Atlan Mod Loader

General information related to Atlan Mod Loader

- **Atlan Fundamentals**
  - **Atlan Mod Loader vs Eternal Mod Injector**
  - **Adding Files to your Mods**
  - **Atlan Config File**
  - **Unzipped Mod Folders**
- **Atlan Patch Notes**
  - **Atlan Mod Loader Patch Notes**
  - **Atlan Mod Packager Patch Notes**
  - **Atlan Resource Extractor Patch Notes**
- **Atlan Texture Modding**
- **Atlan Audio Modding**
- **Atlan Assets Info**

Atlan Fundamentals

The basics of using Atlan Mod Loader to make mods

Atlan Mod Loader vs Eternal Mod Injector

Atlan's approach to mod-loading is fundamentally different from Eternal Mod Injector's. The major limitations of Eternal Mod Injector do not apply to Atlan.

Eternal Mod Injector	Atlan Mod Loader
Replaces files inside of the vanilla game archives	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 loaded.
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.

Atlan does not 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 as a baseline. The dangers of outdated mod files vary based on their importance. Consequences can range from nothing happening, to core mechanics breaking, to your game crashing on startup.

What files are modified?

Atlan Mod Loader edits the following files:

```
The game executable file (i.e. DOOMTheDarkAges.exe)
base/packagemapspec.json
base/meta.resources
base/sound/soundbanks/pc/soundmetadata.bin
base/build-manifest.bin (DOOM: Eternal only)
```

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`

Adding Files to your Mods

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 extracted 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!

Example Mod

You can download an **example mod** that illustrates the "Fake Directory" concept. It also includes an example of the **mod config's** aliasing system. All files in this example mod replace the same game asset.

Atlan Config File

Atlan Mod Loader uses `darkagesmod.txt` as the configuration file for DOOM: The Dark Ages mods. If you're making a mod for DOOM: Eternal, you should use `atlanmod.txt` as your file name. Including a config in your mod zip is optional, but highly recommended!

The config file consists of several optional properties.

ModInfo Chunk

The ModInfo properties are the only ones that should be included in *every* config file. Here is an example:

```
modinfo = {
    name = "Your Awesome Mod"
    game = "Doom The Dark Ages"
    author = "Your Name Here"
    description = "Your Mod Description"
    version = "3.0"
    loadPriority = 1
    requiredVersion = 1
}
```

Property Name	Description
name	Your mod's name. Displayed in the Mod Manager
game	What game this mod is made for. Atlan will not install *DOOM: Eternal* mods in *DOOM: The Dark Ages* and vice versa
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. • Dark Ages Decl mods require a minimum version of 1 • Dark Ages Entitydef and Logic Decl mods require a minimum version of 2 • Dark Ages Mapentities mods require a minimum version of 4 • Dark Ages Sound Mods require a minimum version of 5 • Dark Ages Texture Mods require a minimum version of 6 • Dark Ages Font Mods require a minimum version of 7 • Dark Ages Mods utilizing assets-info require a minimum version of 9 • All DOOM: Eternal mods require a minimum version of 10

Aliasing Chunk

The aliasing block is optional. It is meant to help with mod files whose paths are extremely long and convoluted. You may choose to name the files something simple, then alias them to their real asset paths.

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

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

Assets, MapSpec, and MapResources Chunks

These chunks are used for assets info manipulation. These are advanced properties that require some experience to use correctly. Their usage is explained in-depth on **this page**

Load Level Chunk

This chunk enables idStudio level mods that were installed by Atlan. You can use it to load any number of level mods.

```
// This example causes two idStudio level mods to be loaded
// Their respective installation paths would be:
// 1. base/modarchives/coop_hordamex/
// 2. base/modarchives/pvp_hurricane/

load_level {
	coop_hordamex
    pvp_hurricane
}
```

Atlan Mod Packager requires this property to be set when packaging an idStudio level mod.

