellow/orange **+** button and choose **Mesh**
21. **Right-click** the 0th entry and choose **New**
22. **Expand** the *Mesh* property in the right hand panel
23. **Change** the fields as follows:
    Library = *Default { BLOCK_HEAD_MESHES }*, **Resource** = *Block_Head_Diffuse*
24. You should now see the mesh in the Viewport
25. **Select** the empty resource at **index 2** and change the *Type:* dropdown under *Main* to *Customization Item*. Click on **New** next to it.
26. **Rename** the new resource to *Customization Item: Block Head*
27. **Change** the following properties:
    a. *Category*: **Category** = *Leader*, **Resource** = *Leader: Helmet*
    b. *Provider*: **Category** = *Default*, **Resource** = *Helmet Type: Block Head*
    c. *Sort ID*: **1000**
28. **Save** the library (**CTRL+S**)
29. Go back to the *Package Manager* and go to **Content > Resource Packs**
30. **Add** the newly created *Block_Head_Modularity.rpk*

You can now **save** your mod, set up a *Playset* for it and start up the game. As we've given the head item a very high sorting ID, it'll be the last entry in the Faction Creation's *Appearance* tab. **If it doesn't appear, please make sure that you've added the package in step 30.**

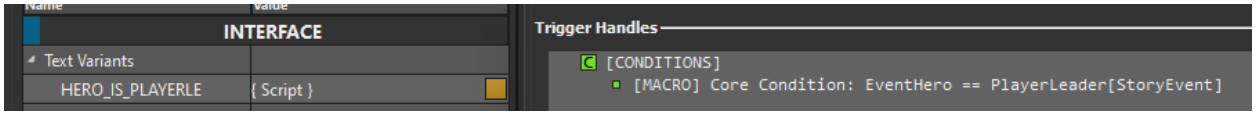
Making a Story Event

Story events are narrative pop-ups during gameplay where a situation is laid out and several choices are offered to the player with gameplay consequences. These are completely new in Age of Wonders 4 and so the resource set up for it is also new. They use the Scripting system to identify when they should appear, what happens by default and what each button's effects are! It is advised to also look into how to Adding Text since Story Events need plenty of custom text!

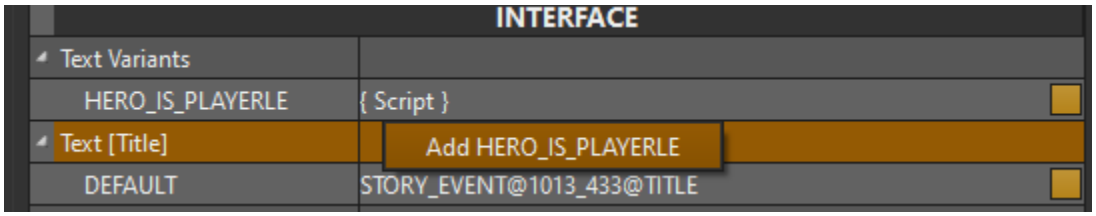
Story Events Consists of

- **Event Texts**
    - Title
    - Header Text
    - Body Text
    - Footer Text
    - Button(s) Text

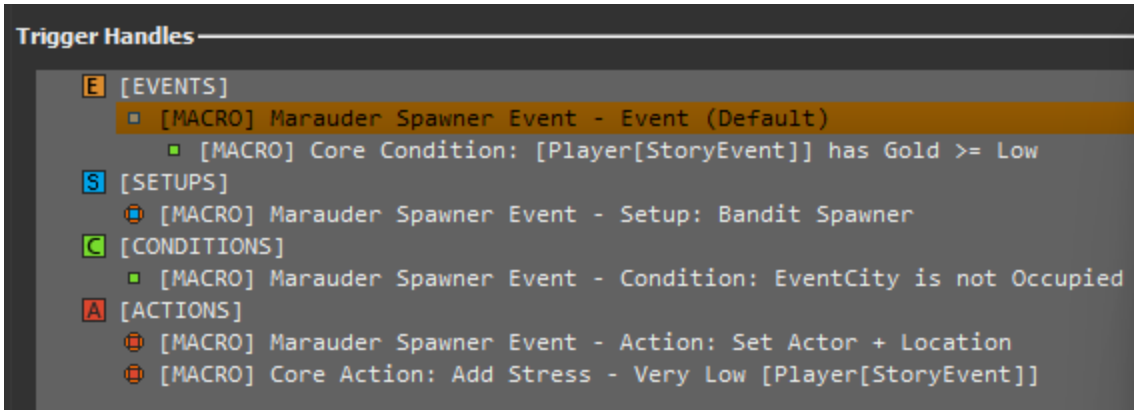
- **Text Variants**
    - These can be set up so that certain text entries get replaced by different text if conditions are met, such as if the hero of the event is a ruler.



