

# **Garmin Points of Interest (POI)**

## **File Format**

Herbert Oppmann

[herby@memotech.franken.de](mailto:herby@memotech.franken.de)

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

2023-08-21

## Content

Garmin Points of Interest (POI) File Format .....	3
Basic data types .....	3
General file structure .....	5
Obfuscation .....	5
General record structure .....	6
Sequence of records on top level .....	7
Record type 0: Header1 .....	7
Record type 1: Header2 .....	8
Record type 2: Waypoint .....	9
Record type 3: Alert .....	9
Record type 4: Bitmap reference .....	10
Record type 5: Bitmap .....	10
Record type 6: Category reference .....	11
Record type 7: Category .....	11
Record type 8: Area .....	11
Record type 9: POI group .....	12
Record type 10: Comment .....	12
Record type 11: Address .....	12
Record type 12: Contact .....	13
Record type 13: Image .....	13
Record type 14: Description .....	14
Record type 18: Media .....	14
Record type 0xFFFF: End .....	14
Record type 15: Product info .....	15
Record type 16: Alert circles .....	15
Record type 17: Copyright .....	16
Record type 19: Safety camera .....	17
Record type 21: Additional data .....	18
Record type 23: ? .....	22
Record type 24: ? .....	23
Record type 26: ? .....	23
Record type 27: Alert trigger options .....	23
References .....	26
Used sources of information .....	26
Standards and specifications .....	26
Sources of sample files .....	26

# Garmin Points of Interest (POI) File Format

Filename extension \*.gpi

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

## Basic data types

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

Type	Length	Description
char	1	ASCII character (see [10])
byte	1	8 bit unsigned integer (range 0 .. 255)
ushort	2	16 bit unsigned integer (range 0 .. 65535)
int	4	32 bit signed integer (range -2147483648 .. 2147483647)
uint	4	32 bit unsigned integer (range 0 .. 4294967295)
coord24	3	A 24 bit integer which has to be multiplied by 360.0° and divided by $2^{24}$ to get a longitude or latitude coordinate.
coord32	4	A 32 bit integer which has to be multiplied by 360.0° and divided by $2^{32}$ to get a longitude or latitude coordinate.

### Version:

A byte or ushort where the value is calculated as major version x 100 + minor Version.

E.g. 0x64 = 100 = V1.0

### GDate:

An uint, where the values 0 or 0xFFFFFFFF both mean "no value given". All other values have to be interpreted as unix time (see [4]), but with a starting value (epoch) of 1989-12-31 instead of 1970-01-01.

Note: While the number counts seconds, unix time is not defined as "number of seconds since the epoch", but "number of days since the epoch" x 24 x 60 x 60 + "number of seconds since midnight". The difference between the two is that unix time ignores leap seconds. The difference between unix time and UTC is already more than 20 seconds.

### PString:

Pascal-like string.

Type	Content
ushort	Length. This is the number of bytes following this length field. The string may be empty, in which case this length is 0.
byte[n]	The characters of the string, encoded according to the CodePage. The string is not zero-terminated.

### LString:

Multi-language string.

Type	Content
uint	Length. This is the number of bytes following this length field. The list of localized strings may be empty, in which case this length is 0.

Type	Content
Repeat until length bytes consumed:	
char[2]	Two ASCII chars locale according to [10]. Within one LString, the locales must be unique. Seen values: CS, DA, DE, EN, ES, FI, FR, IT, NL, NO, PL, PT, RU and SV
PString	Localized string

**VarInt:**

An unsigned integer in a variable-sized encoding

Using 1 byte: (probably, but not seen yet)

	7	6	5	4	3	2	1	0
0	6	5	4	3	2	1	0	=1

Contains 7 bit. Used for numbers in the range 0 .. 0x7F.

Using 2 bytes:

	7	6	5	4	3	2	1	0
0	5	4	3	2	1	0	=1	=0

	7	6	5	4	3	2	1	0
1	13	12	11	10	9	8	7	6

Contains 14 bit. Used for numbers in the range 0x0080 .. 0x3FFF.

Using 3 bytes:

	7	6	5	4	3	2	1	0
0	4	3	2	1	0	=1	=0	=0

	7	6	5	4	3	2	1	0
1	12	11	10	9	8	7	6	5

	7	6	5	4	3	2	1	0
2	20	19	18	17	16	15	14	13

Contains 21 bit. Used for numbers in the range 0x004000 .. 0x1FFFFF.

Using 4 bytes: (probably, but not seen yet)

	7	6	5	4	3	2	1	0
0	4	3	2	1	0	=0	=0	=0

	7	6	5	4	3	2	1	0
1	12	11	10	9	8	7	6	5

	7	6	5	4	3	2	1	0
2	20	19	18	17	16	15	14	13

	7	6	5	4	3	2	1	0
3	28	27	26	25	24	23	22	21

Contains 29 bit. Used for numbers in the range 0x00200000 .. 0x1FFFFFFF.

Note: The encoding does not exclude smaller numbers to be represented in more bytes than is necessary. But this is normally not done and therefore indicates some problem.

