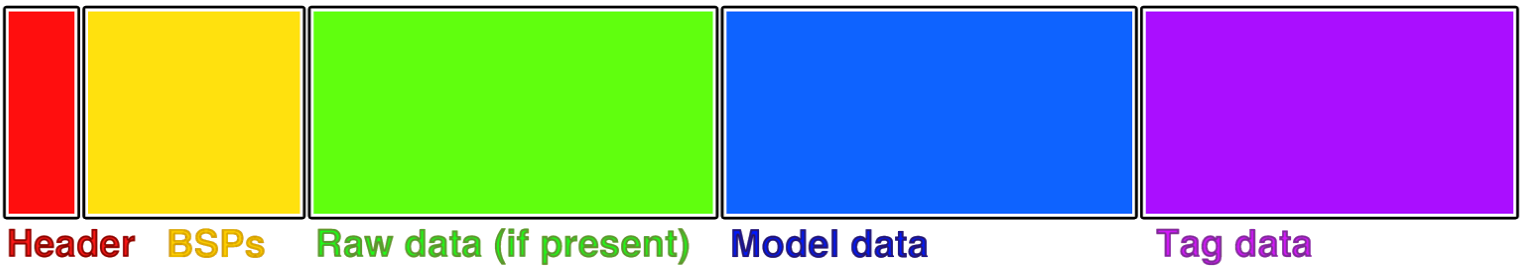
Halo's Map Files (Revision 1.2)

This guide describes Halo's cache files and resource map structure. I don't go over tag class structure here as this would not only take forever to come up with and I don't have time, but it'd end up being hundreds of pages.

Table of Contents

Cache Files

These files are referred to as cache files because contain data that is loaded directly into a location in Halo's memory with pointers and everything intact. All data is stored in this file in little endian even on the PowerPC version of Halo on the Mac.



This is a diagram of the cache file structure in the order output by tool.exe. All cache files start with a map header. This header is exactly 2048 bytes in length.

Directly after the map header are the BSPs of the map. At least one BSP is required for the map to be playable. This block is not directly referenced by any part of the map, but rather the file offsets are referenced in the scenario tag. The BSP tags in the tag index are, in fact, placeholders.

After BSPs are any internalized sound and bitmaps (raw data). Halo PC stores all stock bitmaps and sounds in bitmaps.map and sounds.map, so this section will not be present on stock maps. Like BSPs, the block itself is not directly referenced by any part of the map - Halo just uses file offsets referenced in the bitmaps and sound tags. Custom maps may have lightmaps stored here and any custom textures. In some custom maps, this can be the largest section, taking up hundreds of megabytes.

Right after raw data is models. Models are required for any model tags, and all stock maps store their maps here, but any maps that don't have model tags will not have this block. This block is directly referenced by the tag data header and is separated by vertices and indices.

While tool.exe outputs Halo files in this exact order, all that is required is that the map header to be at the beginning of the cache file. Everything else can be any order and as long as Halo can find everything it needs, the map will be functional.

Halo Custom Edition will calculate the CRC32 of the map file using the BSP data, model data, and tag data. When joining a Halo Custom Edition server, there is a challenge packet that can only be answered using the correct CRC32 value. Otherwise, the player will not be permitted into the game.

Header

All cache files contain a header that is 2048 bytes in length at the start of the map file. Most of this data is simply empty space that is ignored which can be used for storing arbitrary metadata, if desired.

On Xbox maps, everything after this is compressed using zlib compression which is the same algorithm used in .gz files. Otherwise, it is not compressed and can be read directly.

Offset	**Data Type**	**Usage**
0x0	uint32	Integrity - Must equal 0x68656164 (‘head’)
0x4	uint32	Game engine - 0x7 if retail, 0x261 if custom edition, 0x5 if Xbox
0x8	uint32	Map file size (decompressed) - Must be less than 0x38400000
0xC	char[4]	Unused
0x10	uint32	Tag data offset
0x14	uint32	Tag data size
0x18	char[8]	Unused
0x20	char[32]	Map name - Must be null terminated (encoding is ISO 8859-1 or Latin1)
0x40	char[32]	Map build - Unused; can be used to extend map name beyond 31 chars. (encoding is ISO 8859-1 or Latin1)
0x60	uint16	Map type - 0x0 if campaign, 0x1 if multiplayer, 0x2 if user interface
0x62	char[2]	Unused
0x64	uint32	~CRC32 of map’s BSP, model data, and tag data - Unused; Custom Edition recalculates it, anyway
0x68	char[1940]	Unused
0x7FC	uint32	Integrity - Must equal 0x666F6F74 (‘foot’)

Halo Demo Header

The demo version of the game uses a different order of data likely to prevent maps from the retail version of the game from being directly loaded. GameSpy shut down the demo version's multiplayer lobby a few years before GameSpy itself shut down, so it's essentially dead. Like with the standard header, any unused data can be used for arbitrary data, if desired. The demo version actually fills it up with random garbage data, anyway.

