Halo: Combat Evolved Map Structure

Snowy Mouse 

Revision 2.1.2

This is a guide on the structure of the map files built for the game, Halo: Combat Evolved. Games discussed are as follows:

- Xbox version - Halo: Combat Evolved on Xbox as released by Bungie
- Retail PC version - Halo: Combat Evolved as released by Gearbox (and MacSoft for the Mac version)
- Demo version - Demo / Trial versions of the PC version of the game
- Custom Edition version - Halo Custom Edition on PC, an expansion to the PC version that enables the creation of custom maps
- Anniversary version - Halo: Combat Evolved Anniversary on MCC (also known as CEA)

This document will not discuss the Xbox 360 version of Halo: Combat Evolved Anniversary, nor will it discuss betas of the game due to the fact that these are numerous and also outdated. This document will also not discuss any community-created formats, as that would make it difficult to maintain and update this document every time someone made another format or updated an existing one.

This document will also not go over the structure of tags, themselves, as this would require a lot of work as well as possibly hundreds of pages. If you actually need this information, you can find an up-to-date set of definitions in the Invader repository which can be found on GitHub at https://github.com/SnowyMouse/invader/

Contents

1 About the author

The author, Snowy, is a programmer who has worked on modding and fixing Halo: Combat Evolved for over 14 years. She created the Chimera mod, a mod that fixes a number of issues with the PC version of the game, including fixes that are still not present on MCC. She also wrote a number of fixes for the Mac version of the game, having originally played Halo: Combat Evolved through this exact version.

Snowy also created Invader, a project that aims to be a complete reimplementation of the Halo Editing Kit that supports modern platforms while also remaining completely free (as in libre). You can also find these projects at https://github.com/SnowyMouse along with a number of other projects, including work on other games and systems as well as contributions to open source projects such as Tutanota.

If you have any questions, comments, or other forms of feedback, the best way to contact Snowy is by e-mailing contact@snowymouse.com as this address is almost always checked daily.

You can also contact Snowy on Discord. Note that friend requests from strangers do not usually get accepted. Instead, you should join a server she is in, first, and then contact her from there. Here are some servers you can try:

- Invader — https://discord.gg/RCX3nvw
- Open Carnage — https://discord.opencarnage.net
- Halo CE: Refined — https://discord.gg/QzSR2xNGzp
- Halo CE Reclaimers — https://discord.reclaimers.net

2 What is a map file?

Map files are typically levels that the game loads. They are an archive of all of the tags used for a specific scenario, and they are typically loaded at a specific memory address. They also contain memory pointers that are valid once loaded.

Officially, maps use the .map extension on all versions of Halo, even including later versions of the game.

2.1 Resource maps

The Windows and Mac versions of Halo: Combat Evolved, as released by Gearbox and MacSoft, respectively, also includes two additional files with .map extensions: bitmaps.map and sounds.map. Halo Custom Edition introduces a third file, loc.map. These are not actual cache files and are not generally loaded directly by the game. Instead, the game uses it to share data between maps. For the retail version, this is to save disk space. For Halo Custom Edition, this is to provide basic localization so maps will appear to be the correct language regardless of the language the user is playing. Consequently, Halo Custom Edition's resource maps contain the actual tag data for bitmaps, sounds, strings, and fonts.

No other version of the game has resource maps presently. However, Halo: Combat Evolved Anniversary on MCC does support loading Halo: Custom Edition maps (thus using these three caches for loading such maps).

2.2 Halo: Combat Evolved Anniversary resource files

Halo: Combat Evolved Anniversary has its own system for loading bitmap and sound data through ipaks and FSBs.

ipak is a format made by Saber Interactive. It is used for bitmaps in Halo: Combat Evolved Anniversary maps if playing on Halo: Combat Evolved on MCC, and this applies to both the anniversary graphics and classic graphics. Similar to bitmaps.map, it contains bitmap data. However, unlike bitmaps.map, it does not always correspond to the actual tag data. For example, a bitmap tag may claim to be 256x256 16-bit, but the bitmap in the ipak might be larger, such as 512x512 with 32-bit color.

FSBs (FMOD sound bank, not front-side bus!) are a proprietary format for the FMOD sound effects engine used by Halo: Combat Evolved Anniversary. Like sounds.map, it contains sound data, and like ipak, this sound data does not correspond to tag data. As such, it can be various formats (e.g. MPEG, Ogg Vorbis, PCM, ADPCM, etc.). Because FSBs do not contain mouth data yet all languages of this version of the game had their maps built with English tags, stock Halo: Combat Evolved Anniversary maps will only play English mouth data even when on another language.