Total Overhaul Property

This property should only be used under very rare circumstances. Does your mod touch *hundreds* of files or edit dozens of levels? Does this make it fundamentally incompatible with other mods of similar size? You may wish to designate it as a "total overhaul" mod. Atlan will only allow one Total Overhaul mod to be loaded at a time.

```
modinfo {
	total_overhaul
}
```

Elevated Gameplay Property

This property only exists for DOOM: Eternal multiplayer gamemode mods. It shouldn't be used anywhere else. When set, only your mod can load gameplay-affecting files. Any gameplay-affecting files in other mods will fail to load. Only one Elevated Gameplay mod can be loaded at a time. The intent of this feature is to prevent players from accidentally cheating in competitive modded multiplayer by unknowingly loading other gameplay-affecting mods.

```
modinfo = {
	elevated_gameplay
}
```

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 an unzipped mod
2. If a folder's name begins with a `$` then it 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. Dark Ages: Unserialized Mapentities, logic decls, and entitydefs will be serialized
2. Eternal: Uncompressed entities files will be compressed.
3. Both Games: Raw PNG files will be encoded into image assets. The encoding uses a lower compression level than Atlan Mod Packager, so this process will happen quicker!

Zipped mods do not receive these benefits. Atlan Mod Loader expects zipped mod files to be serialized, compressed, or encoded in advance! Remember to use Atlan Mod Packager before

distributing mods containing these files!

Atlan Patch Notes

Patch notes for Atlan tools

Atlan Mod Loader Patch Notes

Patch Notes for Atlan Mod Loader

Version 9

- Added support for PackageMapSpec and MapResources manipulation.
  - See the **this page** for more information.
- Hot Reloading can now be performed while other mod files are installed.
  - See the **this page** for more information.

Version 8

- Replaces DarkAgesPatcher with a built-in executable patcher
  - You can safely delete DarkAgesPatcher and it's other files from your game folder
  - Replacing the executable patcher is a necessary step to eventually bringing Atlan Mod Loader support to DOOM Eternal.

Version 7

- Add support for loading custom Slug Fonts!
- See **this page** to learn how to create font mods!

Version 6.1

- Updated for Revelations
- Improve clarity of some error messages
- Includes the latest update and patch definitions for DarkAgesPatcher
- 6.1 Hotfix: Added executable patcher definitions for Revelations Hotfix 1
- 6.1.1: Prevent crash caused by encoding multiple images on the GPU simultaneously

Version 6

- Added support for loading texture mods!
  - See **this page** for more information
- Refactored unzipped mod loading to improve flexibility and convenience when testing multiple mods.
  - See

this page for more information
- Optimize unzipped mod loading
- Fix an issue where the executable patcher would fail to re-run after verifying game files due to a stale cache file, resulting in startup crashes when mods were installed.

Version 5

- Added support for loading sound mods!

- See **this page** to learn how to create sound mods.
- Version 5.0.1: Add checksum validation when downloading Oodle

Version 4

- Added experimental "Hot Reload Mode". See **this page** for more information
  - Version 4.0.1: Fix potential integer overflow involving Hot Reload Mode and very large entities files.
- Added the `--notimer` argument to skip the 10 second exit timer.
- Added the DarkAgesPatcher definition file for game Update 3
- Updated the reserializer's reflection code for Game Update 3

Version 3.1

- Added the DarkAgesPatcher definition file for game Update 2.2
- Added better support for users manually updating DarkAgesPatcher's definition file
- Always replace backups if the existing files are vanilla, instead of only when the game is updated
- The backup restoration system now executes even when DarkAgesPatcher fails.
- Disable resource murmur hashing since DarkAgesPatcher disables those hash validations.
- Updated reserializer's reflection code for Game Update 2.2

Version 3.0

* Added support for mapentities files
* Updated DarkAgesModManager to v1.1

Version 2.0

* Added support for entityDef mods
* Added tentative support for Logic Decl mods.

Atlan Mod Packager Patch Notes

Patch notes for Atlan Mod Packager

Version 2.3

* Image Encoder: Small performance and memory usage improvements
* Reserializer: Improve and fix some bugs with error logging

Version 2.2:

* Add support for packaging Slug Fonts.
* Add command line arguments

Version 2.1:

* Updated for Revelations
* Improve clarity of image encoding error messages
* v2.1.1: Prevent crash caused by encoding multiple images on the GPU simultaneously.

Version 2.0:

* Added the texture mod packager
* Added dialogues for selecting a folder to package and for choosing the output location of your zip file
* All invalid files are now excluded from the final mod zip. The full list of ignored files is logged.

Version 1.2:

* Log some more error messages to the console.
* Version 1.2.1: Add checksum validation when downloading Oodle

Version 1.1:

* Updated reserializer's reflection code for Game Update 3
* Oodle-compress packaged mapentities files. This massively decreases their zipped and unzipped file sizes.
* Stop serializing mapentities metadata.