Offset	**Data Type**	**Usage**
0x0	char[2]	Unused
0x2	uint16	Map type - 0x0 if campaign, 0x1 if multiplayer, 0x2 if user interface
0x4	char[700]	Unused
0x2C0	uint32	Integrity - Must equal 0x45686564 ('Ehed')
0x2C4	uint32	Tag data size
0x2C8	char[32]	Map build - Unused (encoding is ISO 8859-1 or Latin1)
0x2E8	char[672]	Unused
0x588	uint32	Game engine - 0x6
0x58C	char[32]	Map name - Must be null terminated (encoding is ISO 8859-1 or Latin1)
0x5AC	char[4]	Unused
0x5B0	uint32	~CRC32 of map's BSP, model data, and tag data - Unused
0x5B4	char[52]	Unused
0x5E8	uint32	File size
0x5EC	uint32	Tag data offset
0x5F0	uint32	Integrity - Must equal 0x47666F74 ('Gfot')
0x5F4	char[524]	Unused

Tag Data

This block of data is normally the last section of a cache file. It and the BSPs are essentially the cached part of the cache file and are directly loaded into Halo's memory with little modification afterwards.

This data is always copied to 0x40440000 on Halo PC (0x4BF10000 on demo, 0x803A6000 on Xbox), so if you want to resolve a pointer into an offset from the beginning of this block, merely subtract the relevant address.

This is the structure of this header:

PC	Xbox	Data Type	Usage
0x0	0x0	Tag *	Tag array pointer - Halo caches this upon map load.
0x4	0x4	Tag ID	Principal scenario tag ID
0x8	0x8	uint32	Random number - Unused
0xC	0xC	uint32	Tag count (up to 65536 tags can be referenced)
0x10	0x10	uint32	Model part count
0x14 --	-- 0x14	uint32 void *	Model data file offset Unknown; model data address maybe?
0x18	0x18	uint32	Model part count
0x1C --	-- 0x1C	uint32 void *	Vertex size (everything after the vertices is indices) Unknown; index data address maybe?
0x20	--	uint32	Model data size
0x24	0x20	uint32	Equals 0x74616773 ('tags') - Unused

Tag IDs

Tag IDs are used for identifying tags in tag references. Tag IDs are read as 32-bit integers when checking if they are a null reference (0xFFFFFFFF) before being casted as a 16-bit integer. This means that up to 65536 tags can actually be referenced in a map, ranging from 0x0 to 0xFFFF.

Note that some references will not be checked if null. If they are null when they shouldn't, they will be treated as pointing to a 65536th tag which will crash the game.

Offset	Data Type	Usage
0x0	uint16	Tag index
0x2	uint16	Identifier - Unused outside of checking if null

Tag

The pointer to the tag array can be found in the header of the tag data block as well as the tag count. The address takes up the very first four bytes. It usually points to the data 0x28 (40) bytes after the beginning of the tag data block, or directly after the header, but this can be changed in some map protection schemes.

Each tag entry is 0x20 (32) bytes in size.

Offset	Data Type	Usage
0x0	uint32	Tag class
0x4	uint32	Secondary tag class - equals 0xFFFFFFFF if null - Unused
0x8	uint32	Tertiary tag class - equals 0xFFFFFFFF if null - Unused
0xC	Tag ID	Tag index
0x10	char *	Tag path (encoding is ISO 8859-1 or Latin1)
0x14	void * void * uint32_t	Tag data - Refer to scenario tag if SBSP tag If tag data is in resource maps and tag is a sound, then this points to a stub. Halo will retrieve the sound tag from sounds.map based on tag path. If tag data is in resource maps and tag is not a sound, then Halo will use this asset index.
0x18	uint32	Tag data is in resource maps. (CE only)
0x1C	char[4]	Unused

Tag Reference

Most tag classes reference other tags. For instance, a weapon has to have a model, a projectile, a first-person model, some animations, etc.