3 Data types

These are the various data types used in these structures. Everything in a cache file is little endian (except for Xbox 360 CEA maps which are big endian, but these aren't discussed here) with a word size of 32 bits. HEK tag files are big endian, but this document concerns only the compiled cache files and not necessarily the files used to create them.

Type	Description
char	8-bit signed character type (ISO 8859-1 encoding)
uintN	Unsigned integer of N bits (8 bits = 1 byte)
void	Ambiguous type
X[N]	Array of X containing N elements
X*	Pointer of type X (same size as uint32 in a Halo: Combat Evolved cache file)

These are all based on C types, as this game and its tools are largely written in the C and C++ languages. uint32 here, for example, is the same as the uint32_t type in the C standard library header <stdint.h> (or std::uint32_t from <cstdint> on C++).

This document is written in such a way where you do not need to understand C or C++ to understand its contents. However, you do need to understand hexadecimal, and some experience with using tools like a hex editor can be quite helpful here.

3.1 Now you're thinking with pointers

One of the hardest concepts to grasp regarding these data structures is pointers and memory addresses. An address is a number that refers to a location in memory, where a pointer is a data field that holds an address. All addresses are unsigned integers, thus a pointer technically holds an unsigned integer (i.e. uint32 since Halo: Combat Evolved cache files are 32-bit).

For example, the tag array pointer is typically equal to 0x40440028, and 0x40440028 is the address to the tag array when the map is loaded.

Imagine memory as a stack of cards. An address would work like this: 0 is the address of the top card, 1 is the card below the top card, 2 is the card below that, and so on until you get to 51 which is the bottommost card (in standard playing card decks of 52 cards). This is basically how addresses work in memory, but instead of cards, it's bytes! In a 32-bit address space, there are over 4 billion possible memory addresses (from 0x00000000 to 0xFFFFFFFF) that can technically be accessed. Not all of it is used at one point, but that doesn't really matter for our purposes.

4 Cache file header

The header of a cache file is composed of several fields, but it is always 2048 bytes in length (including padding which is most of it), and it is always the start of a cache file.

4.1 Header structure

Offset	Type	Name	Description
0x0000	uint32	Magic	Must be 1751474532 ("head")
0x0004	uint32	Cache version	Must match the game's cache version
0x0008	uint32	File size	Length of the map when uncompressed
0x000C	uint32	Padding length	Padding after the map (Xbox only)
0x0010	uint32	Tag data offset	File offset of the tag data
0x0014	uint32	Tag data size	File length of the tag data in bytes
0x0020	char[32]	Scenario name	Must match filename (except in CEA)
0x0040	char[32]	Build version	Must match engine version in Xbox
0x0060	uint16	Scenario type	Determines which cache to use on Xbox*
0x0064	uint32	Checksum	CRC32 of the BSP data, model data, and tag data
0x07FC	uint32	Magic	Must be 1718579060 ("foot")

*See Compressed cache files

4.1.1 Demo header structure

The header structure was changed on the demo version. This was likely to prevent the demo version of the game from loading full version maps. To make it even harder, the base memory address was changed as well, and this will be discussed in a later section. The padding in demo maps was also filled with garbage, and the "padding length" field was removed due to being unused on PC (or it was filled with garbage as well, thus it's impossible to find it).

Offset	Type	Name	Description
0x02C0	uint32	Magic	Must be 1164469604 ("Ehed")
0x0588	uint32	Cache version	Must match the game's cache version
0x05E8	uint32	File size	Length of the map when uncompressed
0x0010	uint32	Tag data offset	File offset of the tag data
0x02C4	uint32	Tag data size	File length of the tag data in bytes
0x058C	char[32]	Scenario name	Must match filename (except in CEA)
0x02C8	char[32]	Build version	Must match engine version in Xbox
0x0002	uint16	Scenario type	Determines which cache to use on Xbox*
0x05B0	uint32	Checksum	CRC32 of the BSP data, model data, and tag data
0x05F0	uint32	Magic	Must be 1197895540 ("Gfot")

*See Compressed cache files

4.2 Cache versions

For reference, here are the cache versions. This is used for verifying if the map is even compatible with the current engine.

