Dark Ages Modding

Information specific to modding DOOM: The Dark Ages

Installing Mods (DOOM: The Dark Ages)

> Installing **DOOM: The Dark Ages** mods work the same way as installing **DOOM Eternal** mods from BEFORE the **DOOM Mod Portal** was introduced.

> Currently, only **Steam** is officially supported.

Installing the Mod Loader

1. Download the **Atlan Mod Loader**:

   https://github.com/FlavorfulGecko5/EntityAtlan/releases/tag/ModLoader

   Scroll all the way to the bottom under **Assets** and download `AtlanModLoader.zip`

2. Navigate to your DOOM: The Dark Ages installation folder.
   The default path on **Steam** is `C:\Program Files (x86)\Steam\steamapps\common\DOOMTheDarkAges`

3. Unzip all the contents of **AtlanModLoader.zip** into your DOOM: The Dark Ages installation folder.
   The file, `AtlanModLoader.exe` should be in the same folder as `DOOMTheDarkAges.exe`

Loading and Unloading Mods

1. In your DOOM: The Dark Ages installation folder, open the `mods` folder.

2. Place all your downloaded game mod `.zip` files into the `mods` folder.
   Do not unzip the mods!

To **install** your mods, either:

- Run `DarkAgesModManager.exe` for a friendly interface to manage your mods.
- Run `AtlanModLoader.exe` to quickly install your mods and launch the game.

To **uninstall** your mods, either:

- Uncheck them in `DarkAgesModManager.exe` then run the mod loader.
- Manually move them out of the mods folder and run `AtlanModLoader.exe`

Troubleshooting

To report a bug or get help with troubleshooting, join the DOOM 2016+ Modding Discord server:

https://discord.com/invite/ymRvQaU

Common Questions/Issues:

1. The game is crashing on startup when mods are installed:

   Delete the file modloader_cache.bin located in your installation folder, then run AtlanModLoader.exe again.

2. My Anti-Virus software is flagging Atlan Mod Loader as a virus:

   These are false positives, which may or may not happen depending on your anti-virus setup. Ignore the warning and allow the mod loader to run anyway.

3. What platforms are supported:

   Currently, only Steam is officially supported. If you'd like to help add support for other storefronts, come join the modding Discord linked above.

   Atlan Mod Loader also does not support pirated versions of the game, and we cannot help you with them.

Making Mods: Getting Started

A high-level overview of what's possible with the Atlan Modding Tools. If you have previous experience modding Doom Eternal, this page explains the critical differences between the two games.

Dark Ages vs Eternal

Almost no differences exist between the Dark Ages and Eternal file system. However, major discoveries have simplified the modding process. Massive issues Eternal modders faced are irrelevant in Dark Ages.

Eternal	Dark Ages
Mod loader replaces the files inside of the vanilla game archives.	Mod loader builds it's own resource archive. The vanilla game archives are untouched
Modders had to know what archives contained which assets, and replicate this setup in their mods.	Not an issue. The modded resource archive is loaded globally, so modded assets will always be usable. Modders don't need need to consider where their assets should be stored.
Adding new assets to an archive, instead of replacing existing assets, could cause instability.	Not an issue. Add as many new assets as you want!
Game updates frequently broke mods due to resource archive priorities changing.	Not an issue. The modded resource archive will always have the highest priority.

This doesn't make your mods immune to game updates. When a game update edits a vanilla file, you should update your mods to use that new version. Otherwise, your mod will be outdated because it uses a previous update's file as a baseline. The consequences will vary significantly depending on the mod. It may range from nothing happening, to game balance being off, to core mechanics not working, to game crashes.

What vanilla files are modified?

In total, Atlan Mod Loader edits 4 vanilla files:

```
DOOMTheDarkAges.exe
base/packagemapspec.json
base/meta.resources
base/soundmetadata.bin
```

The modded resource archive is `base/modarchives/common_mod.resources`

The modded StreamDB archive is `base/modarchives/common_mod.streamdb`

The modded audio archive is `base/sound/soundbanks/pc/ATLANMOD.snd`

What is Possible?

