ModInjector JSON Guide

Information about the Modinjector JSON format

assetsinfo JSON

The "assetsinfo" JSON can be used to add entirely new assets to the game.

Format Overview

> The assetsinfo JSON format allows for advanced control over mod injector behavior.

The most common uses of assetsinfo JSON include:

- Adding brand new textures or .decl files to the game (instead of just replacing another file)
- Controlling which of the game's .resources and .streamdb archives are loaded in a map (this allows modders to use assets that aren't normally loaded in the level/map they're working with).

Format Overview

```
{
// experimental - used to add new map layers
    "layers":[
        {
            "name":""
        }
    ],

    // experimental - used to add new maps
    "maps":[
        {
            "name":""
        }
    ],

    // used for adding, removing, or re-ordering of .resources files
    "resources":[
        {
            "name":"",
            "remove":,
            "placeFirst":,
            "placeBefore":,
```

```
            "placeByName":""
        }
    ],

    // used for adding, removing, or re-ordering of game assets
    "assets":[
        {
            "name":"",
            "mapResourceType":"",
            "resourceType":"",
            "streamDbHash":,
            "version":,
            "remove":,
            "placeBefore":,
            "placeByName":"",
            "placeByType":"",
            "specialByte1":,
            "specialByte2":,
            "specialByte3":
        }
    ]
}
```

"Layers" array

The first part of the assetsinfo JSON format is the `layers` array. This can be used to add additional layers to a map, which can then be referenced via .entities.

Each item in this array contains an object that has the `name` property. This field is required for each entry. In there you can set the name of the layer you want to create. It's that simple.

Here is an example usage that adds three custom layers:

```
{
    "layers":[
        {
            "name":"game/sp/custom/my_custom_layer"
        },
        {
            "name":"game/sp/custom/my_custom_layer_2"
```

```
        },
        {
            "name":"game/sp/custom/my_custom_layer_3"
        }
    ]
}
```

> **Note:** It is unclear what effect this has on gameplay, since it is possible to add new layers to the game without this step, and all existing layers in the game can already be used in any level without additional steps anyway.

"Maps" array

The second part of the assetsinfo JSON format is the `maps` array.

As with the `layers` array, each item contains an object that has the `name` property. In there you can set the name of the map you want to create. This feature is currently of limited use, since we can't create new maps anyway.

Here is an example usage that adds two custom maps:

```
{
    "maps":[
        {
            "name":"maps/game/sp/custom/my_custom_map"
        },
        {
            "name":"maps/game/sp/custom/my_custom_map_2"
        }
    ]
}
```

"Resources" array

The third part of the assetsinfo JSON format is the `resources` array. This one is a bit more complicated, so let's begin by explaining what this is actually for.

In DOOM Eternal, each map has a list of **.resources** and **.streamdb** files that will be loaded in the map itself. With this feature, you can tell the game to load additional .resources and/or .streamdb files in maps. In other words, **you can load files that aren't normally loaded**, giving you access to game assets and entities that normally aren't available. You can also tell the game to stop loading .resources

or .streamdb files that are loaded by default in maps.

> This feature is what allows us to load things like spirits and other DLC-specific enemies into the base campaign.

Below is an example usage. This would add the `e5m1_spear.resources` file, as well as several .streamdb files, to a map that it is not normally loaded in. Don't worry if you don't understand this - it is explained in more detail on the **Reference: "resources" array** page of the guide**.**

```
{
    "resources":[
        {
            "name":"e5m1_spear.resources"
        },
        {
            "name":"gameresources_6_1.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        },
        {
            "name":"gameresources_0_6.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        },
        {
            "name":"gameresources_1_8.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        },
        {
            "name":"gameresources_2_2.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        },
        {
            "name":"gameresources_3_2.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        },
```

```
        {
            "name":"gameresources_4_3.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        },
        {
            "name":"gameresources_5_3.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        }
    ]
}
```

"Assets" array

The last feature in the assetsinfo JSON is the `assets` array. This is the most complicated one to understand as it has a lot of fields. In most cases, you will only need to use a few of them.

The main purpose of this section is to add entirely new assets to the game. (Most mods work by replacing another asset that already exists, rather than adding something new). For more details, refer to the **Reference: "assets" array** page of the guide.

An example usage is below. This would add a new image file to the game:

```
{
    "assets":[
        {
            "resourceType":"image",
            "version":21,

"name":"custom_textures/custom_emissive_texture.tga$bc1$streamed$mtlkind=bloommask",
            "mapResourceType":"image"
        }
    ]
}
```

See Also

- **Reference: "resources" Array**

Reference: "resources" Array

> In DOOM Eternal, each map has a list of *.resources* and *.streamdb* files that will be loaded in the map itself. With this feature, you can tell the game to load other *.resources* and/or *.streamdb* files in maps. You can also tell the game to stop loading *.resources* or *.streamdb* files that are loaded by default in maps.

