Garmin Image (IMG) and UserData (ADM) Container File Format

Herbert Oppmann

herby@memotech.franken.de

http://www.memotech.franken.de/FileFormats/

2019-11-17

Content

Garmin Image (IMG) and UserData (ADM) Container File Format

The IMG and ADM Container File Format is used to combine a set of files into a single file. It is similar to a file system on a disk, but it lacks hierarchy. The contained files (called subfiles) form a flat list – there are no subdirectories. Therefore, all contained files must have unique file names.

This container format is used for image files (IMG) with filename extension *.img and for user data files (ADM) with filename extension *.adm.

IMG files contain all the info needed to render a map on Garmin GPS units.

ADM files can be created by Garmin HomePort [13]. The program will only offer this feature when it thinks that a marine GPS set is connected. Either the device needs to be connected via USB, or the SD Memory Card of the device needs to be plugged into a reader on the computer. (Actually, HomePort accepts any drive with a certain directory structure and device identification files on it as a marine GPS device.) The user data files are written into subdirectory Garmin\UserData.

The subfiles in an IMG or ADM container file are completely different:

IMG: DEM, GMP, IDX, LBL, MAR, MD2, MDR, MET, MPS, NET, NOD, QSI, RGN, S16, SNR, SRT, TRE, TRF, and TYP

ADM: PRX , RTE, TRK, and WPT

Therefore this document only covers the ADM/IMG container file format. For the subfiles, there are separate documents for IMG and ADM.

This documentation is based on own research and the sources listed in the references section.

Text highlighted like this is where further investigation is needed.

Basic data types

All values are serialized in little-endian byte order (least significant byte first).

Type	Length	Description
byte	1	8 bit unsigned integer (range 0 .. 255)
ushort	2	16 bit unsigned integer (range 0 .. 65535)
uint	4	32 bit unsigned integer (range 0 .. 4294967295)
ulong	8	64 bit unsigned integer (range 0 .. 18446744073709551615)
short	2	16 bit signed integer (range -32768 .. 32767)
int	4	32 bit signed integer (range -2147483648 .. 2147483647)
long	8	64 bit signed integer (range -9223372036854775808 .. 9223372036854775807)
char	1	A byte, which is a character code according to [14].

General file structure

Boot sector
Optional: One or more gap sectors, filled with 0x00
File table

Clusters of the contained files

The boot sector, the gap sectors, and the file table, together are treated also as a special type of file. It appears in the file table with a name consisting of all blanks (' '). In previous versions of this document, I named this file ADM for administration, but now that there are files with *.adm extension, this would be confusing. It is now named CTRL for control. All contained files, including CTRL, are managed in cluster granularity, which is a power of 2 times the sector size (512 byte). Within the CTRL file, all content (boot sector, gap sectors, file table and their entries) starts on sector boundaries.

Boot sector

This sector is similar to a Master Boot Record (MBR, see [10]).

Offset	Content
0x000-0x088	Header
0x089-0x1BD	unused, = 0
0x1BE-0x1CD	Partition Table Entry 1
0x1CE-0x1FE	Partition Table Entry 2-4, unused, = 0
0x1FE-0x1FF	MBR signature, = 0x55 0xAA

Header

Offset	Type	Content	
0x00	byte	XOR byte If this byte is <> 0, the IMG/ADM file is obfuscated. Each byte of the complete file must be XORed with this value for getting the plain data. In theory, the file always has to be XORed with this value, but a value of 0 will not change anything.	
0x01	byte[7]	= 0	
0x08	byte	Map version major	Seen values: V0.0, V2.0, V5.1, V11.2, V12.0, V14.0, V14.40, V17.0, V18.0, V20.15, V22.0, ...
0x09	byte	Map version minor	
0x0A	byte	Map update month, 1..12 (this is what HomePort uses, but some tools obviously use 0..11)	
0x0B	byte	Map update year MapEdit++ adds 1900 for val > 90, else 2000. The vast majority of dates > 2000 use values >100 here, only very few from the early 200x years use low numbers. To me, it seems that the low numbers came from some buggy implementation, and the rule is only there for compatibility to this bug. Most implementations just add 1900 regardless of the value.	
0x0C	ushort	=0	
0x0E	byte	Map source flag 0=Garmin map visible in Basecamp and Homeport 1=file created by Mapsource	
0x0F	byte	Byte checksum 0=not set otherwise: The (byte) sum of all bytes of the file, including the checksum byte, shall be 0.	
0x10	byte[6]	Signature1 “DSKIMG”, or “DSDIMG” for a demo map.	