Atlan Mod Loader supports modding the following types of files:

- Decl Mods - (Located in `decls/`)
  - If you used an older version of Atlan Resource Extractor, they will be located in `rs_streamfile/generated/decls`
- Strings - In Dark Ages, strings are decls located in `decls/string/`
- Entitydefs - See **Serialization and Atlan Mod Packager**
- Logic Decls - See **Serialization and Atlan Mod Packager**
  - Only existing values can be edited. You cannot create new variables, nodes, pins, or otherwise modify the structure of the graph. Attempting to do so can cause game crashes. These restrictions exist due to the `logicObjectDescriptor` files, which are pre-compiled versions of the decls.
- Map Entities - See **Serialization and Atlan Mod Packager** and **Dark Ages Level Modding**
- Sound Mods - See **Dark Ages Audio Mods**
- Texture Mods - See **Dark Ages Texture Mods**
- Font Mods - See **Dark Ages Font Mods**

Building Your Mod

Before building your mods, you must use **Atlan Resource Extractor** to obtain the game's files. **Please thoroughly read the information on the download page and in it's configuration file.**

To add a vanilla file to your mod:

1. Copy it from the vanilla resources into your mod folder
2. Replicate the filepath produced by Atlan Resource Extractor

Example: Let's say you copy the following vanilla file into your mod folder

```
decls/string/ui/mainmenu/collectible_viewer.decl
```

You must replicate this filepath in your mod folder. This should be it's path when your mod is zipped up.

Fake Directories

A goal of Atlan Mod Loader is improving the developer experience when making mods. One way it does this, is by making directories optional. Instead of creating *literal* directories, add an `@` symbol in the filename. The mod loader translates these symbols into directories for you!

Example: Let's say we want to make a mod using the file:

`decls/string/ui/mainmenu/collectible_viewer.decl`

This file is nested inside of 4 directories! That's a real pain to create and navigate through each time you need to open the file! Instead of creating those directories, just name the file:

`decls@string@ui@mainmenu@collectible_viewer` `.decl`

You can also mix and match! For Example:

1. Have a folder named `decls@strings`
2. Name the file `ui@mainmenu@collectible_viewer` `.decl`

Find a setup and style most convenient for you and the mod you're making!

Mod Config. File

Atlan Mod Loader uses `darkagesmod.txt` as a mod's configuration file. Including a config in your mod zip is optional, but highly recommended!

Here is an example config. file. If you've created mods for DOOM Eternal, most of this will be familiar.

```
modinfo = {
    name = "Your Awesome Mod"
    author = "Your Name Here"
    description = "Your Mod Description"
    version = "3.0"
    loadPriority = 1
    requiredVersion = 1
```

```
}
aliasing = {
    "my_first_decl.decl" = "decls/ability_dash/my_ability_dash_decl.decl"
    "my_second_decl.decl" = "decls/ability_dash/default.decl"
}
```

Property Name	Description
name	Your mod's name. Displayed in the Mod Manager
author	Author information. Displayed in the Mod Manager
description	Your mod's description. Displayed in the Mod Manager
version	Your mod's version. Displayed in the Mod Manager
loadPriority	If multiple mods edit the same file, their load priority is used to help resolve conflicts. Mods with a lower load priority override mods with a higher load priority
requiredVersion	The version of *Atlan Mod Loader* you must be running to load this mod. • Decl mods require a minimum version of 1 • Entitydef and Logic Decl mods require a minimum version of 2 • Mapentities mods require a minimum version of 4 • Sound Mods require a minimum version of 5 • Texture Mods require a minimum version of 6 • Font Mods require a minimum version of 7

The `aliasing` block is **optional**. Are the names of your mod files extremely long and convoluted? Name them something simpler, then use your config. file to declare what's it name should be when mods are loaded!

The format for an alias is `"Real Path In Zip File" = "Game Asset Path"`

Example Mod

I've created a sample mod that illustrates the "Fake Directory" and "Aliasing" concepts. Download it **here** to get a feel for these powerful features!

Unzipped Mod Folders

When developing your mods, you don't need to zip and package them each time you want to test them! To promote fast iteration, Atlan Mod Loader will load unzipped mod files according to the following rules:

1. Each folder inside your mods folder, is considered it's own unzipped mod
2. If a folder's name begins with a `$` then everything in that folder is ignored.

For example, if your mods folder has the following contents inside it:

```
Steam/steamapps/common/DOOMTheDarkAges/
   - mods/
      - MyTextureMod/
            - darkagesmod.txt
            - MyImage.png
      - CombatBalance/
            - DeclA.decl
            - DeclB.decl
            - darkagesmod.txt
      - $RandomStuff/
            - Stuff.decl
      - CoolMod.zip
      - AwesomeMod.zip
```

When Atlan Mod Loader is ran, it will load two zipped mods, and two unzipped mods. Since the unzipped mod folder `$RandomStuff` begins with a `$` it will be ignored!

Other Benefits

Unzipped Mods have the following benefits:

1. Unserialized Mapentities, logic decls, and entitydefs will be serialized
2. Raw PNG files will be encoded into the correct format. The encoding uses a lower compression level than Atlan Mod Packager, so this process will happen much quicker!

Zipped mods do *not* receive these benefits. Atlan Mod Loader expects zipped image files to be properly packaged, and serializable files to be serialized! Remember to use Atlan Mod Packager before distributing mods containing these files!

