Doom Eternal - Atlan Modding

Guides for modding Doom Eternal using Atlan Mod Loader

- **Making Mods: Getting Started (DOOM: Eternal)**
- **Porting Legacy Mods to Atlan**
- **Converting idStudio Mods to Atlan**
- **Eternal Atlan Strings Modding**

Making Mods: Getting Started (DOOM: Eternal)

Contains some quick links for getting started with modding DOOM: Eternal.

This wiki contains many resources for creating mods. You should start by reading the **Atlan Fundamentals** to learn the basics. From there, you can choose what documentation to read based on the types of mods you're making.

Porting Legacy Mods to Atlan

Atlan Mod Loader now supports DOOM Eternal, and will be replacing Eternal Mod Loader. Mods created for Eternal Mod Loader will not work out-of-the-box with Atlan. This page discusses all changes that need to be made to a mod zip for it to work with Atlan.

Atlan Legacy Converter

To convert an Eternal Mod Injector zip to an Atlan zip, use Atlan Legacy Converter! This tool is built into Atlan Mod Packager. In most cases, you do not need to be the original mod author to use this tool.

To use it:

1. Download **Atlan Mod Packager** and unzip it into your DOOM Eternal installation folder.
2. Drag and drop your Eternal Mod Injector zip files onto `AtlanModPackager.exe`
3. Output: For each zip file you convert, you'll receive an output zip with the name `<Original_Zip_Name>_AtlanPackage.zip`
4. Pay attention to the output log, named `packager_log.txt`. If any files failed to convert, or require manual conversion, the output log will tell you. If this is the case, proceed to the next section in this page.

Advanced users may wish to run the legacy converter through the command line. Here is an example of how to do this:

```
AtlanModPackager.exe --gamedir "path_to_doom_eternal_folder" "zip_1.zip" "zip_2.zip"
```

Any number of zip files may be specified.

Manual Conversion

The overwhelming majority of legacy injector mods should convert to an Atlan zip without issue. However, some specific file types will require manual conversion. See the following table.

File Type	Remarks
Decls	Fully supported. New path is `decls/<path>.decl`
Entities	Fully supported. New path is `compfile/<path>.entities`

SWF Files	Fully supported. New path is `cswf/<path>.swf`
Strings JSON	Fully supported. New path is `binaryFile/<path>`
Eternal Mod JSON	Fully supported. New path is `atlanmod.txt`
Assets Info JSON	Manual conversion required. See below.
Static Model	Fully supported. New path is `model/<path>`
Audio: .wem	Fully supported. New path is `audio/<path>`
Audio: .opus	Manual conversion required. See below.
Images	Manual conversion rarely required. See below. New path is `image/<path>`
Other Files	Presumably unsupported. Please discuss with tool creators.

Converting Assets Info JSONs

Atlan uses a completely different format for it's Assets Info. You must manually convert your Assets Info JSON files to this new format. See **this page** for instructions.

Converting OPUS Audio Files

Atlan Mod Loader cannot load OPUS audio files. In order to load legacy SFX audio mods with Atlan, you must convert your OPUS files to WEMs.

1. Convert your OPUS file back to a WAV. Tools like Audacity can do this.
2. Use WWise to convert your WAV to a WEM. See **this page**.
3. The new path of your file in your mod zip should be `audio/<path>.wem`

Converting Image Files

Atlan Legacy Converter will successfully convert the majority of texture files. However, some files have ambiguous names that must be manually resolved. To do this:

1. Extract the texture file from your Legacy zip
2. Rename the file to have the exact extension metadata of the file you want to replace. Use resource explorer tools like **Valen** to identify the correct extension metadata.

3. Put the image into a new zip file, keeping the same path as it had in the legacy zip. Convert that zip with Atlan Legacy Converter.

Converting idStudio Mods to Atlan

Atlan Mod Packager can convert idStudio mods into Atlan mods. The contents of `base.pk5` are parsed and placed into an Atlan zipfile. You do not need to be the mod author to use this conversion tool. However, if any edits are required to ensure compatibility with Atlan, you'll probably need to contact the mod author.

