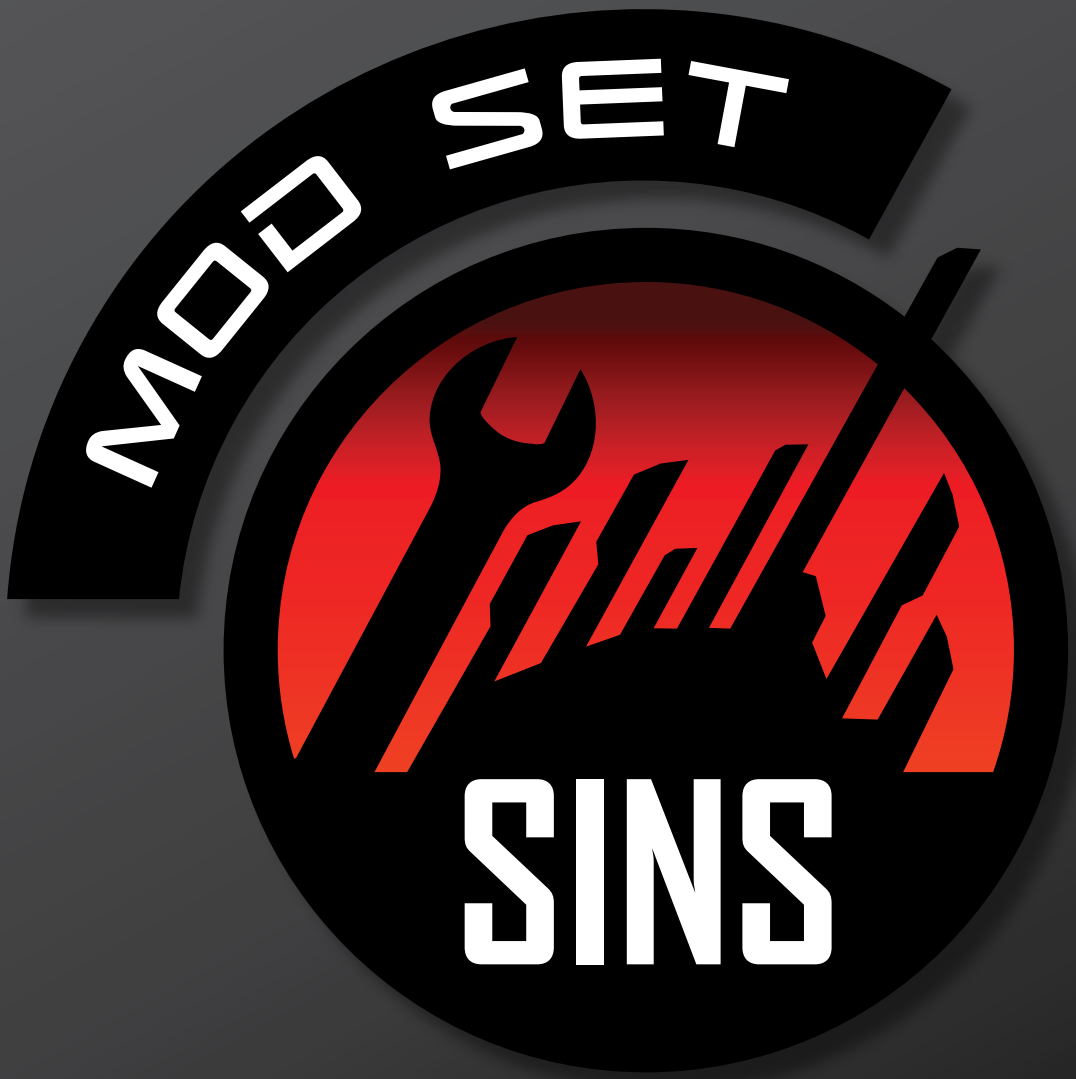

A MODDERS GUIDE TO SINS OF A SOLAR EMPIRE

Modding Basics

Note: This early draft of the modding instructions is far from completion. Hopefully this will be sufficient to get you started.

How Mods work

The engine will attempt to find the files it needs in mods before searching the root game path. Mods work by simply providing the different files to use.

Setting up mods

1. Find your `<Mod Path>` that the game will use.
   - Start the game and open the Mods screen. (Options->Mods).
   - Click the "Show Mod Path" button.
   - A dialog will appear telling you where to place your mods.

Example:

```
C:\Users\<username>\AppData\Local\Ironclad Games\Sins of a Solar Empire\Mods
```

2. Create a new directory here. The name of the directory will be the name of the mod. You must restart the game to see any new mods. You won't have to restart the engine to change a mod, just to create a new one and have it appear in the list.