Serialization and Atlan Mod Packager

Overview of serialization and Atlan Mod Packager

Overview

Doom Eternal's entitydefs, logic decls, and map entities files are stored natively in plaintext. What you see when editing them, is how they actually look when stored in the game files. At runtime, the game parses these text files and converts them into data.

Doom The Dark Ages changes everything. All of these files are pre-serialized into a binary format. Modding them now requires:

1. Deserializing them into a human-readable format. Atlan Resource Extractor does this for you.
2. Editing the files as desired. As a modder, you'll do this step.
3. Reserializing the files before distributing your mod. Atlan Mod Packager does this for you.
4. Installing the modded files using Atlan Mod Loader.

Those interested in the technical details can check out the source code for all these tools **here**!

Atlan Mod Packager

Download Atlan Mod Packager here!

Please read everything on the download page carefully! It tells you how to use Atlan Mod Packager, and how to optimize your workflow when developing your mods!

Serialized Files vs Regular Decls

This section highlights differences between regular Dark Ages decls and Atlan-Serialized files. You must keep all of these in consideration when editing these files.

Colors

Serialized files use *Linear RGB* for color values. Example:

```
127.5 / 255 = 0.5
my_idColor = {
	r = 0.5;
```

```
	g = 0.5;
	b = 0.5;
	a = 0.5;
}
```

File Paths

With normal decls, a reference to an entityDef file might look like this:

```
entity = "projectile_ent/my_projectile_entity"
```

Inside serialized files, this same filepath will look like:

```
entity = "entityDef/projectile_ent/my_projectile_entity"
```

The string before the first `/` is the file type. *To correctly reserialize the file, you MUST ensure the file type is there, and capitalized accurately.* See the end of this page for a list of correctly-capitalized file types.

Additional Example: Let's say we have an `idDeclAbility_Dash*` variable named `dashDecl`:

```
Full Asset Path: "rs_streamfile/generated/decls/ability_dash/modded/my_dash.decl

Regular Decl Files: dashDecl = "modded/my_dash";

Atlan-Serialized Files: dashDecl = "ability_Dash/modded/my_dash"
```

Type Info Object Pointers

If you've spent extensive time editing decl files, you'll have noticed variables with the following format:

```
some_variable = {
	className = "Class_Name_String";
	object = {
		<Object Properties>
		}
}
```

These are `idTypeInfoObjectPtr` variables. They are polymorphic object references whose exact class is determined by the `className` variable.

In Atlan-Serialized files, the syntax of these variables is simplified to:

```
some_variable "Class_Name_String" {
	<Object Properties>
}
```

The goal of this change is to simplify the syntax of these variables and remove a layer of indentation.

File Type Strings

List of all known, correctly capitalized file types. Others exist, but aren't encountered in the vanilla files and therefore aren't included here. If a type you need isn't listed here, you can likely guess the correct capitalization. They typically follow standard camel-casing rules. Obtaining a complete list of *every possible type* would require dumping data from the game executable.

```
ability_Dash
ability_ParrySwarm
ability_ShieldThrow
actorModifier
actorPopulation
advancedScreenViewShake
aiBehavior
aiBehaviorEvents
aiComponentList
aiDamageDeclCollection
aiDazeSettings
aiEnforcerAbilities
aiEvent
aiGlobalSettings
aiPoolNumbers
aiPositioningParms
aiUpgrades
aimAssist
ammo
anim
animCameraLensEffects
animWeb
attackgraph
audioLogStory
automap
automapProperties
collectibles
```

```
collisionShape
colorLUT
combatEncounterScoring
damage
damagemarkertype
destructible
devInvLoadout
dialog
ecozoneconfig
entityDamage
entityDef
env
equipmentLauncher
executions
explosion
extraLife
flare
footstepEvents
formationAttackParams
formationPositioningParams
fx
fxConditionLogicEvents
gameItem
geomcacheoptions
globalDataComponent
gorecontainer
gorewounds
gridSpaceData
handsBobCycle
highlights
hud
image
impactEffect
importance
interaction
inventoryItem
inventoryconversion
layer
lightrig
loadScreenBuckets
```

```
logicClass
logicEntity
logicFX
logicLibrary
logicObjectDescriptor
logicProfile
logicUIWidget
lootDrop
lootDropComponent
mapInfo
mapbrushes
material2
materialInteraction
model
modelSkinned
modelstream
notification
notification2
opacitymicromapdata
particle
perkGroups
perks
physicsObject
playableCharacter
playerMovement
playerProps
poi
projectile
propAttribs
propChargeBoost
propCoopRotateBob
propExtraLife
propHealth
propItem
propMoveable
propStatusEffect
propUse
questCategory
questDefs
renderParm
```

```
renderProgResource
rs_emb_sfile
rs_emb_sfilem
rumble
scoringRubric
sectorState
sectorremeshmodel
skeleton
soundPack
soundevent
soundrtpc
soundstate
soundswitch
statusEffect
strandsHaircg
string
swf
syncInteractions
table
targeting
throwable
transportControllerInfo
transportLayoutInfo
tutorialEvent
twitchPain
uiWidget
uiwalkthroughmenuargentcell
uiwalkthroughmenudossier
uiwalkthroughmenumodbot
vegetation
viewInfo_viewTuning
violenceEvent
visorSplatterEffect
waterThresholds
watergridconfig
weapon
weaponClass
windsource
```