    - You can add alternative text entries when these conditions are met by right clicking on any of the text entries and selecting *Add [textvariantname]*



- **Scene Visuals**
    - **Scene Settings**, you can link to existing ones, or set up a **Story: Scene Settings** and **Modularity: Scene**
    - **Story Tags,** which uses the tag system to further modify the scene visuals
- **Scene Type**
    - This is essentially the category this story event belongs to, it is used to determine the rarity and pace of the event. Certain categories will not guarantee that the event will spawn every time the condition is met, which is how we do “random” events without spawning them every turn. You can either link to an existing one or create your own **Story: Event Type**
- **Script - On Trigger**
    - The script that determines when this story event should start. Unlike normal scripts, these scripts also have **Events** and **Setups**
    - **Event** is where you can determine the exact Event(s) the script should listen to.
        - Under an **Event** you can set up additional conditions
        - **Events** often say which data they will pass on in the event, such as *Player[Triggering].*
    - **Setup** is where you can set up variables to use in this story event (or in a global pool for later). It’s often useful to store the things like triggering stack or player for later use. The Setup part of a script will still happen even if Conditions aren’t met. Unless they are conditions in **Event**
    - **Conditions** which is where you can check for extra conditions to see if the event should be valid. None of the actions and the story event itself will trigger if the conditions aren’t met.
    - **Actions** where you can add special script actions that can affect the game or data, such as destroying structures, spawn units, or set diplomatic standings. These will go off as soon as the Story Event is valid and triggered (so before any choice is made by the player)



Example of a Story event On Trigger script using Macros

- **Script - Post Event**
    - A script that triggers when the event is resolved (so any of the choices were clicked by the player).
- **Buttons**
    - You can dynamically add buttons by right clicking on the **Buttons** field and clicking **Add**
    - Each button has:
        - **Type** which determines the icon presented next to the event. Useful for letting the player know what kind of choice this is.
        - **Text Button** which is the text of the button, can also have variants when there are **Text Variants** set up.
        - **Script** the script contains **Conditions** to determine if this button can be pressed, and **Actions** to execute if the button is pressed by the player.
        - **Condition Not met Behavior**, what the button should do when the **Conditions** in the **Script** are not met
            - **Disable** will simply remove the button from the list
            - **Hide** will grey out the button and means the button cannot be clicked
            - **Hide with Highlight** does the same as **Hide**, but will highlight the button in yellow to communicate the importance of this button.
- **Tags** is where you can add tags that can be used to identify this event by the event flow system. Check tags under *Ranking* for the kind of tags you can add.
- **Sentiment** whether this event is considered positive, negative or neutral for the player.
- **Rarity** is the priority for this event when multiple events are valid under the same conditions. Highest rarity events will take priority over more common ones.
- **Pace** is how much impact this event has on the flow. A heavier pace setting means that the game will try to spawn less events and/or less heavy events for the player afterwards. Best kept at *Inherited* which takes the linked **Event Type** pace value.
- **Weight Filter** and **Override Weight Filter Value.** Used to apply an additional priority filter through events on top of Rarity. Best to keep this on *Inherited* and *False*

To Create a new Story Event, follow these steps:

1. Create the text for the **Title**, **Header, Lore, Footer** and each **Button** by following Adding Text
2. Open Resource Editor and create a new *.rpk file*
3. In this new file, go to **Content** -> **Language** -> **Add** and select your mod's *.mo file*
4. Also go to **Content** -> **Resource Packs** -> **Add**, find and select *Events_Core.rpk*.
    a. This contains a lot of handy macros and resource dependencies often used for events such as premade scenes and
5. create a new **Category** and in it, create a **Story: Event** resource.
6. Set up this story event
    a. Assign text to each text entry.
    b. Link or create **Scene Settings**
    c. Set the **Event Type**
    d. Add **Buttons** and link text to them.
    e. Set up **Scripts** for at least the **Script - On Trigger** to indicate when the event should spawn. But also add **Scripts** to buttons to determine the outcome when clicked.
    f. Set up the Event's **Tags, Sentiment, Rarity, Pace** and **Weight Filter**, where possible, you may want to just choose *Inherited* on some options.
7. Make sure this new *.rpk file* is loaded in with your mod's *.acp file*.