3. You should have included with this package a "Reference Data" folder. We have provided these files because the game shipped with most of the data files converted to binary for optimization. The reference data files are text versions of any these files so that they'll be easier to read and modify. The engine doesn't care if the data files are in binary or text. If you can't find a file in the Reference Data folder this means the original file in the Sins directory didn't need to be converted and can be used as is.

Example 1: Remove Advent

Summary: We are going to remove an entire race from the game. No one will be able to play the Advent in our new Mod.

1. Create a new directory in your `<Mod Path>` called "01 - Remove Advent".

Example:

```
C:\Users\<username>\AppData\Local\Ironclad Games\Sins of a Solar Empire\Mods\01 - Remove Advent
```

2. Copy your "GameInfo" directory from `<Reference Data>` to `<Mod Path>`

Example:

```
C:\Users\<username>\AppData\Local\Ironclad Games\Sins of a Solar Empire\Mods\01 - Remove Advent\GameInfo\
```

should now have many .entity files and other important data files.

3. Delete `<Mod Path>\GameInfo\PlayerPsi.entity`. (Psi was our code word for Advent, Phase is the code word for Vasari, Tech is the code word for the TEC).

4. Restart the game and enable your new mod. This may take a few seconds, the engine is restarting to use your new mod.

5. Now setup a new game. Note that you can no longer select the Advent race for you player. It has been removed.

Example 2: Weak Advent

Summary: We are going to add the advent back in, but they won't have the technology to build all of their different capital ships.

1. Create a new directory in your <Mod Path> called "02 - Weak Advent"

Example:

```
C:\Users\<username>\AppData\Local\Ironclad Games\Sins of a Solar Empire\Mods\02 - Weak Advent
```

2. Copy your "GameInfo" directory from <Reference Data> to <Mod Path>

3. Open up <Mod Path>\GameInfo\PlayerPsi.entity in your favorite text editor.

4. Search for "capitalShipInfo". It should be on line 29 in the file.

5. Right below "capitalShipInfo", change the "count 5" to "count 1", then delete the next 4 lines:

```
entityDefName "CAPITALSHIP_PSIBATTLESHIP"
entityDefName "CAPITALSHIP_PSIPLANETPSIONIC"
entityDefName "CAPITALSHIP_PSICOLONY"
entityDefName "CAPITALSHIP_PSICARRIER"
```

You should now see:

```
capitalShipInfo
        Page:0
                count 1
                entityDefName "CAPITALSHIP_PSIBATTLEPSIONIC"
```

6. Save the file, restart the game and enable this mod. Start the game as advent and build you Capital Ship Factory. Note that you can now only build one type of capital ship!

Example 3: Not So Weak Advent

Summary: We are going to buff up the only capital ship advent can build from Tutorial 02.

1. Create a new directory in your `<Mod Path>` called "03 - Not So Weak Advent"

2. Follow steps 2-5 in Tutorial 02. Or simply copy the `GameInfo` from Tutorial 02 into this tutorial's directory.

3. Open up CAPITALSHIP_PSIBATTLEPSIONIC.entity in your favorite text editor.

4. Search for "`MaxHullPoints`". You should see "`StartValue 2100.000000`" right below it. Change this to "`StartValue 1337000`". Save and close this file.

5. Copy your "`String`" directory from `<Sins Path>` to `<Mod Path>`.

6. Open up `<Mod Path>\String\English.str` in your favorite text editor.

7. Search for "`IDS_CAPITALSHIP_PSIBATTLEPSIONIC_NAME`". Right below this should be:

```
Value "Rapture Battlecruiser"
```

change this to:

```
Value "Super Duper Cruiser"
```

5. Save the file, restart the game and enable this mod. Start the game as advent and build you Capital Ship Factory. Now build your only Capital Ship. Note that the infocard now has the new name and the ship has new hull points.

Mesh Format

Almost everything should be self explanatory after examination of any mesh file in text form except for the following:

`hasValidTangents`: This tells the engine whether the mesh already has valid tangents or not. If a mesh doesn't provide valid tangents they will be generated by DirectX. This is not preferred as 3D packages tend to do a better job.

`NumCachedVertexIndicesInDirection:<FRONT,BACK,LEFT, RIGHT,UP,DOWN>`: These indice lists help the engine optimize a number of operations (in particular weapons fire) and each list is a sample of indices that reference vertices with normals that are within some min angle of the direction indicated and are sorted by distance to the origin of that direction. The origin of that direction is defined as the unit normal vector of that direction * mesh radius and the min angle is ((90 + 45) / 2) degrees.