Version 1.0.2:

* Updated the reserializer's reflection code for Game Update 2.2

Version 1.0.1:

* Add support for serializing mapentities files.

Atlan Resource Extractor Patch Notes

Patch notes for Atlan Resource Extractor

Version 4.0

Extra Scripts
* Added an "extra scripts" section controlled by the config. file
* Currently, the extra scripts will:
    * Dump a complete list of correctly-capitalized resource types
    * Decode .mapresources files into text
* v4.1: Add more extra scripts

Resource Extractor:
* Added support for decompressing compfiles (DOOM Eternal .entities files)
* Added blang-to-text conversion (DOOM Eternal language files, resource type is "binaryFile")

Audio Extractor:
* Added full support for extracting DOOM Eternal audio files.
* Eternal samples are extracted to their original sample paths/names.
* Improved multithreading setup
* Fixed some rare vgmstream file-locking problem that could cause file decoding to fail
* v4.1: Add warning when no audio files are extracted

Soundbank Extractor:
* Add support for extracting DOOM Eternal's soundbanks

Version 3.3:

* Updated for Revelations
* Include sector names in the mapentities files

Version 3.2:

* Added the AudioKinetic Soundbank extractor!
* Improved how the audio extractor categorizes SFX samples.
    * If you've previously extracted SFX, you should rename your SFX output folder before running the extractor again. Otherwise, you'll likely end up with duplicate copies of many samples.

Version 3.1:

* Decls are now extracted to `decls/` instead of `rs_streamfile/generated/decls/`
  * The extractor will automatically move the original decl folder if it exists.
  * Atlan Mod Loader recognizes this new extraction path and handles it correctly. The original path will still be recognized. Existing decl mods won't require an update because of this change.
* Output path length limit has been raised to 40 characters
* Added a decode_samples toggle to the audio extractor
* v3.1.1: Fixed a crash when running the audio extractor while sound mods are installed
* v3.1.2: Add checksum validation when downloading Oodle

Version 3.0:

* Added audio extraction
* Fixed a very rare issue where outdated versions of certain files could be extracted.
* v3.0.1: Fixed a crash when running the extractor on a modded game.

Version 2.3:

* Updated deserializer's reflection code for Game Update 3
* Mapentities header chunks are now encoded in alphabetical characters. This fixes issues where certain text editors would mangle the header chunk due to automatic line ending replacement.

Version 2.2:

* Updated deserializer's reflection code for Game Update 2.2
* Simplified extraction paths for mapentities files
* Added support for extracting logicObjectDescriptor files
* Added support for extracting files from DOOM Eternal's resource archives

Version 2.1:

* Heavily revise the deserialized mapentities file format
* Added eventcall strings.

Version 2.0.1:

* Fix a bug where the deserializer would not correctly format multi-line strings in entityDef files.

Version 2.0:

* Integrated the Entity Deserializer!

Version 1.3:

* Optimize the extraction process

Version 1.2:

* Added config file
* Added log file
* Added 16-character limit to the length of the output directory.

Version 1.1:

* Properly decompress mapentities
* Add logicEntities and binarymd6defs

Atlan Texture Modding

Documentation on creating texture mods with Atlan. This information applies to both DOOM: The Dark Ages and DOOM: Eternal. For DOOM: Eternal, it's preferable to create texture mods using idStudio. If you can't use idStudio, this guide offers an alternative approach.

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. You must use Atlan Mod Packager to prepare your finished texture mods for distribution
4. **Mod Config File** - When adding *new* images, it is *highly recommended* you use the aliasing system provided by your mod's configuration file!

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

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

Workflow Overview

This is a basic overview of how to create a texture mod:

1. Assemble your modded textures using the techniques described below. Your finalized images must be PNGs.
2. When you're ready to publish your mod, use Atlan Mod Packager to package it into a zip file! This step is *required*. Atlan Mod Loader will not convert PNGs it finds in zip files. It expects zipped images to be converted in-advance. This is necessary for having reasonable mod loading times, as converting images is an expensive process.

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

Replacing Existing Images

Replacing existing images is more convenient than adding original images to the game. This approach requires Valen v0.3.3 or higher (see above). Image replacement should be sufficient for the majority of texture mods.