Your new event should now be ready to appear in game! Remember to check existing scripts and events . Event Types may also cause your event to not always appear even when trigger conditions are met due to cooldown, pacing or new game time outs.

*Tip: while scripting, there are a lot of MACROs you can use for a lot of event actions. To add a Macro in your script, select a **Core - Macro** entry when choosing a Condition Type, Event Type, Action Type or Setup Type to add to your script. Then you can select from existing macros to link.*

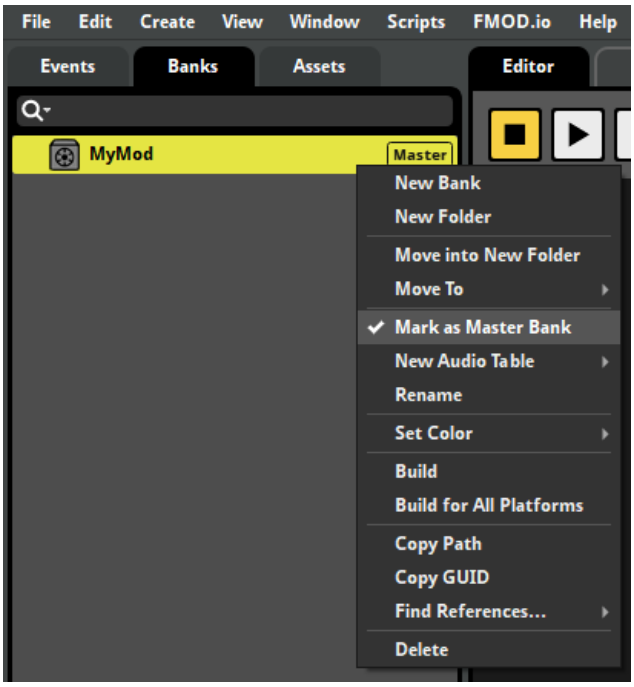
Adding Audio

The creating and mixing of Audio in Age of Wonders 4 is done through **FMOD Studio 2.0**. The assigning these sound effects to actions in game is done though the Resource Editor. This guide will not go into detail about working in FMOD Studio.

Project Setup

1. Download the *.fspackage* file from the example files provided alongside this guide.
2. Open FMOD Studio and open the *.fspackage* file.
3. FMOD Studio will give a popup that some folders are missing. This is normal, just press **Recover** and the folders will be created for you.

4. Next you'll need to add a **bank** for your mod and mark it as a **Master Bank**. When creating new events you'll want to assign them to this bank.