The text versions of all mesh files are found in `<Reference Data>\Mesh`.

Mesh Texture Formats

Sins of a Solar Empire uses two primary texture encoding systems, one for ships (which also includes asteroids and structures) and one for planets. More mesh, texture, and rendering considerations will be covered in the future.

Ships

Textures are referenced from .mesh files in the following form:

```
DiffuseTextureFileName "example-cl.dds"
SelfIlluminationTextureFileName "example-da.dds"
NormalTextureFileName "example-bm.dds"
```

`DiffuseTextureFileName` is expecting a color texture.

`SelfIlluminationTextureFileName` is expecting a data texture.

`NormalTextureFileName` is expecting a normal texture.

Color Texture Format:

red - color

green - color

blue - color

alpha - specular

Data Texture Format:

red - team color

green - self illumination

blue - reflection map

alpha - bloom

Normal Texture Format:

This texture expects a normal texture using NVIDIA's DXT5_NM format. A free tool to export normal maps in this format can be found here: http://developer.nvidia.com/object/photoshop_dds_plugins.html

Note: If your mesh doesn't need a given texture (eg. asteroids don't need a data texture), it should refer to "Black-da.dds". This is because some video cards interpret no texture as all white when Sins expects all black.

Planets

Textures are referenced from .mesh files in the following form:

```
DiffuseTextureFileName "example-cl.dds"
SelfIlluminationTextureFileName "example-da.dds"
```

`DiffuseTextureFileName` is expecting a light side texture. `SelfIlluminationTextureFileName` is expecting a dark side texture.

Light Side Texture Format:

red - light side color red

green - light side color green

blue - light side color blue

alpha - specular

Dark Side Texture Format:

red - dark side color red

green - dark side color green

blue - dark side color blue

alpha - unused

Creating Sins Meshes from XSI Meshes

The xsi format is a popular mesh format from the SoftImage|XSI 3D modelling package. Most popular 3D modelling packages can export into the xsi format. You can convert .xsi files into text versions of Sins.mesh files by using the provided command line tool "`ConvertXSI.exe`". By default meshes are optimized (We only use unoptimized for internal debugging purposes).

The syntax is

```
convertxsi <src> <dest> <--nooptimize>
```

Examples:

```
convertxsi mymesh.xsi mymesh.mesh
convertxsi mymesh.xsi mymesh.mesh --nooptimize
```

Converting Sins Data Files into Binary

Once you have completed creating or editing any Sins text data files you may want to convert them into binary to speed up load times. This can be done by using the provided command line tool "ConvertData.exe".

The syntax is

```
convertdata <type> <src> <dest>
```

The valid types are: mesh, particle, brushes and entity.

Example:

```
convertdata mesh mymesh_text.mesh mymesh_binary.mesh
```

Ability Modding

1. Introduction

- Abilities provide a lot of the racial differentiation between the races, especially at the tactical level.
- All ability files appear in the "GameInfo" directory.
- Capital ships, cruisers, frigates, structures, and stars can all have up to 4 abilities, and so can planets with some exceptions.

2. Changing abilities on entities

To change which entities have which abilities, look for lines with

```
ability:0 "<ability name 0>"
ability:1 "<ability name 1>"
ability:2 "<ability name 2>"
ability:3 "<ability name 3>"
```

and replace "<ability name #>" with the name of the new ability you want the entity to have.

Until you are more familiar with the details and dependencies of abilities, it is recommended that abilities be swapped between entities of the same type, and belonging to the same race (eg. moving abilities from one capital ship to another, or from one frigate to another frigate)

3. Modding Abilities

Abilities achieve a lot of their functionality with another object type called buffs. For the majority of abilities, you can think of the ability as a bootstrap or delivery system for a buff payload that actually does the 'work' of the ability.

Lets look at a simple example: `AbilityAbilityGuidance.entity`:

- This ability normally improves the ability cooldown rate of a targeted friendly ship, but let's use it as a template for a new ability called Hinder Abilities that hinders the cooldown rate and increases the antimatter cost of abilities of enemy ships instead.
- First, copy the ability from `<Reference Data>` into `<Mod Path>\GameInfo\` and rename the copy '`AbilityHinderAbilities.entity`', then open the copy.
- The first distinguishing line is:

```
buffInstantActionType "ApplyBuffToTarget"
```

- You can think of `BuffInstantActions` as 'fire and forget' building blocks of ability functionality. This particular one does as it's type name suggests: it applies a buff entity to the target object specified by the player.