## General file structure

The file consists of a list of variable-size records immediately following each other. The list is terminated by an End record. Records may again contain lists of subordinated records.

FormatVersion '01' only: The presence of the Additions record indicates that there are additional data after the End record. These additional data are not organized as list of records. The content and structure of the additional data is determined by the Additions record, see below. The additional data usually follow the End record immediately, but sometimes there is a small unused area (filled with 0x00 bytes) between it. When the list of records is obfuscated, also these unused bytes are obfuscated.

## Obfuscation

Obfuscation is optional, and seen in FormatVersion '01' only. The Header1 record contains a field Obfuscation which tells, whether obfuscation is used, and over which parts of the file. When obfuscation is used it always starts after the Header1 and Header2 records. The difference is where it ends. With Obfuscation=3, obfuscation stops at the offset given in field "Start 61" of the Additional data record. With Obfuscation=5, also the additional data after the End record are obfuscated, which means obfuscation runs until the end of file.

Starting with the first byte of the first obfuscated data, consecutive groups of 4 bytes are processed. Each byte within such a 4-byte group is processed individually. The first byte is processed with constant 0x48, the second with 0x06, the third with 0xB3, and the last with 0x00. A byte consists of two nibbles (groups of four bits). The operation here is a subtraction which is done individually for each nibble. One can also say, a byte-wide subtraction without half-carry, i.e. without overflow from the low-nibble to the high-nibble.

Example: End of the Header1 record (or its subordinated record in its extra data) and start of the first record which is obfuscated (in red), which is here the Additions record.

In file:	61	73	6F	6E	21	20	5D	06	B3	00	72	06	B3	00	70	06	
							-	48	06	B3	00	48	06	B3	00	48	06
Clear text:	61	73	6F	6E	21	20	15	00	00	00	3A	00	00	00	38	00	

Here 0x15 0x00 is the record type, 0x00 0x00 is the record flags, 0x3A 0x00 0x00 0x00 is the record length, and so on.

The bytes with the orange background are where a normal byte-wide subtraction would have a different result than this nibble-wise subtraction. Instead of subtracting, it is also possible to add (nibble-wise, without half-carry) the two's-complement of the constant byte values, which are 0xB8 0xFA 0x4D 0x00.

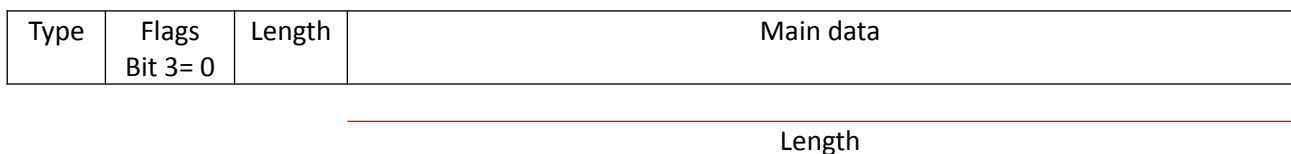
It does not matter whether the start of the obfuscated data is on a 4-byte-boundary or not. The 4-byte-groups just start with the first obfuscated byte and run through the whole obfuscated data without being

re-started, e.g. on a record boundary. Obfuscation stops with the last byte that should be obfuscated, even if we are in the middle of a 4-byte group. There is no padding to make the size a multiple of 4.

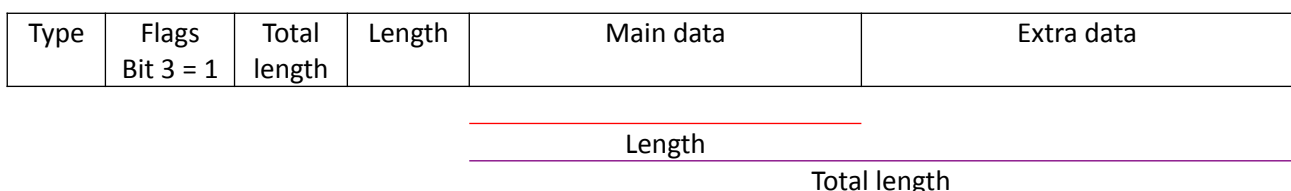
## General record structure

Type	Content
ushort	Type Identifies the record content. See table below.
ushort	Flags, seen values: 0, 8 and 0x18 Bit 3: Extra data is present Bit 4: Only seen in FileFormat '01' on Header2 and POI group records. This flag is set on Header2 if and only if at least one of the POI groups has this flag also set. This flag is set on a POI group if and only if it contains a type 23 and a type 24 record.
uint	If extra data present: Total data length in byte. This is the length of the main data plus the extra data.
uint	Length of main data in byte. Must be <= Total data length.
byte[Main data length]	Main data
byte[Total data length - main data length]	If extra data present: Extra data

Layout without extra data:



Layout with extra data:



Records contain main data and may optionally contain extra data. Main data is usually used for a structure (a sequence of fields with specific data type and meaning), but sometimes for a list of subordinated records. Extra data is usually used for a list of subordinated records, but sometimes for embedded media. Lists of subordinated records stop at the end of the main or extra data area. They are not terminated by the end record.