Usage

To run the idStudio Mod Converter:

1. Download **Atlan Mod Packager** and unzip it into your DOOM Eternal game folder.
2. Drag and drop the idStudio mod's `modinfo.json` or `modinfo.dev.json` onto `AtlanModPackager.exe`
3. Output: `AtlanPackage.zip` will be created inside the idStudio mod folder.
4. Take note of any errors in the console output. All console output is written to `packager_log.txt`

Advanced users may wish to run the idStudio mod converter from the command line. You may do this as follows:

```
AtlanModPackager.exe "path/to/modinfo.json"
```

Only one idStudio mod can be converted per execution of the packager.

Supported File Types

Not all file types encountered in a PK5 can be built into an Atlan mod zip. The following file types are supported. Adding support for more file types is possible, if there is sufficient demand.

Type	idStudio Path	Atlan Zip Path
Decls	generated/decls/	decls/
Strings	strings/<...>.blang	binaryFile/
Audio Files	sound/<...>.wem	audio/
Entities	generated/compfile/	compfile/

SWFs	generated/cswf/	cswf/
Images	generated/image/	image/
Static Models	generated/model/	model/
Rigged Models	generated/basemodel/	baseModel/
Animations	generated/anim/	anim/
Skeletons	generated/skeleton	skeleton/
Binarymd6defs	generated/binarymd6def	binarymd6def/

Atlan Config File

You may place an `atlanmod.txt` config file in your idStudio mod folder. Place it in the same folder as the modinfo JSON, and it will be packaged into the zip file. See **this page** to read about the Atlan config. file format.

idStudio Level Packager

> This is an advanced feature that requires careful preparation and attention to detail. Please read the wiki pages on **Packaging idStudio Mods** and **Known Fixable Issues** before proceeding.

Atlan Mod Packager can package idStudio level mods into a form that's usable with Atlan Mod Loader.

Atlan Config. File

To package an idStudio level, you **must** have an `atlanmod.txt` in your idStudio mod folder. Your config file **must** have the `load_level` property set. This tells the packager what to name the folder your level files get placed in.

This name must only contain lowercase letters and underscores. To prevent conflicts with other Atlan level mods, **this name should be unique to your level mod.** Some examples:

```
// BAD: Not unique names, may be used by other level mods and cause conflicts
load_level {
	mod
}
load_level {
	customlevel
}
```

```
load_level {
	hell_level
}

// GOOD: Sufficiently unique names that nobody else may accidentally use
load_level {
	velser_hell_crusade
}
load_level {
	svs_pvp_hurricane
}
```

PK5 Contents

Whether your level mod correctly works, is determined by whether your `base.pk5` is packaged correctly. To understand why, you must know how the level packager works:

Packaging a level in idStudio produces several archives:

- One `gameresources_mod.resources` and one `warehouse_mod.resources` archive, which get loaded globally.
  - These archives are typically very small, and mostly contain vanilla game assets. However, some files unique to your mod that need to be loaded globally may be packaged into these archives.
- One `game/shell/shell.resources` archive
  - This archive should always be empty, unless you're modifying something related to the main menu.
- One level-specific `.resources` that only loads when playing your custom level.
  - This is typically quite large, and will contain all your level's assets.
- One `.streamdb` file.
  - This is typically quite large, and is necessary for your mod to work properly.

Atlan does not package the gameresources or warehouse archives. Only the level-specific archives and the .streamdb file are packaged. Atlan relies on the assumption that any files packaged in these global archives are also packaged in the `base.pk5`. Atlan will analyze your level mod's global archives, and alert you if any unique files aren't included in the PK5. **Pay close attention to the console output, which gets written to the packager_log.txt. You may receive some false-negatives about files your mod doesn't edit. These can likely be ignored.**

Packager Output

When packaging an idStudio level, Atlan Mod Packager will generate 2 zip files:

`AtlanPackage.zip` - This is the zip file normally generated when packaging any idStudio mod. It contains your mod's Atlan config file, and the contents of `base.pk5`. There is nothing different about it.