First, use Valen to browse the game files for the images you want to replace. Export your images. You'll notice that the names of the exported images have the format: `name#<16-character-code>.png`

The `#<code>` is important! Atlan detects that code and uses it to determine the correct asset path for your image. This way you don't need to recreate the entire original image path, which is often very long and complicated!

In your mod's folder, your image should have the path `image/<...>#<code>.png`

The actual path or name before the `#code` does not matter. All that matters is the path start with `image/` and the name ends with `#<code>`. Remember: **do not modify the code**

```
// Some Examples: Both of these paths refer to the same image with code #1234567812345678
image/art/models/characters/doomslayer/torso/1001/doomslayer_torso_1011#1234567812345678.png
image/my_cool_slayer_mod#1234567812345678.png
```

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:

```
// Remember: The modloader converts '@' symbols into '/' symbols
image/my_awesome_image.png$streamed$mtlkind=albedo
image@your_username_here@your_modname_here@image_1.png$streamed$mtlkind=albedo
image/cool_skin/helmet/helmet_albedo.png$streamed$mtlkind=albedo
```

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 extension properties vary 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 won't need to worry about this. Atlan analyzes the extension properties discussed above to deduce these values. 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. The following snippet shows the formats with examples

```
// Version 1: ~FORMAT_CODE~MATERIAL_TYPE_CODE
"image/my_blendmask.tga$streamed$mtlkind=blendmask~FMT_BC7~TMK_BLENDMASK"

// Version 2: Uses the optional third parameter:
~FORMAT_CODE~MATERIAL_TYPE_CODE~REAL_FORMAT_CODE
```

```
"image/my_blendmask.tga$streamed$mtlkind=blendmask~FMT_BC7~TMK_BLENDMASK~FMT_BC7_SRGB"
```

As you can see, there are two formats:

1. `~FORMAT_CODE~MATERIAL_TYPE_CODE`
2. `~FORMAT_CODE~MATERIAL_TYPE_CODE~REAL_FORMAT_CODE`

Format #2 is for very rare circumstances. There are times where the game encodes an image one way, while the extension properties and image metadata both claim it's encoded in another way! If your image does not render correctly in-game and you've exhausted all other possibilities, try format #2. Otherwise, Format #1 will be enough.

Aliasing System

When adding new image files, their extension data quickly grows long and complicated. You may choose to use the mod config file's aliasing system to alleviate this problem. This way, instead of including the extension information in the filename itself, you may specify it in the config file.

```
// Example of using the aliasing system on a new image file
aliasing = {
    "image/helmet_doodles.png" =
"image/helmet_doodles.png$streamed$mtlkind=albedo~FMT_BC1_SRGB~TMK_ALBEDO"
}
```

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_X16       = 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,
};
```

Atlan Audio Modding

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
```

Atlan Assets Info

This section describes advanced settings available through your mod's configuration file. Please read **this page** to gain a basic understanding of the Atlan config. file before continuing here.

> These are advanced settings. Please take care when using them. Improper usage may cause instability or crashes.

PackageMapSpec Manipulation

Located in your game installation is the file `base/packagemapspec.json`

This file declares the resource and streamdb archives that are to be loaded in each level. By editing this file, you can change what resources a level loads. For example, if you wanted to use DLC AI in a base-campaign level, you can import the DLC level's archives to gain access to those assets.

The following is an example of a mod config. file that performs packagemapspec manipulation

```
modinfo = {
    name = "MapSpec Example"
    author = "FlavorfulGecko5"
    description = "Demonstrates how the MapSpec block works"
    version = "1"
    loadPriority = 1
    requiredVersion = 9
}
mapspec = {
    "game/ripatorium/ripatorium$game/sp/m1_intro/m1_intro$.resources"
    "game/ripatorium/ripatorium$game/sp/m7_armada/m7_armada$.streamdb"
    "game/ripatorium/ripatorium$game/sp/m11_styx/m11_styx"
    "game/dlc/m15_hub/m15_hub$game/sp/m9_cosmic_a/cosmic_a$_patch1.resources"
}
```

First, let's introduce the format: `"FROM_MAP$TO_MAP$FILTER"`

- FROM_MAP is the map you want to import files from.
- TO_MAP is the map you want to import files into.
- FILTER is a filter string for specifying the exact files you want to import. This field is optional.