Some record types appear only as subordinated records within other records. But independent of where a record appears, the record type always has the same semantic. So the record content can be described with a single list with the record type as key.

Records seen in all FormatVersions:

Value	Meaning
0	Header1
1	Header2

Value	Meaning
2	Waypoint
3	Alert
4	Bitmap reference
5	Bitmap
6	Category reference
7	Category
8	Area
9	POI Group
10	Comment
11	Address
12	Contact
13	Image
14	Description
0xFFFF	End

Records seen in FormatVersion '01' only:

Value	Meaning
15	Product info
16	Alert circles
17	Copyright
18	Media
19	Safety camera
21	Additions
23	
24	
26	
27	Alert trigger options

## Sequence of records on top level

FormatVersion '00':

Header1 Header2 POI group\* End

FormatVersion '01':

Header1 Header2 [Additions] POI group\* End [Additional data]

[] = optional, \* = zero or more occurrences

## Record type 0: Header1

Main data: Length is at least 16.

Type	Content
char[6]	'GRMREC'
char[2]	FormatVersion: Two decimal digits. Values seen: '00', '01'
GDate	TimeStamp
byte	Flags1.

Type	Content
	<p>Seen values: 0, 1 in 3D_Bogenschiessen.gpi, 3 in Angelgeschaefte.gpi and 5 in AustriaGermanyCyclopsSample.gpi</p> <p>Whether Product info record follows is independent of the bits here.</p> <p>E.g. in Caravanhaendler.gpi, it is 3 and there are three 0x00 byte between the PString length and the first character?!</p> <p>Is always 0 in FormatVersion '00'</p> <p>Bit 0: TimeStamp valid?</p> <p>Bit 1: ?</p> <p>Bit 2: ?</p> <p>Bit 7-3: =0</p>
byte	<p>Obfuscation</p> <p>0 = no obfuscation</p> <p>3 = Obfuscation starts after Header1 and Header2 records and stops at the offset given in field "Start 61" of the Additional data record</p> <p>5 = Obfuscation starts after Header1 and Header2 records and stops at end of file</p> <p>Is always 0 in FormatVersion '00'</p>
PString	<p>Name</p> <p>Several examples from [16] have three bytes 0x00 between the string length and the actual string bytes, which are not counted by the string length?!? When this happens, the Flags1 byte has a value of 3, but not vice-versa. I assume this is a bug in the program generating the file.</p>

Extra data is sequence of records.

For FormatVersion '01':

[Product info]

## Record type 1: Header2

Main data: Length is 12.

Type	Content
char[6]	'POI', 0, 0, 0
char[2]	<p>FormatVersion: Two decimal digits.</p> <p>Same as in record Header1</p>
ushort	<p>CodePage, (see [5] and [6])</p> <p>Seen values:</p> <p>0x036A Thai (Windows)</p> <p>0x03B6 Chinese Traditional (Big5)</p> <p>0x04E2 Central European (Windows)</p> <p>0x04E3 Cyrillic (Windows)</p> <p>0x04E4 Western European (Windows)</p> <p>0xFDE9 Unicode (UTF-8)</p>
ushort	<p>Seen values</p> <p>if FormatVersion =0: 0</p> <p>if FormatVersion =1: 17</p> <p>(Note: 17 is the code for the copyright record contained in the extra data.)</p>

Extra data is sequence of records.



For FormatVersion '01':  
[Copyright]

## Record type 2: Waypoint

Main data: Length is at least 15.

Type	Content
coord32	Latitude
coord32	Longitude
ushort	Unknown4. =1
byte	Unknown5. Seen values 0, 1, 2 (3D_Bogenschiessen.gpi), 3 (dorognoe_radio.gpi), 0x12 (bashneft_hi_res.gpi) Could also be flags. Bit 0 could be "Alert record in extra data" Bit 1 ? Bit 3-2: =0 Bit 4 ? Bit 7-5: =0
LString	Shortname

Extra data is sequence of records.

For FormatVersion '00':  
[Category reference] [Bitmap reference] [Alert] [Comment] [Address] [Contact] Image\* [Description]

For FormatVersion '01':  
[Category reference] [Bitmap reference] [Alert] [Comment] [Address] [Contact] Image\* [Description]  
[26]

## Record type 3: Alert

Speed, proximity and other stuff.

Main data: Length is 12.