Dark Ages Level Modding

Eternal vs Dark Ages

This page covers the critical differences between Eternal and Dark Ages entities files.

It is assumed you have some experience with editing Doom Eternal entities files. While important differences do exist between the files, there are substantial similarities to Eternal. Check out the book on **Eternal Level Modding** if this subject is new to you. Most of the knowledge you'll gain there is applicable to Dark Ages.

Serialization

In Doom Eternal, entities files are plaintext and serialized by the game at runtime. In Dark Ages, they are already serialized. See **Serialization and Atlan Mod Packager** for the rules and restrictions they follow, compared to regular decls.

Submaps

The first difference you'll notice in a Dark Ages entities file, is that every entity has a number assigned to it. This value is it's submap index.

```
entity 3 {
<entity data>
}
entity 4 {
<entity data>
}
entity 4 {
<entity data>
}
```

Doom Eternal entity files were a single, long, uninterrupted list of entities. In Dark Ages, entities are split into multiple lists based on the submap they're located in. Each list starts with it's own `idWorldSpawn` entity named `world`.

idTech8 implements a "World Composition" system that divides the level's world into submaps, loaded and unloaded at will. Think about idTech7's **refmaps** - except they're no longer just a tool for developer workflow.

Due to this World Composition system, Dark Ages does not load every entity on map load. Only entities from the active submaps are loaded. The active submaps depend on your location in the level.

Identifying Submaps

Now that we know what submaps are, how do we figure out what region of a level they belong to? Many mapentities contain a couple dozen submaps, so it's important to figure this out.

There are a few techniques to help you figure things out.

1. Compare what entities are close by to a given location in the world. (Use EntitySlayer's spawnPosition filter!)
2. Check the names of the other entities in the submaps
3. Check the `entityPrefix` variable in the submap's `world` entity.

The following world entities come from `m4_siege.mapentities`

```
// Prefix: cin_locations_atlan_cockpit
// Conclusion: This submap is for the cinematic at the end of Siege Part 2
// (Many submaps exist solely for cinematics. This makes sense, since many cinematics
// load entirely different worlds you don't see during gameplay.)
entity 1 {
	entityDef world {
		inherit = "entityDef/worldspawn";
		expandInheritance = false;
		systemVars = {
			entityType = "idWorldspawn";
		}
		edit = {
			entityPrefix = "cin_locations_atlan_cockpit";
			levelSoundState = "soundstate/level_mix/none";
			globalAIsettings = "default";
			automapDecl = "automap/default";
		}
	}
}

// Prefix: s4_battlegrounds_north_cave
// Conclusion: This submap is for one of the 2 detached cave segments in Siege Part 1
// Further analyze the submap's other entities to determine which cave it's for.
entity 14 {
```

```
	entityDef world {
		inherit = "entityDef/worldspawn";
		expandInheritance = false;
		systemVars = {
			entityType = "idWorldspawn";
		}
		edit = {
			entityPrefix = "s4_battlegrounds_north_cave";
			levelSoundState = "soundstate/level_mix/none";
			globalAIsettings = "default";
			automapDecl = "automap/default";
		}
	}
}
```

Adding New Entities

When adding new entities to a file, you must assign them a submap index based on what part of the level they'll be appearing in. Use the above techniques to identify the submaps relevant to the part of the level you want to edit. Then, give your modded entities the appropriate submap index.

Only use a submap index that already exists in the vanilla file.

Make sure a submap's `world` entity stays listed before any of it's other entities

Submap 0

Entities in most submaps will be loaded/unloaded based on where you are in the level. But Submap 0 functions as the "persistent level". Entities placed in submap 0 will always be loaded and active.

Multi-Level Entities

As a consequence of this enhanced loading system, levels are a lot bigger. So big, that sometimes two different levels' entities are contained in the same file! For example:

- *Siege Part 1* and *Siege Part 2* are both part of `mapentities/maps/game/sp/m4_siege/m4_siege.mapentities`.
- *Abyssal Forest* and *Ancestral Forge* are both part of `mapentities/maps/game/sp/m5_forge/m5_forge.mapentities`

Layers

DOOM Eternal made constant use of the layer's system. It still exists in Dark Ages, but only for specific purposes.