Version	Game
5	Xbox
6	Demo
7	PC / CEA (MCC pre-Season 7, Xbox 360)
13	CEA (MCC post-Season 7)
609	Halo Custom Edition

Additional cache versions are used for later games (e.g. Halo 2 Vista uses version 8). For the sake of simplicity, only Halo: Combat Evolved cache versions will be listed here.

4.3 Resource map header structure

Although not technically a cache file, resource maps can contain tags on Halo Custom Edition.

Offset	Type	Name	Description
0x0000	uint32	Type	0=bitmaps, 1=sounds, 2=loc
0x0004	uint32	Paths offsets	File offset to the path strings
0x0008	uint32	Resource offset	File offset of the resource array
0x000C	uint32	Resource count	Number of resources

4.3.1 Resource

Each individual resource is listed like this, with 12 bytes per resource in an array.

Note that the path offset is not a file offset. To get the file offset, add the "Paths offset" value from the header.

Offset	Type	Name	Description
0x0000	uint32	Path offset	Offset to the null terminated path from "Paths offset"
0x0004	uint32	Data size	Size of the resource in bytes
0x0008	uint32	Data file offset	File offset to the data of the resource

4.4 Compressed cache files

For some versions of the game, cache files are compressed. This includes the Xbox version as well certain files in Halo: Combat Evolved Anniversary.

4.4.1 Xbox compression

The Xbox uses DVDs. These are limited in capacity (4.7 or 8.5 GB for a dual-layered disc), thus compression is often utilized to fit as much data the disc as possible. Also, the DVD drive is fairly slow, being only a few megabytes per second, and it can carry a high latency. As you can imagine, loading data directly from it can be quite slow.

To get around this, the Xbox has three cache partition for its games on the hard drive. It is here where Halo: Combat Evolved stores its uncompressed maps as well as some of the game state data.

Xbox maps are compressed using a single zlib stream of everything after the header. The header is then copied to a cache on disk along with the uncompressed data (the cache is reserved to a fixed size across several files on the cache partition), and then the game reads the data from here. The "scenario type" field in the header indicates which reserved file to decompress to. These correspond to build 2276 (english NTSC):

Value	Type	Size	Number of caches
0	Single Player	278 MiB	2
1	Multiplayer	47 MiB	3
2	User Interface	35 MiB	1

If it's set to 0 (singleplayer), it will use one of the two 278 MiB caches. If it's set to 1 (multiplayer), it will use one of the three 47 MiB caches. Lastly, if it's set to 2, it will use the 35 MiB cache.

With this, ui.map only ever has to be decompressed once in a gameplay session. The last two Single Player maps you played also do not have to be re-decompressed, allowing you to quickly go back to the last level or resume a checkpoint without a long wait time. Multiplayer maps obviously get the most since it's very easy to play several multiplayer maps.

4.4.2 CEA compression

CEA's Saber3D files are compressed in chunks. Prior to an update, so were maps. These archives started with the chunk count followed by file offsets to each chunk. Each chunk started with the decompressed size of the chunk followed by the zlib stream. By doing this, compression and decompression can be threaded, and you can do random reads from the file. However, compression ratio is worse when you do this, and if doing this to a map, compressing the header made it so the map could not be identified as a cache file or parsed without first decompressing the header.

5 Tag data

This is the tag data section. On Xbox, this also contains model data, where on later versions, model data is stored in a separate location of the map. BSP data is stored separately but loaded into this region of memory dynamically by the game (one BSP at a time), and information on how to access that is stored in the scenario tag through the structure BSP array (most of this is normally padding in a .scenario tag file but is set when the scenario is compiled into a cache file).

5.1 Tag data address and size

Halo PC and demo both have a 23 MiB tag space, where the Xbox version has a 22 MiB tag space (larger on later versions of the game). CEA, on the other hand, has a 31 MiB tag space.

To translate a pointer to a file offset, add the file offset of a known memory address and subtract this known memory address.

In Halo PC, tag data is loaded at 0x40440000 (or 0x4BF10000). On Xbox, tag data is loaded at 0x803A6000. On CEA, the tag data can be loaded anywhere, and it uses the tag data address minus the size of the header (0x28) to determine the base address.

For example: To find the file offset of 0x40440028 on Halo PC, subtract 0x40440000 and add the file offset of the tag data.

5.2 BSP header

BSP tags have a header that points to the actual BSP tag. There are other fields here, too, but Halo PC only uses the tag pointer. The Xbox version uses the other fields, and some CEA maps use some of these fields, too.