Type	Content
ushort	Proximity in meter
ushort	Speed in 100 x meters / second, 0 = none
ushort	Unknown6. Seen values 0 (dorognoe_radio.gpi) and 0x100
ushort	Unknown7. Seen values 0 (dorognoe_radio.gpi) and 0x100
byte	Alert (0=Off, 1=On) (see [7]) Seen value 1
byte	Type of alert: 0=proximity (360° alert), 1=along road, 2=TourGuide (see [7]) Seen values 1 and 2 (in 2011_DOC_Camping_Sites_Update.gpi)
byte	Sound number (see [7]) In case Audio alert below is 0x00: seen values 4..105 could this be a symbol number? else if Audio alert below is "predefined audio": 0=beep, 1=tone, 2=3 beep, 3=silence, 4=plung, 5=double plung Seen values: 4 and 5

Type	Content
	Else (custom audio): 1..n
byte	Audio alert 0x00, 0x01, 0x02, 0x03, 0x06, 0x07, 0x09, 0x0A, 0x0B, 0x0C, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x3A, 0x3B, 0x3E, 0x76, 0x77, 0x78, 0x79, 0x7A, 0x10 for predefined audio, 0x20 for custom audio (in media record)

Extra data is a sequence of records.

For FormatVersion '01':

[Alert circles] [Alert trigger options]

## Record type 4: Bitmap reference

Main data: Length is 2 or 4.

Type	Content
ushort	Bitmap ID Points to the Bitmap record with this ID
ushort	Unknown8. Optional, =2
3 x ushort	Optional, =3, 4, 5

No extra data.

## Record type 5: Bitmap

Main data: Length is 36.

Type	Content
ushort	Bitmap ID, starting with 0
ushort	Height in pixel
ushort	Width in pixel
ushort	Line size in byte Is calculated as: (Width in pixel x Bits per pixel) / 8 bits per byte, rounded up.
ushort	Bits per pixel
ushort	Unknown9. =0
uint	Image size in byte Is calculated as: Line size in byte x Height in pixel
uint	Unknown10. Seen value 44
uint	Number of color palette entries Seen values: 0 (no palette), 16 (then bits per pixel must be 4) and 256 (then bits per pixel must be 8)
uint	Transparent color Seen values 0 = Black and 0xFF00FF = Magenta
uint	Flags2, seen value 0, 1 and 0x0100 (D0743030F.gpi) Bit 0 (mask 0x0001): enable transparent mode? According to [2] Bit 8 (mask 0x0100): Transparency (alpha channel) present The combination of bit 0 set and bit 8 set has not been seen, so it is unclear whether this is allowed, and if so, in which order the dependent data would come.
uint	Unknown11. Seen values: 0 and Image size + 44

Type	Content
Image size x byte	Pixel
Number of color palette entries x uint	If number of color palette entries >0: Color palette for 4 or 8 bit per pixel  There is one file where the palette is shorter than indicated in “number of color palette entries”, but it is big enough for the highest Pixel entries.
Height x Width byte	If Flags2.bit8 is set: Seems to be transparency (alpha channel) when there is no palette. One byte for each pixel. Values seen: 0 - 128
n x byte	=0 Flags2.bit0 is set when this is present, but not the other way round n is up to Image size, but sometimes less

No extra data.

## Record type 6: Category reference

Main data: Length is 2.

Type	Content
ushort	Category ID Points to the Category record with this ID

No extra data.

## Record type 7: Category

Main data: Length is at least 6.

Type	Content
ushort	Category ID, starting with 0
LString	Category

Extra data is sequence of records.

For FormatVersion '01':  
[Bitmap reference]

## Record type 8: Area

Main data: Length is 23.

Type	Content
coord32	Max. latitude
coord32	Max. longitude
coord32	Min. latitude
coord32	Min. longitude
uint	Unknown12. =0
ushort	Unknown13. =1

Type	Content
byte	Unknown14. Seen values 0, 1, 2 (3D_Bogenschiessen.gpi), 3 (dorognoe_radio.gpi), 5 (2011_DOC_Camping_Sites_Update.gpi), 8 and 18 (spb_metro_norm.gpi)

Extra data is sequence of records.

For FormatVersion '00':

Area\* Waypoint\*

For FormatVersion '01':

Area\* (Waypoint\* | Cyclops data\*)

## Record type 9: POI group

Main data: Length is at least 4.

Type	Content
LString	Data source
n x Record	List of area records (type 8).

Extra data is sequence of records.

For FormatVersion '00':

Category\* Bitmap\* Media\*

For FormatVersion '01':

Category\* Bitmap\* Media\* [23 24]

## Record type 10: Comment

Main data: Length is at least 4.

Type	Content
LString	Comment

No extra data.

## Record type 11: Address

Main data:

If FileVersion == '00': Length is 2.

If FileVersion == '01': Length is at least 2.

Type	Content
ushort	Flags: Bit 0: City present Bit 1: Country present Bit 2: State present Bit 3: Postal code present Bit 4: Street present

Type	Content
	Bit 5: Street address present Bit 15-6: =0

The following fields are only present when the respective bits in the Flags are set. In FileVersion '00', the strings are in the extra data, while in FileVersion '01', the strings are in the main data and nothing is in the extra data.

Type	Content
LString	City
LString	Country
LString	State
PString	Postal code
LString	Street
PString	House number

## Record type 12: Contact

Main data:

If FileVersion == '00': Length is 2.

If FileVersion == '01': Length is at least 2.