- $ characters separate the three fields.
- You can find all map names in the maps list, at the bottom of the `packagemapspec.json` file.

Now let's walk through each example in the above config. file:

`game/ripatorium/ripatorium$game/sp/m1_intro/m1_intro$.resources`

1. Imports all `.resources` archives loaded by the Ripatorium into Village of Khalim.

`game/ripatorium/ripatorium$game/sp/m7_armada/m7_armada$.streamdb`

2. Imports all `.streamdb` archives loaded by the Ripatorium into Sentinel Command Station

`game/ripatorium/ripatorium$game/sp/m11_styx/m11_styx`

3. This import statement has no filter string. Therefore, ALL resource and streamdb files loaded by Ripatorium get imported into Harbor of Souls

`game/dlc/m15_hub/m15_hub$game/sp/m9_cosmic_a/cosmic_a$_patch1.resources`

4. This imports `m15_hub_patch1.resources` into the first Cosmic Realm level. It's not recommended to use this feature this way versus importing all resource archives. However, if you really want just a patch archive imported for some reason, this is one way you can do it.

MapResources Injection

You can use the Config file to inject data into the game's MapResources files. To understand what MapResources files are, and why you'd want to edit them, see **this page**.

Atlan allows you to inject data into any level's MapResources file. This is accomplished using the mapresources and assets properties. See the following example:

```
modinfo = {
    name = "MapResources"
    author = "FlavorfulGecko5"
    description = "Demonstrates how the MapResources block works"
    version = "1"
    loadPriority = 1
    requiredVersion = 9
}
mapresources = {
```

```
"common" {
    my_asset_list
    }

    "game/sp/m1_intro/m1_intro" {
    LOAD_ALL
        ListTwo
        ListThree
    }
}
assets = {

    my_asset_list {
    "aiComponentList/custom/my_custom_component_list"
    }

    ListTwo {
    "damage/my_damage_decl"
        "projectile/my_projectile_decl"
        "entityDef/modded/cool_entity"
    }

    ListThree {
    aiComponentList/custom/another_component
    }
}
```

The workflow is straightforward:

1. Use the `asset_lists` block to define lists of game assets.

In your asset lists, the text before the first `/` character is the resource type. **The capitalization of the resource type matters, and must be correct!** Atlan Resource Extractor's extra scripts will provide a full list of correctly capitalized resource types for you.

This example config. defines three asset lists named "my_asset_list", "ListTwo" and "ListThree"

2. Use the `mapresources` block to list all maps whose MapResources files you want to edit. Then, add the names of your asset lists to their blocks.

LOAD_ALL is a special indicator. If encountered, Atlan will disable the layermask of all vanilla entries to ensure every resource in the file is always loaded. For Dark Ages, this shouldn't be necessary to set. For Eternal, you may want to set this when doing extensive entities modding and encountering stability

issues. Read **this page** to understand what all of this means.

This example config does several things:

- The asset list `my_asset_list` is injected into `common.mapresources`
- The asset lists ListTwo and ListThree are injected into `m1_intro.mapresources` and LOAD_ALL is enabled on it.

Practical Usage

For the majority of mods that add new game assets, the following setup should be enough to ensure your custom assets always load correctly. Adding assets to `common.mapresources` will cause them to be globally loaded.

```
mapresources {
    "common" {
        new
        }
}
assets {
    new {
        "Your Custom Assets"
        }
}
```

Edge Cases

There are some situations where you'll need to edit level-specific mapresources. For example, let's say you add a new asset that references a DLC AI file only loaded in the DLC levels. What happens if you put this new asset into common's MapResource file?

1. Your custom asset will be loaded on game startup
2. Since your asset has inaccessible, level-specific dependencies, they won't get loaded. This leaves your asset in a semi-broken state. Since it's loaded by the common MapResources, the game will never reload or refresh this asset during level transitions.
3. When it comes time to use this game asset, it won't work properly. Your game may have even crashed before this point.

The "correct" action in this situation is to edit the level-specific MapResources where this asset will be used, instead of the common MapResources file.

Modder's Resources

Atlan Resource Extractor provides information to help you correctly edit the MapResources files. By running the *extra scripts*, you receive:

- A list of all correctly capitalized resource types
- All vanilla MapResources files decoded into plaintext, allowing you to see what assets are globally loaded, and which are level-specific.