Many of the following lines are parameters for this `BuffInstantAction`, some of which will be explained a little later. The ones we are interested in for this new ability are the next two:

```
buffType "BuffAbilityGuidance"
targetFilter
```

`buffType` denotes the buff entity that will be applied to the target, so you will want to find the file '`BuffAbilityGuidance.entity`' and copy and rename it to '`BuffHinderAbilities.entity`'. Then change the line in the ability file we've been working with to read

```
buffType "BuffHinderAbilities"
```

So it points to the new buff placeholder we just created.

`targetFilter` is a sub-object that describes what types of things are valid targets for this `BuffInstantActionType`. Some types may not have any target filter at all (e.g. abilities that apply the buff directly to the object with the ability)

`targetFilters` have three lists:

1) the ownership list specifies what relationship types are valid with respect to the target relative to the entity using the ability. More elements in this list broadens the scope of valid targets.

2) the object list specifies which types of objects are valid targets. More elements in this list broadens the scope of valid targets.

3) the constraint list specifies additional properties of the target that must all be satisfied for it to be valid. Unlike the first two lists, more elements in the constraint list narrows the scope of valid targets.

Since we want Hinder Abilities to affect enemy ships, replace the ownership parameter value "Friendly" with "Enemy".

Also, now that this ability targets only enemy ships, we no longer need the line

```
constraint "NotSelf"
```

so go ahead and remove it entirely. Don't forget to decrease the count of numConstraints by 1 as well.

Lastly, don't forget to change the name and description strings at the bottom of the ability to new ones you'll add to the strings file:

```
nameStringID "IDS_ABILITY_ABILITYGUIDANCE_NAME"
descStringID"IDS_ABILITY_ABILITYGUIDANCE_DESCRIPTION"
```

All the changes required in this file are complete, so go ahead and save it. In order to have Hinder Abilities work as desired though, some additional changes are still required to the other file you copied and renamed: 'BuffHinderAbilities.entity'.

4. Modding Buffs

- Open 'BuffHinderAbilities.entity'
- For the most part, buffs are sort of like the target filter you modified in the last section: a collection of lists of objects. Instead of target properties like in the target filter, a buff's lists contain different kinds of actions that the buff will perform.

  - the first action type is InstantActions:
    You may remember that these are the same type that abilities have, although unlike abilities, buffs are not restricted to having a single instant action. Also unlike abilities, buffs are more flexible in the conditions that can trigger these actions.

- the next action type is `PeriodicActions`, which are a way of performing `InstantActions` repeatedly on a schedule.

- next up are `OverTimeActions`, which usually are actions that have an updating, ongoing effect on the object with the buff on it.

For `BuffHinderAbilities`, we won't need to modify any of these action types. In addition to these actions, buffs also have lists of modifiers that simply change the state of the entity the buff gets applied to:

- `EntityModifiers` are generally percentage change bonuses or penalties to specific attributes (eg. weapon damage or armor)
- `EntityBoolModifiers` are binary state changes (e.g. whether this unit is immune to damage or not)

To achieve the goal of making Hinder Abilities reduce the cooldown rate, you'll need to modify the existing `entityModifier` for "`AbilityCooldownRate`" to negative values.

Let's make the increased antimatter cost of abilities be +50% for all levels. To do that we'll need to add another modifier for "`AntimatterCostOfNonUltimates`" like this:

```
buffEntityModifierType "AntimatterCostOfNonUltimates"
        value
                Level:0 0.50000
                Level:1 0.50000
                Level:2 0.50000
```

Don't forget to increment the count of `numEntityModifiers` by 1 to account for the newly added modifier.

The last major aspect of buffs is finish conditions, which as you might expect control when the buff will expire. Most buffs use a simple

"TimeElapsed" condition, but other more specialized conditions also exist. The existing finish condition is already suitable for Hinder Abilities, so leave it unchanged. Using the instructions of part 2, go ahead and assign this new ability to a capital ship and try it out!

5. Closing Comments

The full potential and flexibility of buffs lies in learning how to combine these independent parts of buffs in interesting ways.

- A key part of this is recognizing that many instant actions can spawn new buffs of their own, each potentially having different properties and behaviours than their parent buff.
- Many abilities follow this pattern

```
AbilityEMPBlast.entity/BuffEMPBlastSpawn.entity
BuffEMPBlastAction.entity
```

so look at some of these abilities to see how these relationships work.

- Poking around in other abilities will also reveal more action types and modifiers that are available for creating your abilities and buffs.

Have Fun!
More to Come...
- Ironclad