If an entity has a layer, it will look something like this:

```
entity 14 {
	layerIndex = 1;
	layers = {
		'spawn_target_layer"
	}
    entityDef some_entity { ... }
}
```

Dark Ages entities can only be added to ONE layer. The `layerIndex` property is important. You may think of it as the "ID" for a layer. It must be included when specifying a layer. **The same layer may have a different layerIndex value across different submaps.** The consequences of using a layer whose layerIndex is not already defined in the submap, is unknown.

End-of-File Data

If you scroll down to the bottom of a mapentities file, you'll see a massive blob of encoded data:

```
headerchunk {
	<massive amount of alphabetical data>
}
```

This is the header chunk of the original file. It is there intentionally, and **you must not touch it**. This data is critical for correctly serializing the file.

Essential Tools

Tools you should use when making Dark Ages level mods.

EntitySlayer

An editor for Doom Eternal and Dark Ages mapentities files. **Download EntitySlayer here.**

EntitySlayer has a number of useful tools that streamline the level modding process. Recent updates have added a diff-checking system that automates updating your level mods after a game update!

Check out EntitySlayer's **README** for more information.

Kaibz Mod

Kaibz Mod is the Dark Ages equivalent to Doom Eternal's Meathook mod. It adds a number of console commands useful to level modding. **Download Kaibz Mod here.**

Command	Description
k_spawninfo	Copies the player's spawnPosition and spawnOrientation to your clipboard. This command is equivalent to `mh_spawninfo` from Doom Eternal's Meathook mod. Currently, this command only works when playing as the Slayer.
k_activeEncounters	Prints a list of active idEncounterManagers to the console. This is extremely helpful for figuring out what encounter managers are orchestrating the current combat encounter.
k_noclip	Toggles noclip
k_hotReload	See **Hot Reloading** for instructions on the proper usage of this command.
setviewpos <x> <y> <z>	Teleports you to the desired location. The vanilla `teleportposition` command does not work in Dark Ages. Use this instead. (This is a vanilla command, but it's included here for reference)

Connecting EntitySlayer to Kaibz Mod

Doom Eternal's Meathook mod possesses an RPC interface. This lets other processes communicate with the game to execute console commands. EntitySlayer uses this interface to enable several convenient editor features, like setting an entity's spawnPosition from the player's current position.

Kaibz Mod offers equivalent functionality using a Windows Pipe interface. To enable this feature:

1. Launch the game, and open the Kaibz Mod menu by pressing the `F8` Key.
2. Open the `MISC.` menu. In that menu, set `Enable Mod Interface` to `YES`
3. You're good to go! Kaibz Mod saves your settings, so you don't need to turn the interface on every time you launch the game.

> Kaibz Mod does not support every feature offered by Meathook. A small number of options in EntitySlayer still won't work because of this.

Hot Reloading

How to use Atlan Mod Loader and Kaibz Mod to reload modded entities files without restarting the game.

> This is an experimental feature. It may or may not be broken on Linux. If you attempt this procedure, please report whether it's successful or not.

The biggest problem with editing level mods is iteration time. Each time you want to test new changes, you must re-install the mod and restart the game. It's a time-consuming process that grinds down development. But with the power of Kaibz Mod and Atlan Mod Loader, this inconvenience can be eliminated!

Install Kaibz Mod

You must install **Kaibz Mod** to perform hot reloading! It is recommended you enable it's Mod Interface while developing level mods. See **Connecting EntitySlayer to Kaibz Mod** to learn how to enable the mod interface.

Before Launching the Game

Atlan Mod Loader will engage "Hot Reload Mode" under the following conditions:

1. An **unzipped and unpackaged** .mapentities file exists in your mods folder.
2. No other mod files exist in your mods folder.

Let's say you want to edit the mission *Hebeth*. Your mods folder should look like this. **Do not add any files besides the mapentities.**

```
- Steam/steamapps/common/DOOMTheDarkAges/
   - mods/
      - hot_reload/
         - mapentities@maps@game@sp@m2_hebeth@m2_hebeth.mapentities
```

Once this is setup, run Atlan Mod Loader and launch the game.

Reloading the File

Let's say you've made some changes to the Hebeth mapentities file and you're ready to reload it. Run the console command `k_hotReload` to reload the map!

Notes:

1. This may alt-tab you out of the game for several seconds. The alt-tabbing is most severe when running the game in `Full Screen` mode. If it gets annoying, try changing to `Borderless Windowed` mode.
2. You may notice a command prompt briefly pop up and close. This is Atlan Mod Loader running due to Kaibz Mod.

Quick Test

The first time you attempt hot reloading, you should do something simple to verify it works on your system. The following code snippet will place a floating text entity at the start of Hebeth.