Offset	Type	Content
		(Some sources mention “DSKDEM”, but if this is true, it is very old. Available Firmware Update files in GCD format contain the strings “DSKIMG”, “DS-DIMG” and “GARMIN” directly in sequence, but do not contain “DSKDEM”.)
0x16	ushort	Sector size, =512
0x18	ushort	Sector1. Values seen: 4, 8, 16, and 32
0x1A	ushort	Head1. Values seen: 16, 32, 64, 128, and 256
0x1C	ushort	Cylinder1. Values seen: 32, 256, 512, 1023, 1024, and also odd values
0x1E	ushort	=0
0x20	uint	=0
0x24	uint	uint checksum 0=not set otherwise: The (uint) sum of all bytes of the file (skipping the byte checksum at offset 0x0F and skipping this uint checksum). Is != 0 only on maps produced by Garmin.
0x28	uint	=0
0x2C	uint	=0
0x30	byte[9]	Part number (compressed), see [9]. The 9x8 bits have to be interpreted as 12 character codes with 6 bits each. Example: 10 D4 40 56 14 91 0D 14 41 Reverse bytes (shown in binary format with most significant bit (bit 7) first): 0100 0001 0001 0100 0000 1101 … Create groups of six bits each: 010000 010001 010000 001101 ... Add 0x20 = 0010 0000, this may overflow into the 7th bit: 00110000 00110001 00110000 00101101 ... Interpret bytes as ASCII code: '0' '1' '0' '-' ...
0x39	ushort	Creation year
0x3B	byte	Creation month, 1-12 (this is what HomePort uses, but some tools obviously use 0..11)
0x3C	byte	Creation day, 1-31
0x3D	byte	Creation hour, 0-23
0x3E	byte	Creation minute, 0-59
0x3F	byte	Creation second, 0-59
0x40	byte	Sector number where file table starts within CTRL file. Values seen: 2 and 8 (maybe depends on cluster size)
0x41	byte[7]	Signature 2 “GARMIN\0”
0x48	byte	=0, Maybe the Signature2 is one byte shorter and this is a ushort
0x49	char[20]	Map description, padded with space (0x20) chars ADM has “USERDATA” here
0x5D	ushort	Head2, == Head1
0x5F	ushort	Sector2, == Sector1
0x61	byte	Bytes per Sector exponent, =9 for 512 bytes per sector
0x62	byte	Sectors per Cluster exponent. Known values: 0..7 The cluster exponent must be chosen high enough so that the total number of clusters is < 65535.
0x63	ushort	Total number of cluster in IMG/ADM file. This should be file size / cluster size, but often, it is either one more or it is even the next power of 2.

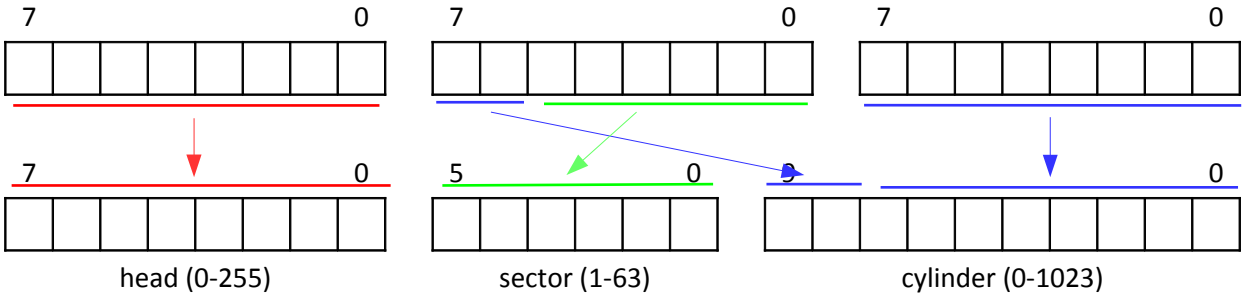
Offset	Type	Content	
0x65	char[30]	Map description, continued, padded with space (0x20) chars	
0x83	ushort	Magic value to tell if the next three fields are valid. 0: not valid (field values shall then be 0) 0xDEAD: valid Maybe the following fields are describing the version of the software that wrote this file?	
0x85	byte	Version major	Seen values: V0.0 when previous field is 0, else: V2.2, V3.0, V3.5, V3.6, V3.7, V3.9, V3.11
0x86	byte	Version minor	
0x87	ushort	More version info? Known values: 0, 9, 0x0056, 0x00A5, 0x00E5, 0x0228 and 0x05B2	

BLUCHART maps have something different in their checksum calculation ...

Partition Table Entry 1

Offset	Type	Content
0x1BE	byte	Status: 0x00=non-bootable, 0x80=bootable, =0
0x1BF	CHS	CHS address of first absolute sector in partition, = 0/1/0
0x1C2	byte	Partition type, =0x00
0x1C3	CHS	CHS address of last absolute sector in partition Cylinder * Head1 * Sector1 + Head * Sector1 + Sector == Number of sectors in partition Actually, this number is nonsense, because on a real disk you don't know Head1 and Sector1, and it is the Cylinder x Head x Sector which defines the size of the disk. I.e. Cylinder is the last usable cylinder, Head is the last usable head, and Sector is the last usable sector, and the partition always covers the full array.
0x1C6	uint	LBA of first absolute sector in partition = 0
0x1CA	uint	Number of sectors in partition. == Cluster * 2 ^ sectors per Cluster exponent.