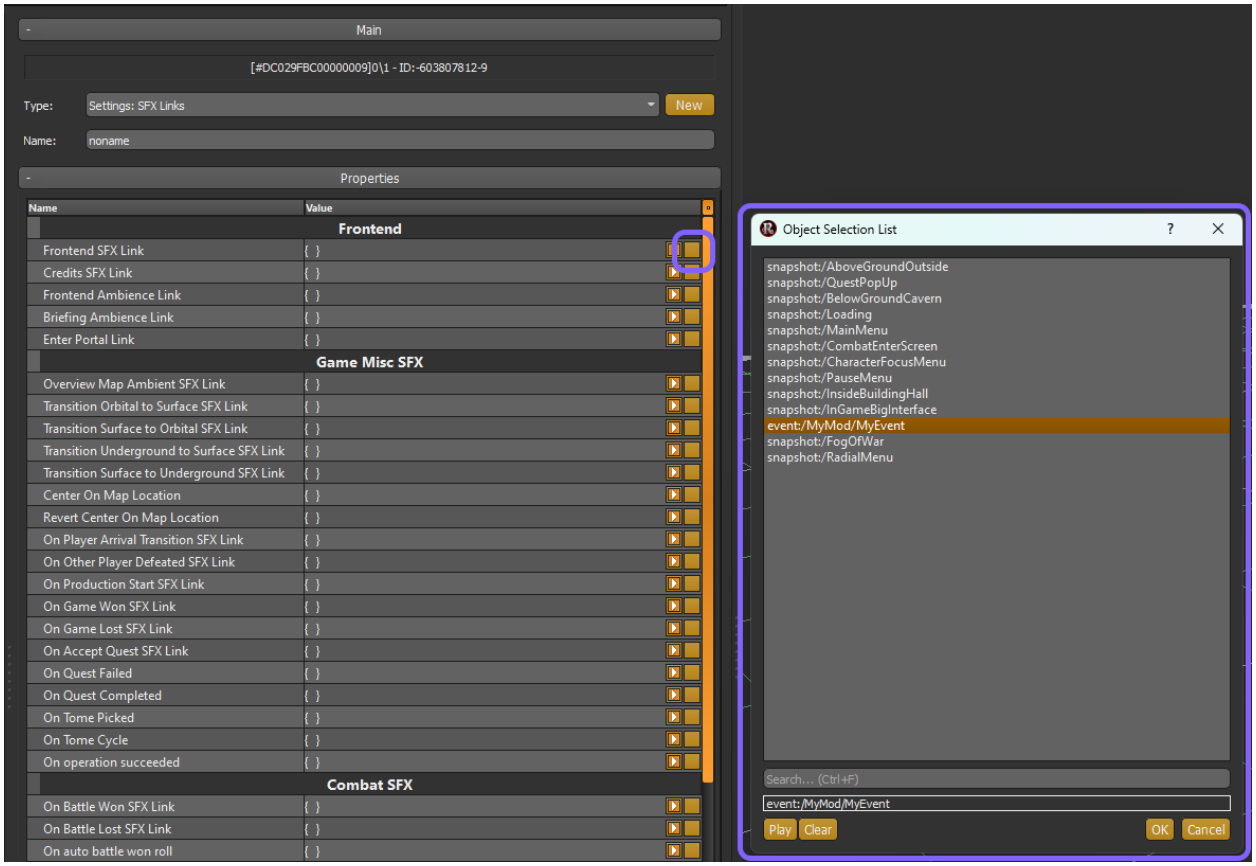
Resource Setup

After you're done making/editing events, you'll have to **build** your banks and **copy** them over to the **mod's Audio Folder**.

1. In the package manager, open the resource editor and make a new package.
2. Add the package to the Package Manager via **Content > Resource Packs > Add**.
3. In the Resource Editor, go to **Content > Audio > Add** and add the **bank** files from the **mod's Audio Folder**. This gives you access to the audio events in the bank.

Changing Action Sound Effects

1. In the Resource Editor, add the resource **Settings: SFX Links** to your Resource Pack.
2. Scroll down the properties of the newly created resource and set the **Priority** to something higher than 10. This value determines which version of the resource overwrites which. This is especially important when a player has **multiple mods** enabled.
3. By clicking the **orange square** of one of the properties you can assign it an audio event to override the normal one.



Changing Interface Sound Effects

1. In the Resource Editor, open *Audio.rpk* in *Title**Packs*\. This pack contains all the base game's audio links and settings.
2. **Link** any custom made banks you may have as described in the previous section.
3. Switch to the category **Noesis SFX Tags**.
4. Find the tag you want to change. These tags come from the UI's *.xaml* files and are automatically found by the game when loading in the UI.
5. By clicking the **orange square** of the Sound property you can assign it an audio event to override the normal one.

Adding Music

1. In the Resource Editor, add either the **AoW Music: Strategic ONLY music** or **AoW Music: Tactical ONLY music** resource.
2. By clicking the **orange square** of the Music Link property you can assign it an audio event.

Optional: Strategic music has a **Tag Filter** property that can be used to determine the appropriate moment for this music to play. To gain access to the tags, you'll have to link the *Audio.rpk* found in *Title**Packs*\. Go to **Content > Resource Packs > Add** to add the *Audio.rpk*.

Glossary

Package Manager: The tool used to create new mods. Can be found next to the AOW4.exe in the game files.
Resource Editor: The editor used to edit existing content and to add new content for mods.
Resource Package: Often shortened to Package or rpk. A package is used in the Resource Editor. It is a collection of resources that is grouped together into a single file with the .rpk extension.
Resource: A resource is a single line of logic found in the Resource Editor. It can contain anything depending on the resource from Units to Targeters for Spells to definitions of entire gameplay systems.
Category: These are tabs found in Resource Packages. These are purely for organizational purposes.
Playset: Used in the Paradox Launcher. It is used to select which mods you want to have active when launching the game.