1. Paste this into Hebeth's mapentities file. **Make sure you paste it somewhere AFTER submap 0's world entity.**
2. Edit the entity's `edit/headerText/text` property.
3. Trigger a hot reloading using the above procedure.
4. Verify that the text you see in-game has changed to reflect your edits.

```
entity 0 {
	entityDef hot_reload_test {
		inherit = "entityDef/gui/text";
		expandInheritance = false;
		editorVars = {
			placeable = false;
		}
		systemVars = {
			entityType = "idGuiEntity_Text";
		}
		edit = {
			headerText = {
				text = "Hot Reload Test!";
			}
			canvasFile = "ui/shapes/worldgui_text";
			useSWFTransform = true;
```

```
	swfScale = 0.016667;
	spawnPosition = {
		x = -1245.702881;
		y = -1230.202271;
		z = -77.942009;
	}
	spawnOrientation = {
		mat = {
			mat[0] = {
				x = -0.000000;
				y = -1.000000;
			}
			mat[1] = {
				x = 1.000000;
				y = -0.000000;
			}
		}
	}
	flags = {
		noknockback = false;
	}
	renderModelInfo = {
		model = "editors/models/gui_text.lwo";
		scale = {
			y = 15.000000;
		}
	}
	clipModelInfo = {
		type = "CLIPMODEL_NONE";
	}
  }
}
```

Reloading the File (Without Kaibz Mod)

> Hot Reloading without Kaibz Mod is NOT RECOMMENDED! This section is here purely for documentation purposes! Please use the above method instead of this one!

To Hot Reload without using Kaibz Mod, perform the following steps:

1. Alt-Tab out of the game.
2. Run Atlan Mod Loader and wait for it to complete
3. Quit out to main menu, then load back into Hebeth using Mission Select.
4. The changes you've made to the file should be reflected in-game.

MapExecutions Decls

Location: `rs_streamfile/generated/decls/mapexecution/`

These decls are new to idTech8, and define object pools for hundreds of different entity types. It's unknown how important they are compared to the `aipoolnumbers` decls. If you ever run into a problem with entities not spawning or the game crashing when too many of an entity are spawned, consider experimenting with these files.

Dark Ages Audio Mods

Getting Started

Make sure you've run **Atlan Resource Extractor** to extract audio files from the categories you wish to edit.

Converting Your Audio Tracks

Dark Ages uses WEM OPUS encoding for all SFX and Music tracks. To convert your custom audio samples into this format, you must install and use WWise.

Follow **this guide** to learn how to do this, with the following adjustments

- When editing the Default Conversion Settings (step 2 of the linked guide): select "WEM Opus" as the format. Do **NOT** select "Vorbis".
- No other changes to the default conversion settings appear to be necessary. However, feel free to experiment and see if anything effects the quality of your audio when in-game.

Naming Rules

The rules for naming your audio files are much different than the rules for naming other types of mod files. You'll notice in the extracted audio files, each sample has a number at the end of it's filename. For example:

```
audio/MUSIC/MUS_Combat_UnchainedPredator_Heavy_2_Main_B_105bpm_MIX_633792027.wav
```

Take note of the number `633792027` at the end of the name. This is the sample ID. Your modded audio files can be named anything, so long as:

1. The path begins with `audio/`
2. The name ends with the ID of an **existing** audio sample. (In theory, you *can* create entirely original samples. But then you'll need to edit the Sound Banks binaries to make the game use them. This is very complicated, and likely not worth the effort)

Anything in-between `audio/` and the sample ID does not matter. The file extension does not matter.

The following are all examples of valid modded audio sample paths. All of them modify the same sample. It is recommended that you give your custom samples an easily identifiable name, that clearly illustrates what sample is being replaced.

```
audio@633792027 .wem
audio@My_Cool_Sample_633792027.wem
audio/music/unchained_heavy_2_main_b_633792027.wem
```

Dark Ages Texture Mods

Getting Started

Creating texture mods requires the following resources:

1. **Atlan Resource Extractor** - Use this to extract material2 decls from the game. These plaintext files define the game's materials and directly reference image assets.

2. **Valen** - A resource extractor that can extract vanilla image files from the game. Atlan Resource Extractor does *NOT* support extracting textures. Use this instead.

3. **Atlan Mod Packager** - Packages your custom texture PNG files into a format the game recognizes. This is a necessary step to reduce mod loading times, as converting the images is an expensive process. You must use Atlan Mod Packager to prepare your finished texture mods for distribution

4. **Mod Config File** - It is *highly recommended* you use the aliasing system provided by your mod's configuration file! Image files have long names that don't clearly describe what piece of a mesh they're for! The aliasing system is helpful for organization, and prevents you from needing to change the extension on your PNGs to match the vanilla filename!