CHS



File Table

Note: The file table starts on a sector boundary, not necessarily on a cluster boundary. Most of the times it ends at a cluster boundary, having unused entries at the end if needed. But sometimes it has less entries and there is unused space in the last cluster.

File Table Entry

Each entry in the file table is one sector in size.

Offset	Type	Content
0x000	byte	Flag: 0=unused, 1=used
0x001	char[8]	Filename, padded with space (0x20) chars
0x009	char[3]	Filename extension, padded with space (0x20) chars
0x00C	uint	File size in byte
0x010	ushort	Extent number mkgmap says: 1 byte flag “special”und 1 byte part
0x012	byte[14]	Unused, usually =0x00, but sometimes =0xFF
0x020	ushort[240]	List of Cluster numbers. The first cluster in the file is 0. Unused entries (after the end of the file) are 0xFFFF. The list must be contiguous (there may be no holes in it). The entries are really unsigned short (value 50888 has been seen).

Big files may occupy more than 240 clusters. In this case, there are multiple file table entries for the same file. Usually, all entries for one file are in consecutive order in the file table, but this can’t be relied on. The first entry has extent number 0, the second entry has extent number 1, and so on.

All file table entries for the extents have correct flag and file name, but the file size is set only in the first extent. File size is 0 in the following extents.

The file table entries for the CTRL file are special: The file name and extension is all blanks (0x20). The Extent number of the first extent is 0x0001 for ADM and 0x0003 for IMG, for the next it is 0x0100, 0x0200, and so on. The size must be a multiple of the sector size.

For the CTRL file, the file size must be a multiple of the sector size (512 byte) and should be a multiple of the cluster size, but sometimes, it is smaller. In this case, the remaining place up to the end of the cluster is filled with 0xFF.

How to start?

OK. After we have read in the CTRL file, we know which files are present, their names, how long they are, and where their clusters are. But wait a minute - the CTRL file is itself a file which is described in the CTRL file. So this leads to a chicken and egg problem!

This is solved by the following rules:

- The CTRL file always starts with the first cluster (cluster number 0). So the very first bytes of the IMG/ADM file are the Header, where we can find out the cluster size and the starting sector of the file table. This linear mapping means that CTRL file offsets are identical to IMG/ADM file offsets.
- If the file table does not start within the first cluster, all clusters of the CTRL file up to and including the first file table entry must be in consecutive order within the IMG/ADM file, i.e. the linear mapping continues. This ensures we can read the first file table entry without knowing the file table in advance.
- The file table entry for the first extent of the CTRL file is the very first entry within the file table. This ensures that after we have read the cluster that contains the beginning of the file table, we know at least the first 240 cluster of the CTRL file. In case the CTRL file is so big that it uses more than one ex-

tent, the file table entry for the next extent must be within the first 240 clusters of the CTRL file, and so on. This ensures we can step by step determine the CTRL file.

In reality, things are much easier:

Usually the CTRL file starts with the beginning of the IMG/ADM file and runs contiguously until its end. After that, the first payload file starts and runs contiguously until its end, and so on. So in practice, you will only see consecutive cluster numbers within the file table entries.

So if you want to cut down on effort, you may take some shortcuts in the dissection logic here. You could access the payload file data with just knowing the length and a single offset for each file.

References

Used sources of information

[1] libgarmin http://libgarmin.sourceforge.net/

[2] mkgmap http://www.mkgmap.org.uk/

[3] GPSBabel https://www.gpsbabel.org/

[4] Garmin IMG Format https://sourceforge.net/projects/garmin-img/

[5] http://pinns.co.uk/osm/

[6] http://wiki.openstreetmap.org/wiki/OSM_Map_On_Garmin/IMG_File_Format

[7] http://whiter.brinkster.net/raster_img.shtml

[8] https://www.gpspower.net/garmin-discussions/224115-lock-img-file.html

[9] https://www.gpspower.net/garmin-maps/266563-garmin-transalpin-2012-pro-10.html

[10] https://en.wikipedia.org/wiki/Master_boot_record

[11] Decoding Proprietary Marine Track Files https://github.com/rahra/parsefsh

[12] Perl script which converts user data to industry-standard GPX records https://sourceforge.net/projects/adm2gpx/

[13] Garmin HomePort https://www8.garmin.com/support/download_details.jsp?id=7263

Standards and specifications

[14] ISO/IEC 8859-1:1998 Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1

Sources of sample files

[15] http://wiki.openstreetmap.org/wiki/OSM_Map_On_Garmin/Download

[16] http://www.garniak.pl/viewtopic.php?t=398

[17] https://www.digital-eliteboard.com/forums/seekarten.976/?prefix_id=22

[18] http://www.gpsfiledepot.com/maps/view/518/

[19] http://www.zinnware.com/HighAdv/BucktailPath_200305/GPS_tracking_download.html

[20] http://www.switchbacks.com/nwtrails/

[21] http://www.island-olaf.de/travel/marokko/gps.html

[22] http://www.elgps.com/mapas.html

[23] http://www.pltopo.pl/podglad-mapy/

[24] http://www.garda-gps.de/dl_pejo.html