container.mask

About

In idTech7 and 8, `container.mask` is the single file located inside the `meta.resources` archive. It is a set of bit-masks, one for every resource archive in the game. Each file inside the archive has a bit in the mask, based on the order they appear inside the archive's file list. If a file's bit is `0` , then that file will not be loaded by the game.

This file is built to guarantee that only one version of a file will ever be loaded by the game simultaneously. When a game update adds new resource archives, they often contain updated versions of existing files. Only the latest copies of a file will have their bitmask files set to `1` . When combined with the `packagemapspec.json` , the container mask creates 2 layers of security for ensuring the correct copies of a file are loaded.

Edge Cases

When determining the highest-priority version of a file, it was previously thought you only needed to refer to the `packagemapspec.json` file. This assumption is wrong. Edge cases exist where the container mask disables the version of a file found in the higher-priority archive, and enables a version in a lower-priority archive instead. This may happen in instances where id modifies - then later reverts - a file over the course of several game updates. Therefore, you **must** use the container mask to determine which copy of a file is actually used by the game.

If multiple versions of a file exist, and **all** of them are disabled by the container mask, you cannot determine the latest version with absolute certainty. You could go by highest-priority archive or by file timestamps, but neither of those tools will be perfect.

Enabling Multiple Versions of a File

What if two different versions of a file are both enabled via the container mask? In the vanilla filesystem, this does not happen. However, when it does occur due to modding, the game will use the version from the highest-priority archive, according to the packagemapspec.json `files` array.

Format

The file format is fairly straightforward

```
struct bitmask {
    uint64_t    hash;       // Hash used to identify the resource archive
    uint32_t        size;           // Size of the mask, in 64-bit integers
```

```
    uint64_t*       mask;           // Array of 64 bit integers representing the raw bitmask.
length == size
}

struct maskfile {
    uint32_t    timestamp;      // Not present in idTech8
    uint32_t    num_bitmasks;   // Number of bitmasks in the file
    bitmask*    bitmasks;       // Array of bitmasks, length == num_bitmasks
}
```

Hashes

As seen in the structures, the container.mask uses hashes to associate each archive with a bitmask. These are Farmhash64 hashes of each archive's metadata section. The hashed data begins after the archive's header, and ends at the `IDCL` magic that terminates the meta section. (The magic is included in the hashed data)

This code shows how to calculate a container mask hash for an archive.

Extra Bitmask Slots

As seen in the structures, the bitmasks are defined as 64-bit integers. This results in several implicit behaviors:

- The maximum number of files that can be stored in a resource archive is always a multiple of 64. Trying to store too many files (such that you exceed the size of the bitmask), will likely cause crashes or instability.
- If the number of files in a resource archive is not a multiple of 64, the bitmask will have some unused bits at it's end. These bits are either all `1` or all `0` - with their exact value determined by whether or not it's the highest-priority archive in it's patch group.

```
Example: common.resources, common_patch1.resources and common_patch2.resources

If the archive priority (based on packagemapspec.json) is:
- common_patch1.resources
- common_patch2.resources
- common.resources

Then:
```

```
- All extra bitmask bits in common_patch1 will have a value of 1
- All extra bits in every other archive's bitmask will have a value of 0
```

Container Mask (Audio Archives)

The `.snd` archives for storing audio files in idTech7 and 8 also have their own container mask. It is located in the `soundmetadata.bin` file. It's stored near the beginning of the file in DOOM Eternal. In DOOM The Dark Ages, it's at the end of the file.

Format

The audio container mask format is slightly more complex

```
struct sndBitmask {
    uint32_t  hash;      // Identifies the .snd container associated with this bitmask
    uint32_t  mask_size; // Size of the bitmask (in 32-bit integers)
    uint32_t* bitmask;   // Array of integers representing the raw bitmask. Length == mask_size
}

// Encompasses a group of snd archives
// (i.e. SFX.snd, SFX_Patch_1.snd and SFX_Patch_2.snd all belong to one mask group)
struct sndMaskGroup {
    uint32_t    group_name_length;
    char*       group_name;    // Group name string. Not null-terminated. Length ==
group_name_length
    uint32_t    num_bitmasks;  // Number of bitmasks in this group
    sndBitmask* bitmasks;      // Array of bitmasks. Length == num_bitmasks
}

struct sndMaskChunk {
    uint32_t      num_groups; // Number of mask groups in the chunk
    sndMaskGroup* groups;     // Array of mask groups. Length == num_groups
}
```

Hashes

The audio container mask also uses hashes to associate each .snd archive with it's bitmask.

AudioKinetic's case-insensitive FNV hashing algorithm is used. **Here is the algorithm.**

The hash is performed on the filename without it's extension. (Example: if the container name is `SFX_Patch_1.snd` then the string `SFX_Patch_1` is hashed.)

In DOOM The Dark Ages, certain snd archives have special, hard-coded hashes instead of regular FNV hashes. Specifically, `sound/soundbanks/pc/MUSIC.snd` has a hash of 0, and `sound/soundbanks/pc/SFX.snd` has a hash of 1. The handheld versions of these archives in `sound/soundbanks/hhpc` have regular FNV hashes.

Revision #2
Created 11 February 2026 06:16:23 by FlavorfulGecko5
Updated 3 March 2026 01:36:03 by FlavorfulGecko5