Supplemental resources may also exist for common areas of texture modding:

1. Custom Slayer Skins: Check out Kuddly Kraken's **Creation Kit**

Assembling Your Mod Zip

All image mods must have `image/` at the beginning of their filepath (or configuration alias). After this, simply recreate the path of the image file you want to replace. **This includes the extension properties.** For Example:

```
image/models/customization/characters/doomslayer/set_b/1001/doomslayer_set_b_torso_1001.tga$s
treamed$mtlkind=albedo
image/models/customization/characters/doomslayer/set_b/1001/doomslayer_set_b_torso_1001.tga
```

Path #1 refers to a vanilla image file. However, Path #2 does not exist and the game will not recognize your image. **Extension metadata is important! It must be recreated perfectly when you want to replace an existing image.**

As with other resource types, an `@` or a `\` can be used in place of a `/` character.

Aliasing

As you can see, these image names grow long and complicated very quickly. The aliasing system can alleviate this problem. Lets say we have an image called `helmet_doodles.png` and we want it to replace the above image. Using our config aliasing system:

```
aliasing = {
    "helmet_doodles.png" =
"image/models/customization/characters/doomslayer/set_b/1001/doomslayer_set_b_torso_1001.tga$
streamed$mtlkind=albedo"
}
```

This is equivalent to renaming your PNG to that filepath.

Developing and Packaging

Rinse and repeat this process until you've built out your entire texture mod folder. When you're ready to ship, run AtlanModPackager to package it into a zip file ready for loading your mods!

While developing and testing your mod, remember to take advantage of Atlan Mod Loader's **unzipped mod loading capabilities.**

Sample Mod

The following link takes you to a sample mod implementing the above example! It uses the aliasing system to replace the Nightmare Slayer skin's helmet texture! To test this mod:

1. Download and unzip it's contents
2. Run AtlanModPackager on the unzipped folder
3. Use Atlan Mod Loader to load the newly created mod!

Google Drive Link

Adding New Images

> Adding original images to the game is more complicated than replacing existing textures. Please pay careful attention to all of the information in this section. If all you're doing is replacing existing textures, you can skip this section

If you want to add an original image to the game, instead of replacing an existing image, you'll need to follow several more steps:

1. Reference your new images inside one (or multiple) material2 decls.
2. Determine the correct extension properties for your image
3. Specify your image's encoding information. This step is not always necessary depending on the image.

Material2 Decls

Once you've loaded your new image assets into the game, you must reference them in one or more material2 decls! You can edit a vanilla decl to use your texture, or create an entirely new material2 to reference in other files. As the mod author, you make the design decisions!

Extension Properties

When deciding the asset name of your new image, you can choose any unique name:

```
image/my_awesome_image.tga
image@your_username_here@your_modname_here@image_1.tga
image/cool_skin/helmet/helmet_albedo.png
```

However, the extension properties **must be written correctly.** If you fail to write them accurately, the game will not be able to find and load your modded image!

The exact type varies by the type of image you're adding, and by the inherited template of the material2 decl you're using it in. For example, if you're adding an albedo image, your extension properties should be `$streamed$mtlkind=albedo`

There is no exhaustive list of correct extension properties. The best way to be certain you're correct, is by studying the names of vanilla images produced by that particular material2 template. More obscure templates may require some trial and error to get the extension properties correct.

Encoding Information

When adding a new image, Atlan requires two key pieces of information: the image format (i.e. BC7, BC1, etc.) and the material type (i.e. albedo, normal, specular, etc.)

For *most* cases, you probably won't need to worry about this. Atlan will analyze the extension properties discussed above, and correctly deduce these settings. However, there are cases where the extension properties are insufficient for multiple reasons:

1. Ambiguity: Depending on the material template, the format cannot be determined with absolute certainty! For example an image with properties `$streamed$mtlkind=blendmask` can use different formats depending on the material type!
2. Absence: Some material templates omit extension properties from their images entirely!

3. In very rare cases, the extension properties may flat-out lie about the type or format used.

In these circumstances, you must specify the correct type and format by adding them to the end of the image name (or alias name, if it has an alias). For Example:

```
aliasing = {
    "my_blendmask.png" =
"image/my_blendmask.tga$streamed$mtlkind=blendmask~FMT_BC7~TMK_BLENDMASK"
}
```