Offset	Data Type	Usage
0x0	uint32	Tag class
0x4	char *	Tag path - Unused
0x8	uint32	Unused
0xC	Tag ID	Tag index

Reflexive

These data structures contain pointers that point to other data in Halo's memory, and it also contains the number of objects in Halo's memory. It's usually within the tag data itself.

Offset	Data Type	Usage
0x0	uint32	Data count
0x4	void *	Data address
0x8	uint32	Unused

Legacy Parsing

This is the “old” way that’s used by older apps including Eschaton before this stuff was actually properly mapped out. It involved magic and other horrible nonsense, but it’s pretty interesting.

The tag array pointer was referred to as the “Primary Magic” of the map. The tag array was always assumed to be 0x28 bytes from the tag data block header (originally referred to as the map index). To get the “Map Magic” of the map, subtract the offset of the tag array from the tag data header (usually 0x28 or, if on Xbox, 0x24) from the primary magic, then subtract the offset to the tag data block (originally referred to as meta data). To convert a pointer to an offset, you can subtract this value from the pointer.

Using this method meant that the address of the tag data block (0x40440000 / 0x4BF10000) did not have to be stored, enabling lazy compatibility between both the demo and retail versions of the game with less parsing code (though the headers are still parsed different so it’s not perfect). However, it was less accurate and could be easily foiled using a hex editor if one wanted to protect their map files. Also, this method made it more complicated to do a bounds check to check if data was within the map file to prevent memory corruption.

Now that all of this information is available on cache files, using this method to calculate the offsets of pointers is not recommended. Instead, you can subtract the address of the tag block (equal to 0x40440000 on Halo PC) from the pointer and use this as an offset from the tag data block. It’s much simpler as well as more accurate to do it this way.

Resource Maps

These files are used to store asset data between maps in the maps folder and are also using the extension .map. They are not cache files, and they use completely different header structures.

They vary between versions of the game and are used to reduce space of map files. On Custom Edition, they are also used to store localization data, allowing for a CE map to work on all languages of the game, where on the standard retail version of the game, each map has to be tailored to its original language.

Resource maps aren't used in the Xbox version of the game. All assets are present in the cache files.

	Xbox	**Retail / Trial**	**Custom Edition**
Assets in cache files?	Yes	Optional	Optional
Present?	No	Yes	Yes
Maps	--	• bitmaps.map • sounds.map	• bitmaps.map • sounds.map • loc.map
Required?	--	No	Yes*
Holds tag data?	--	No	Yes
Platform independent	--	No	Usually
Bitmap naming scheme	--	<tag>_<bitmap>	<tag> <tag>__pixels
Sound naming scheme	--	<tag>__<range>__<permutation>	<tag> <tag>__permutations
Loc naming scheme	--	--	<tag>

* Halo will crash when a map other than the stock ui.map is loaded as these files contain necessary tag data in addition to the assets. There is a download for bitmaps.map and sounds.map with the asset data stripped (less than 1% the original sizes) available at this site: http://opencarnage.net/index.php?/topic/4834-ce-dedicated-server-bitmapsmap-and-soundsmap/

Header

The header of a resource map is much smaller than that of a cache file at only 16 bytes in length versus 2048 bytes. There's no vast amounts of unused data. This doesn't seem to be used in retail or demo versions of the game, as the maps themselves have offsets to data in the resource maps.

Offset	Data Type	Usage
0x0	uint32	Type (1 = bitmaps; 2 = sound; 3 = loc)
0x4	uint32	Resource paths offset
0x8	uint32	Resource index offset
0xC	uint32	Resource count

Resource Index

Each of these defines an asset or tag used.

Offset	Data Type	Usage
0x0	uint32	Resource path offset from tag paths offset
0x4	uint32	Resource size
0x8	uint32	Resource data offset

Data Types

These are some basic types of data. Most data types are basic C types or primitives.

Data Type	**Definition**
<X> *	This is a pointer that points to data in Halo's memory of type X. A pointer is the same size as an unsigned 32-bit integer.
char[<X>]	This is an array of X amount of characters.
tag	This specifies an asset in a Halo cache file, such as a weapon, model, bitmap, sound...
uint<X>	This is an unsigned X-bit integer.
void	This is any type of data with an unspecified number of bits in lengths.