Type	Content
ushort	Flags: Bit 0: Phone present Bit 1: Phone2 present Bit 2: Fax present Bit 3: Email present Bit 4: Link present Bit 5: Comment? present Bit 15-6: =0

The following fields are only present when the respective bits in the Flags are set. In FileVersion '00', the strings are in the extra data, while in FileVersion '01', the strings are in the main data and nothing is in the extra data.

Type	Content
PString	Phone
PString	Phone2
PString	Fax
PString	Email
PString	Link
LString	Comment?

## Record type 13: Image

Main data: Length is at least 5.

Type	Content
byte	Unknown15. =0
uint	Length of image
byte[Length of image]	Image Content (Bitmap or JFIF = JPEG Interchange File Format)

No extra data.

## Record type 14: Description

Main data: Length is at least 5.

Type	Content
byte	Unknown16. Seen values: 1, 5, 50
LString	Description

No extra data.

## Record type 18: Media

E.g. sound in MP3 format, pictures in JFIF (JPEG interchange file format).

Main data: Length is 3.

Type	Content
ushort	Media ID
byte	Media format: 0 = WAV, 1 = MP3 (see [8])

Extra data:

Type	Content
uint	Total length (including this length field!)
Repeated until total length reached:	
char[2]	Locale
uint	Length of media for this locale
byte[Length of media]	Media Content

## Record type 0xFFFF: End

Record flags is 0.

Main data: Length is 0, i.e. this record has no content.

No extra data.

**The following records are only useable with FormatVersion '01'**

## Record type 15: Product info

Main data: Length is 5 or 6.

Type	Content
ushort	FID (Family ID a.k.a. MapId) 0xFFFF = none
byte	PID (Product ID) 0xFF = none
byte	RgnID (Region ID) 0 = none 1 = United Kingdom + Ireland 2 = Netherlands 3 = France 4 = Belgium + Luxemburg 5 = Australia + New Zealand 6 = Spain + Portugal 7 = Italy + Slovenia 8 = Austria + Germany 10 = Nordics 14 = Eastern Europe 26 = Greece 27 = USA + Canada 28 = Russia 29 = South Africa 32 = Middle East 0xFF = none
byte	VenID (Vendor ID) Seen values: 0 (none), 2, 3, 11, 24, 27, 28, 30, 36 and 43 At least for speed cameras this is used like a release number
byte	Unknown17. Optional, seen values: 0xE9 and 0xEE

No extra data.

## Record type 16: Alert circles

Coordinates are the center of a circle and uint value is radius in meters. Alert will be triggered when crossing the circles.

Main data: Length is  $2 + n * 12$ .

Type	Content
ushort	Number of alert circles following (>0)
Repeat:	
coord32	Latitude
coord32	Longitude
uint	Radius in meters

No extra data.

## Record type 17: Copyright

Main data: Length is at least 24.

Type	Content
ushort	Flags1. Seen values: 0x0004, 0x0014, 0x0402, 0x0404, 0x0480 and 0x0481 Bit 0 (mask 0x0001): ? Bit 1 (mask 0x0002): File contains type 23/24 records (must match Flags.bit4 on Header2) Bit 2 (mask 0x0004): ? Bit 4 (mask 0x0010): Date is present in this record Bit 7 (mask 0x0080): File contains cyclops records Bit 10 (mask 0x0400): Device model string is present in this record
ushort	Flags2. Seen values: 0x0000, 0x0080, 0x00A0, 0x01A0, 0x0880 and 0x8880 Bit 5 (mask 0x0020): File contains type 26 records, an Additions record, and additional data containing a waypoint index Bit 7 (mask 0x0080): ? Bit 8 (mask 0x0100): Pictures and LString present in this record Bit 11 (mask 0x0800): File contains an Additions record and additional data of yet unknown format Bit 15 (mask 0x8000): Unknown uint is present in this record
ushort	Unknown24. Seen values: 0, 10, 20 and 25
ushort	Unknown25. Seen values: 0, 2, 3, 5, 9, 11, 22, 24, 27 and 36
LString	Data source
LString	Copyright (may be empty)

Optional, present if Flags1.bit10 is set:

Type	Content
PString	Device model

Optional, present if Flags2.bit8 is set:

Type	Content
12 Pictures, each consisting of a picture header and the picture itself:	
Only present on 1 <sup>st</sup> and 3 <sup>rd</sup> picture:	
VarInt	Some length On first picture: Could be Length of complete optional block On third picture: Could be length of remaining 10 pictures
byte	Unknown26. Seen values: On first picture: 23 – ? On third picture: 10 – could be number of remaining pictures
byte	Unknown27. =0
VarInt	Length of picture entry in byte, i.e. length of remaining header bytes plus the picture itself
byte	Unknown28. =0
ushort	Width in pixel
ushort	Height in pixel
VarInt	Length of picture in byte
byte[]	Picture in JFIF (JPEG interchange file format)
LString	Unknown29



Optional, present if Flags1.bit4 is set:

Some date – what for?

Type	Content
byte	Month
byte	Day
ushort	Year

Optional, present if Flags2.bit15 is set:

Type	Content
uint	=2

No extra data.

## Record type 19: Safety camera

Cyclops data, see <https://www.cyclops-uk.com/>.