Offset	Type	Name	Description
0x0000	void*	Tag pointer	Pointer to the base struct of the BSP tag
0x0004	uint32	Lightmap material count	Number of lightmap materials
0x0008	void*	Rendered vertices	Pointer to an array of rendered vertices
0x000C	uint32	Lightmap material count	Number of lightmap materials repeated
0x0010	void*	Lightmap vertices	Pointer to an array of lightmap vertices
0x0014	uint32	Tag FourCC	FourCC of the tag (usually "sbsp")

5.3 Header structure

Beginning the tag data is another header. This is different between the Xbox version and later versions of the game, but these first five fields are the same:

Offset	Type	Name	Description
0x0000	void*	Tag array pointer	This is the pointer to the tag array
0x0004	uint32	Checksum	This is a checksum of the tag files used in the map
0x0008	uint32	Scenario ID	Tag ID of the scenario tag
0x000C	uint32	Tag count	Number of tags in the map
0x0010	uint32	Model part count	Number of model parts in the map

5.3.1 New header structure

Since the PC version, these are the values after the first five values of the header:

Offset	Type	Name	Description
0x0014	uint32	Model data file offset	This is the file offset to the vertex data
0x0018	uint32	Model part count	This is a repeat of the model part count
0x001C	uint32	Vertex data size	Size of the vertex data in bytes
0x0020	uint32	Model data size	Total size of the model data in bytes
0x0024	uint32	Magic	Typically equal to 1952540531 ("tags")

5.3.2 Xbox header structure

The Xbox version header uses pointers instead of file offsets since the model data is also in the tag data.

Offset	Type	Name	Description
0x0014	void*	Vertex data pointer	This is a pointer to the vertex data
0x0018	uint32	Model part count	This is a repeat of the model part count
0x001C	void*	Triangle data pointer	This is a pointer to the triangle data
0x0020	uint32	Magic	Typically equal to 1952540531 ("tags")

5.4 Tag

A tag is a singular asset used in a map. It is the fundamental building block used to make a map. Pretty much everything that makes a map a map is because of tags, and this is what makes the Halo engine so unique. Each tag in the tag array is comprised of this 32 byte structure:

Offset	Type	Name	Description
0x0000	uint32	Primary fourCC	Primary fourCC of the tag
0x0004	uint32	Secondary fourCC	Secondary fourCC, if present, or 0xFFFFFFFF
0x0008	uint32	Tertiary fourCC	Tertiary fourCC, if present, or 0xFFFFFFFF
0x000C	uint32	Tag ID	ID of the tag
0x0010	char*	Tag path pointer	Address of the null terminated path of the tag
0x0014	void*	Tag data pointer	Address of the tag data
0x0014	uint32	Resource index	Index of tag data if external (Custom Edition only)
0x0018	uint32	External	If 1, tag data is external (Custom Edition only)

You may notice there are two values at 0x0014. This is because Halo Custom Edition supports using external tags. This number is the index in the resource map in which the tag data is located, and on startup, Halo will load this tag into memory and change this to a valid pointer.

The only exception to this is sound tags which are, instead, matched by tag path. The reason for this is likely due to the fact that sound tags can reference other sound tags (i.e. promotional sounds), and doing this in a resource map in the current implementation isn't possible. When Halo finds an indexed sound tag, it will then set the pointer of the permutations array of the sound tag to the one loaded in memory. External data like this is copied directly into tag data on Halo Custom Edition.

BSP tags also work differently. The tag data pointer is not set to anything meaningful until the BSP data is actually loaded. This pointer is instead read from the scenario tag's structure BSP array. As mentioned previously, only one BSP can be loaded at a time. To make it seamless, BSP transitions are usually done in hallways which are duplicated between both BSPs. You can spot a BSP transition when the game says "Loading... done", or you may also notice the subtle change in lighting.

A common misconception is that, on Halo Custom Edition, nearly maxing out the tag space causes crashing, but tag space usage alone does not cause crashing. Rather, if any external tags do not fit the remaining tag space (which is possible if the map was compiled for a different set of resource maps than is being used by the user), the game will crash as it starts overwriting tag data with BSP data. For example, font tags with more characters can be megabytes larger than expected, as is common with languages that use a large set of characters such as Chinese. Playing a map that goes near the limit on the default font tags that, on an English version of the game, are only a few hundred kilobytes will surely crash on these games. In fact, even the Spanish version of the game is slightly larger than English. The only surefire way to prevent a crash from this is to not use external tags. This may come at the cost of a much larger map as well as no localization for strings, but the map will technically be more stable.