Structure

The basic structure of the "resources" array is below:

```
{
"resources":[
{
"name":"",
"remove":,
"placeFirst":,
"placeBefore":,
"placeByName":""
}
]
}
```

- **`name`** (string) (required) - the name of the file to load, e.g. `"e5m1_spear.resources"`
- **`remove`** (bool) (optional) - if set to `true`, the file specified by the **`name`** property will be removed from the map. Defaults to `false` if not set.
- **`placeFirst`** (bool) (optional) - if set to `true`, the file specified by the **`name`** property will be loaded with the highest priority in the map. Defaults to `false` if not set.
- **`placeBefore`** (bool) (optional) - if set to `true`, this resource will be loaded before the resource named in the **`placeByName`** property. Defaults to `false` if not set (meaning it will be loaded after).
- **`placeByName`** (string) (optional) - used with **`placeBefore`** to control the order in which resource files are loaded. For example, setting `"placeBefore": true` and `"placeByName":"e1m1_intro.resources"` will cause the file specified by **`name`** to be loaded with a higher priority than e1m1_intro.resources.

If neither `placeBefore` nor `placeByName` are set, then the asset will be loaded last, with the lowest priority.

Example Usage

The most common use of this feature is to give custom map/level creators access to assets that aren't normally available in a map. For example, you can tell the game to load **e5m1_spear.resources** (and the required .streamdb files) within another level, such as e1m1_intro.

By doing this, a person modifying the e1m1_intro map can then freely access files/entities from e5m1_spear.resources, the same as if they existed in e1m1_intro already. This would allow you to use DLC AI such as spirits (and many other things) in e1m1_intro, where they normally can't be used.

Below is a sample .json snippet that loads the e5m1_spear.resources and .streamdb files.

```
{
    "resources":[
        {
            "name":"e5m1_spear.resources"
        },
        {
            "name":"gameresources_6_1.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        },
        {
            "name":"gameresources_0_6.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        },
        {
            "name":"gameresources_1_8.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        },
        {
            "name":"gameresources_2_2.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        },
```

```
        {
            "name":"gameresources_3_2.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        },
        {
            "name":"gameresources_4_3.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        },
        {
            "name":"gameresources_5_3.streamdb",
            "placeBefore":true,
            "placeByName":"gameresources_0_4.streamdb"
        }
    ]
}
```

How Do I Know Which Files to Add?

First, take a look at this spreadsheet (note: there are multiple tabs at the bottom):

https://docs.google.com/spreadsheets/d/10R5fWKGyqPuCPdueqKeCYycHETSHg0QeFOCbWIQZUCY/

This spreadsheet contains the list of the default *.resources* and *.streamdb* files that are loaded in all the game's maps. So, let's say that we want to have access to all the assets in **e5m1_spear.resources** in **e1m1_intro.resources**. The first thing we need to do is find the "e5m1_spear" map name in the spreadsheet, using the "map name" column.

You will see that the map name appears multiple times. This is because the map loads multiple .resources and .streamdb files. In the "file name" column you will see the names of the files that are loaded in the map. The order of the files matters. It actually goes from bottom to up, so the files at the bottom are loaded first during the map loading process, and the files at the top are loaded last. This means that the files at the top override the others (if they share files in common). This is what we refer to as **load priority**.

EternalMod JSON

> This .json file allows mod makers to define information about their mods, such as mod name, author(s), description and version number.

> This feature is only available for zipped mods. You can not place an EternalMod JSON file at the root of your *~/Mods/* directory; it will be ignored.

How to Use:

Using the EternalMod.json file is simple:

1. Create a file named **"EternalMod.json"**
2. Place the `EternalMod.json` file at the root of your zipped mod, outside of the container folders. **Example:** `~/Mods/My_Amazing_Mod.zip/EternalMod.json`
3. Use a text editor such as Notepad++ to edit the file as needed.

Format:

This is the format of this JSON file (all fields are optional):

```
{
    "name":"",
    "author":"",
    "description":"",
    "version":"",
    "loadPriority":,
    "requiredVersion":
}
```

- **name**: *(string)* The name of your mod.
- **author**: *(string)* The author(s) of the mod.
- **description**: *(string)* Description of what your mod does.
- **version**: *(string)* The version number of your mod.

- **loadPriority**: *(integer)* This field determines the load priority of your mod. The lower the number, the higher is the priority. Mods with higher priorities will be loaded last during the mod loading process. For example, a mod with load priority `-5` will be loaded *after* another mod with priority `5`). If the load priority is not specified, it will be `0` by default. Unzipped (loose) mods will always be loaded last.

- **requiredVersion**: *(integer)* The minimum mod loader version that is required to install your mod. Normally, you should set this to the current mod loader version number. In order to check the version number the current mod loader, you can run it with the '--version' option: `DEternal_loadMods.exe --version`

Example File:

Here's an example of an **EternalMod.json** file with all the fields filled in:

```
{
    "name":"Enemy Randomizer",
    "author":"proteh",
    "description":"This mod randomizes every enemy encounter in the base game and the DLCs.",
    "version":"3.0",
    "loadPriority":100,
    "requiredVersion":6
}
```