Main data: Length is at least 26.

Type	Content
coord24	Max. latitude
coord24	Max. longitude
coord24	Min. latitude
coord24	Min. longitude
byte	Unknown31. Defines the structure of the following bytes. Seen values: <ul style="list-style-type: none"> <li>0x00, 0x03, 0x0f, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1f, 0x20, 0x21, 0x22, 0x23, 0x24, 0x27, 0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2f, 0x30, 0x31, 0x32, 0x33, 0x37, 0x38, 0x39, 0x3a, 0x3b</li> <li>0x81</li> <li>0x80, 0x82, 0x83, 0x85, 0x8b, 0x8c, 0x8d, 0x8f, 0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x97, 0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9f, 0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa7, 0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xaf, 0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb6, 0xb7, 0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xc0, 0xc1, 0xc2, 0xc3, 0xc8, 0xc9, 0xca, 0xcb, 0xd0, 0xd1, 0xd2, 0xd3, 0xd8, 0xd9, 0xda, 0xdb, 0xe0, 0xe1, 0xe2, 0xe3, 0xe8, 0xe9, 0xea, 0xeb, 0xf0, 0xf3</li> </ul> Seems to be: Bit 7-3: Type (or maybe only Bit 7-4: Type and bit 3 is also a flag) Bit 2-0: Flags?
byte[11]	Optional, if Unknown31 == 0x81 First byte seen: 2, 3, 10, 11 Remaining bytes: 00 00 00 55 55 15 c7 71 dc 01 (Notice that a three-nibble-value repeats: 0xc71 0xc71) Instead of first byte being 0x81, the condition could also be: 1) If bit 7 of first byte is set, a second byte follows 2) If second byte is 2, 3, 10 or 11 (or if bit 1 of the second byte is set), then this sequence follows, otherwise not.
byte	Optional, if Unknown31 is 0x80 or is > 0x81

Type	Content
	Seen values 1, 8, 9
byte	Number of 10-bit values following. 0 means one value following, 1 means two values following ...
byte	Optional, if Unknown31 == 0x81 = 10 (bit-length of each value)
byte[]	List of 10-bit values (compacted). The least significant bits of each 10 bit value come first. Unused bits are in the most significant bits of the last byte and are 0. Example: 21 85 24 32 In bits: 0010 0001 1000 0101 0010 0100 0011 0010 The three 10-bit-values are: 0100100001 0100100001 1100100010
coord24	Latitude
coord24	Longitude
byte[n]	Unknown area. n >= 3, up to 175 seen

Observed patterns for unknown area: Not yet known

No extra data.

## Record type 21: Additional data

The presence of this record indicates that additional data are following after the end record.

This record is only present if at least one of the following flags are set: Copyright record Flags2.bit5 (waypoint index present) and Flags2.bit11 (additional data present). Main data length is > 58 if and only if Flags2.bit11 is set.

(In the following, UAData is short for Unknown additional data.)

Main data:

Length: 58

Type	Content
ushort	Length until end of this record, i.e. it is the record length minus 2
uint	Offset of the index table (from beginning of file) Is 0 if waypoint index not present.
byte[32]	=0
uint	Offset of the index table (from beginning of file) Is 0 if waypoint index not present.
uint	Total length of the index table in byte (the first entry in the index table is the length in byte of the following entries, so this value here is the same, but +4 for the first entry itself). Is 0 if waypoint index not present.
uint	Unknown36. =1 Is 0 if waypoint index not present.
uint	Unknown37. =0
uint	Unknown38. =0

Length: 146

Type	Content
byte[56]	=0
uint	Unknown60. =7
uint	Start offset of UADData2 (= Start offset of UADData1 + Length of UADData1)
uint	Start offset of UADData1
uint	Length in byte of UADData1
byte[16]	=0

Length: 182

Type	Content
uint	Unknown61 Seen values 0, 7, 14 With values 0 and 7, all remaining fields in this block were 0.
byte[16]	=0
uint	Start offset of UADData3
uint	Unknown62. =1
uint	Length in byte of UADData3 Seen value 12.
uint	Unknown63. =92

Length: 242

Type	Content
byte[60]	=0

Length: 310

Type	Content
byte[20]	=0
uint	Start offset of UADData4
uint	=1
uint	Length in byte of UADData4 Seen value 7.
uint	=0
uint	Start offset of UADData5
uint	=1
uint	Length in byte of UADData5 Seen value 10.
uint	=0
uint	Start offset of UADData6
byte[12]	=0

No extra data.

The additional data after the end record are all pointed to via absolute file offsets. Theoretically, there could be unused bytes before, between, and at the end. But all examples are compact. It is not clear whether the referenced data are placed in the file in a predetermined order.

## Structure of waypoint index table

Type	Content
uint	Length of the remaining part of the table in byte (i.e. 4 x number of waypoints +4+4)
uint	Unknown75, seen value 2
uint	Number of waypoints, no matter how they are distributed amongst the POI Groups
Number of waypoints times:	
uint	Absolute (file) offset of the waypoint, not necessarily in order

## Structure of UADData1

Table of content (structure below), followed by the referenced Sections.