As you can see, the format is `~FORMAT_CODE~MATERIAL_TYPE_CODE`

The following code block lists all possible values for these fields. Determining the correct values for a given image may take some careful analysis, depending on the material template.

```
enum textureMaterialKind_t {
    TMK_NONE            = 0,
    TMK_ALBEDO          = 1,
    TMK_SPECULAR        = 2,
    TMK_NORMAL          = 3,
    TMK_SMOOTHNESS      = 4,
    TMK_COVER           = 5,
    TMK_SSSMASK         = 6,
    TMK_COLORMASK       = 7,
    TMK_BLOOMMASK       = 8,
    TMK_HEIGHTMAP       = 9,
    TMK_DECALALBEDO     = 10,
    TMK_DECALNORMAL     = 11,
    TMK_DECALSPECULAR   = 12,
    TMK_LIGHTPROJECT    = 13,
    TMK_PARTICLE        = 14,
    TMK_DECALHEIGHTMAP  = 15,
    TMK_AO              = 16,
    TMK_UNUSED_3        = 17,
    TMK_UI              = 18,
    TMK_FONT            = 19,
    TMK_LEGACY_FLASH_UI = 20,
    TMK_UNUSED_4        = 21,
    TMK_BLENDMASK       = 22,
    TMK_PAINTEDDATAGRID = 23,
    TMK_COUNT           = 24,
```

```
};

enum textureFormat_t {
    FMT_NONE            = 0,
    FMT_RGBA32F         = 1,
    FMT_RGBA16F         = 2,
    FMT_RGBA8           = 3,
    FMT_RGBA8_SRGB      = 32,
    FMT_ARGB8           = 4,
    FMT_ALPHA           = 5,
    FMT_L8A8_DEPRECATED = 6,
    FMT_RG8             = 7,
    FMT_LUM8_DEPRECATED = 8,
    FMT_INT8_DEPRECATED = 9,
    FMT_BC1             = 10,
    FMT_BC1_SRGB        = 33,
    FMT_BC1_ZERO_ALPHA  = 54,
    FMT_BC3             = 11,
    FMT_BC3_SRGB        = 34,
    FMT_BC4             = 24,
    FMT_BC5             = 25,
    FMT_BC6H_UF16       = 22,
    FMT_BC6H_SF16       = 36,
    FMT_BC7             = 23,
    FMT_BC7_SRGB        = 35,
    FMT_DEPTH           = 12,
    FMT_DEPTH_STENCIL   = 13,
    FMT_DEPTH16         = 31,
    FMT_X32F            = 14,
    FMT_Y16F_X16F       = 15,
    FMT_X16             = 16,
    FMT_Y16_X16         = 17,
    FMT_RGB565          = 18,
    FMT_R8              = 19,
    FMT_R11FG11FB10F    = 20,
    FMT_R9G9B9E5        = 57,
    FMT_X16F            = 21,
    FMT_SMALLF          = 60,
    FMT_MAINVIEW_SMALLF = 61,
    FMT_RG16F           = 26,
```

```
    FMT_R10G10B10A2     = 27,
    FMT_RG32F           = 28,
    FMT_R32_UINT        = 29,
    FMT_RG32_UINT       = 58,
    FMT_R16_UINT        = 30,
    FMT_R8_UINT         = 55,
    FMT_ASTC_4X4        = 37,
    FMT_ASTC_4X4_SRGB   = 38,
    FMT_ASTC_5X4        = 39,
    FMT_ASTC_5X4_SRGB   = 40,
    FMT_ASTC_5X5        = 41,
    FMT_ASTC_5X5_SRGB   = 42,
    FMT_ASTC_6X5        = 43,
    FMT_ASTC_6X5_SRGB   = 44,
    FMT_ASTC_6X6        = 45,
    FMT_ASTC_6X6_SRGB   = 46,
    FMT_ASTC_8X5        = 47,
    FMT_ASTC_8X5_SRGB   = 48,
    FMT_ASTC_8X6        = 49,
    FMT_ASTC_8X6_SRGB   = 50,
    FMT_ASTC_8X8        = 51,
    FMT_ASTC_8X8_SRGB   = 52,
    FMT_DEPTH32F        = 53,
    FMT_RGBA16_UINT     = 56,
    FMT_RG16_UINT       = 62,
    FMT_RGBA16          = 59,
    FMT_NEXTAVAILABLE   = 63,
};
```

Dark Ages Font Mods

Overview

DOOM: The Dark Ages uses Slug Fonts to render in-game text.

To get a list of all slug font files, use **Atlan Resource Extractor** to extract resources of type `slug_font`

To load your custom slug font:

1. Get your desired font file (a .ttf or a .otf)
2. Download the Slug Font Demo at **https://sluglibrary.com/** and use it to convert your font file to a .slug
3. Rename your .slug to the name of the font resource you want to replace. You can leave the file extension as .slug, as the mod loader will ignore it.

For English Characters:

1. UI text uses `slug_font/cantoria_regular` and `slug_font/cantoria_semibold`.
2. Subtitles use `slug_font/calibri`

Many of the custom HUDs introduced in Dark Ages also have their own dedicated font files.