`AtlanPackage_InstallOnce.zip` - A zip file unique to the level packager. It will contain:

- Your level's .resources and .streamdb archives, stored in `modarchives/<load_level>`
- Your level's baked lightmaps, stored in `virtualtextures/`
- A generated config. file with asset lists for your level mod's global assets.
  - **Do not modify this config. file if you don't know what you're doing.**
  - Found at `modarchives/<load_level_property>/atlanmod.txt`
- Another generated config. file with the `install_once` property set
  - This property tells Atlan Mod Loader to perform a level mod installation with this zip. This zip file will be deleted after installation is complete, keeping it's contents permanently unzipped on the user's disk.
  - The archives are unzipped and stored in `base/modarchives/<load_level>` while the lightmaps are unzipped into `virtualtextures`
  - This config file is named `atlanmod.txt` and is stored at the root of the zip file.

Loading Level Mods

Installing a level mod does not mean it's archives always get loaded by the game. This is where your `AtlanPackage.zip` (the Atlan mod with the PK5 data) matters. The config file's `load_level` property instructs Atlan to load your level's archives. This also means that multiple level mods can be loaded simultaneously, if several mods have a `load_level` property in their config. files.

Eternal Atlan Strings Modding

Atlan Mod Loader supports injecting data into Doom Eternal's binary language (blang) files.

Extracting Vanilla Strings

To extract Doom Eternal's vanilla string data, use **Atlan Resource Extractor** to extract resources of type `binaryFile`. This will dump the game's vanilla strings to CSV files. Each language has it's own blang file.

Creating String Mods

To setup a string mod for a particular language, add a new CSV file to your mod that matches the original .blang path.

For Example: to inject data into `binaryFile/strings/english.blang`

your CSV path should be `binaryFile/strings/english.csv`

The file extension doesn't matter. It can be .csv, .txt, or anything else you prefer.

CSV Format

Atlan Mod Loader requires your CSV obey the following format: `#STRING_NAME "STRING TEXT"`

To summarize:

1. Your string name MUST start with a `#`
2. Your string text MUST be wrapped in quotes `"Like this"`
3. The following escape sequences are supported: `\n \r \\ \" \t`
4. OPTIONAL: You may have a comma between your string's name and text.
5. HIGHLY RECOMMENDED: Your string names should only use lowercase letters, underscores, and numbers

You may refer to the vanilla string CSV files as examples. These are all valid string CSVs that Atlan can load.

You can replace the content of existing strings or add new strings.

The following is a basic example CSV that replaces several strings found in the main menu.

```
#str_code_mainmenu_dlc1_name "THE ANCIENT CHUNGUS 1"
#str_code_mainmenu_dlc2_name "THE ANCIENT CHUNGUS 2"
#str_code_mainmenu_extras_name "SUPER COOL EXTRA STUFF"
#str_code_mainmenu_horde_name "HORSE"
```

Default CSV

Doom Eternal supports 13 different languages, each with it's own blang file. If you're only adding strings to your native language's blang, your custom strings will not exist when the game is played in other languages. To ensure parity between all languages, you may wish to add your untranslated strings to every blang file.

Atlan Mod Loader supports this with a special CSV: `binaryFile/strings/default.csv`

Strings in this CSV get added to all blang files that don't have their own CSV in the same mod.

For example: Let's say you have a mod with `binaryFile/strings/default.txt`

and `binaryFile/strings/french.txt`

1. The data inside `french.txt` gets injected into the French blang
2. The data inside `default.txt` gets injected into every blang **except** French

This way, your mod can have a default/fallback CSV while being localized in specific languages.

Additional Information

To learn how to add colored text and other customization to your strings: see **String Customization**