Type	Content
ushort	Length of table of contents =92
ushort	Unknown100. =5
uint	CodePage, seen value 65001 = UTF-8
uint	Unknown 101. = 0x6C02 (27650)
uint	Start offset of UADData1/Section 0
uint	Length of UADData1/Section 0
uint	Length of each entry in UADData1/Section 0 =5
uint	=0
uint	Start offset of UADData1/Section 1
uint	Length of UADData1/Section 1
uint	Length of each entry in UADData1/Section 1 =4
uint	=0
uint	=0 (Section 2?, empty)
uint	=0
uint	=1
uint	=0xE01
uint	Start offset of UADData1/Section 3
uint	Length of UADData1/Section 3
uint	Length of each entry in UADData1/Section 3 =4
uint	=0
uint	Start offset of UADData1/Section 4
uint	Length of each entry in UADData1/Section 4 =5 <b>Note the reverse order!</b>
uint	Length of UADData1/Section 4
uint	=0

The referenced Sections immediately follow the table of contents. Although the offsets are absolute file offsets, the referenced Sections are **within** UADData1. The order of the Sections does not correspond with the order in which they are listed in the table of contents. Seen sequence: Section 0, 3, 4, 1.

Content of sections – unknown

## Structure of UADData2

Table of contents.

Type	Content
uint	Length of table of contents =84
uint	Length of UADData2/Section 0 (may be 0)
uint	Start offset of UADData2/Section 0
uint	Length of each entry in UADData2/Section 0 =37
uint	Length of UADData2/Section 1
uint	Start offset of UADData2/Section 1
uint	Length of each entry in UADData2/Section 1 =3
uint	Length of UADData2/Section 2 (may be 0)
uint	Start offset of UADData2/Section 2
uint	Length of each entry in UADData2/Section 2 =6
uint	=0 (Section 3?, empty)
uint	=0
uint	=0
uint	Length of UADData2/Section 4 (may be 0)
uint	Start offset of UADData2/Section 4
uint	Length of UADData2/Section 5 (may be 0)
uint	Start offset of UADData2/Section 5
uint	Length of each basic entry in UADData2/Section 5 Seen values: 1 when length =0, 3 when length != 0
uint	Length of UADData2/Section 6 =0
uint	Start offset of UADData2/Section 6 = Start of UADData2/Section 2
uint	Length of each entry in UADData2/Section 2 =0

The referenced Sections immediately follow the table of contents, and therefore UADData2. They are **not part of** UADData2, that is different than with UADData1. The order if the Sections does not correspond with the order in which they are listed. Seen sequence: Section 1, 2, 4, 5, 0. Which makes sense, because Section 0 holds the offsets of the entries of Section 4 and 5, and thus is better written after filling Sections 4 and 5.

Content of section 0: Sequence of fixed-size entries of following structure:

Type	Content
byte[20]	Unknown
uint	Offset of entry in UADData2/Section 4
uint	Length of entry in UADData2/Section 4
uint	Offset of entry in UADData2/Section 5
uint	Length of entry in UADData2/Section 5 Is always a multiple of 3, the length of each basic entry in Section 5
byte	Unknown =9

These entries create pairs consisting of one entry in section 4 and one entry in 5. The entries in section 4 and 5 are consecutive without gap and cover the whole section 4/5.

Content of section 1 – unknown

Content of section 2 – unknown

Content of section 4: Sequence of variable-sized entries, whose offset/length is in section 0 – unknown

Content of section 5: Sequence of variable-sized entries (consisting of a set of 3-byte basic entries), whose offset/length is in section 0 – unknown

## Structure of UADData3

Type	Content
uint	Start offset of UADData3/Table of pictures
uint	=0x90
uint	=0x1D8

Structure of UADData3/Table of pictures:

Type	Content
uint	Length of this table in byte (without this length field)
byte	=0
Fields per picture:	
ushort	Seen values 0x2A, 0x40, 0x50, 0x5E
ushort	Same value again
uint	Bit 0-30: Length of picture in byte Bit 31: ?
uint	Start offset of picture (in PNG format)

## Structure of UADData4

Type	Content
byte[3]	=0
uint	Start offset of UADData4/Table of pictures

Structure of UADData4/Table of pictures:

Type	Content
ushort	Number of pictures
Fields per picture:	
uint	Length of picture including this 12-byte header
uint	Type or flags? Seen value 0x0A, 0x12, 0x1A, 0x22, 0x10A, 0x112, 0x11A
uint	Length of picture in byte (= first field - 12)
byte[n]	Embedded picture (in PNG format)

## Structure of UADData5

Type	Content
uint	Start offset of UADData5/Table of pictures
byte[6]	=0xFF

Structure of UADData5/Table of pictures: Same as UADData4/Table of pictures

## Structure of UADData6

Type	Content
byte[8]	=0

## Record type 23: ?

Main data: Length is at least 10.

Type	Content
ushort	Unknown39. =1
ushort	Unknown40. Seen values 2 and 6
ushort	Number of Bitmap records in extra data
LString	Unknown42. ?

Extra data is sequence of records.

Bitmap+

## Record type 24: ?

Main data: Length is 2.

Type	Content
ushort	Unknown44. =1

No extra data.

## Record type 26: ?

Main data: Length is 5 or 6.

Type	Content
byte	Unknown45. Seems to be flags. Seen values 1 and 0x41. Bit 0= always set Bit 6=indicates the presence of the optional 6 <sup>th</sup> byte
uint	Unknown46. ? <b>Some kind of checksum?</b>
byte	Unknown47. Present if and only if bit 6 of Flags byte is set. Seen value 0.

No extra data.

## Record type 27: Alert trigger options

Main data: Length is at least 4 and even.

See [9].

Type	Content
ushort	Bit 2-0 (mask 0x0007): number of bearings following Bit 3 (mask 0x0008): Whether date/time list follows Bit 15-4: =0
Repeat number of bearings:	
ushort	Bit 8-0 (mask 0x01FF): bearing angle

Type	Content
	Bit 12-9 (mask 0x1E00): bearing width (needs to be multiplied with 5) Bit 13 (mask 0x2000): bi-directional (0=no, 1=yes) Bit 15-14: =0
The following fields are present only if bit 3 is set:	
ushort	Length of date/time list in byte
Repeated until length of date/time list reached: (which should be same as until flag "last entry" is set)	
byte	Type 0x00: by month 0x20: dates 0x40: day of year by week 0x60: day of month 0xC0: week of month 0xE0: week of year
byte	Flags Bit 1-0: depends on type Bit 2: has time start Bit 3: has time end Bit 4: has days of week end Bit 6-5: =0 Bit 7: last entry (0=no, 1=yes)
...	depends on type and flags, see below
byte	if Flags.bit2 is set: Bit 6-0: time start hours Bit 7: has time start minutes
byte	if Flags.bit2 and time start hours.bit7 are set: time start minutes
byte	if Flags.bit3 is set: Bit 6-0: time end hours Bit 7: has time end minutes
byte	if Flags.bit3 and time end hours.bit7 are set: time end minutes
byte	if Flags.bit4 is set: days of week end Bit 0: Sunday Bit 1: Monday Bit 2: Tuesday Bit 3: Wednesday Bit 4: Thursday Bit 5: Friday Bit 6: Saturday Bit 7: =0

Type 0x00: by month

Type	Content
byte	if Flags.bit0 is set: month from
byte	if Flags.bit1 is set: month to

Type 0x20: dates

Flags.bit1 = has day/month is always set

Type	Content
------	---------



if Flags.bit0 (has no year) is set:		
	ushort	Bit 3-0: month Bit 8-4: day Bit 15-9: =0
else		
	byte	Day
	ushort	Bit 15-4: year Bit 3-0: month

Type 0x40: day of year by week

Type	Content
if Flags.bit1 (has days of year) is set:	
	ushort Day of year from
	ushort Day of year to
if Flags.bit 0 (has days of week start) is set:	
	byte Days of week start Bit 0: Sunday Bit 1: Monday Bit 2: Tuesday Bit 3: Wednesday Bit 4: Thursday Bit 5: Friday Bit 6: Saturday Bit 7: =0

Type 0x60: day of month

Flags.bit0 = 0

Type	Content
if Flags.bit1 (has days of month) is set:	
	byte Day of month from
	byte Day of month to

Type 0xC0: week of month

Flags.bit0 = 0

Type	Content
if Flags.bit1 (has week of month) is set:	
	byte Week of month from
	byte Week of month to

Type 0xE0: week of year

Flags.bit0 = 0

Type	Content
if Flags.bit1 (has week of yeat) is set:	
	byte Week of year from
	byte Week of year to

No extra data.

## References

### Used sources of information

- [1] [https://www.gpsbabel.org/htmldoc-development/fmt\\_garmin\\_gpi.html](https://www.gpsbabel.org/htmldoc-development/fmt_garmin_gpi.html)
- [2] [https://github.com/gpsbabel/gpsbabel/blob/master/garmin\\_gpi.cc](https://github.com/gpsbabel/gpsbabel/blob/master/garmin_gpi.cc)
- [3] <https://www.pinns.co.uk/osm/gpi.html>
- [4] [https://en.wikipedia.org/wiki/Unix\\_time](https://en.wikipedia.org/wiki/Unix_time)
- [5] Code Pages <http://msdn.microsoft.com/en-us/goglobal/bb964653.aspx>
- [6] Code Pages [http://en.wikipedia.org/wiki/Code\\_page](http://en.wikipedia.org/wiki/Code_page)
- [7] <https://www.gpspower.net/software-tools/196870-garmin-content-toolkit-v-4-0-a-4.html>
- [8] <http://www.poi-factory.com/node/31703>
- [9] GPI Reader by [milokz@gmail.com](mailto:milokz@gmail.com)  
<https://github.com/dkxce/KMZRebuilder/blob/master/GPIReader.cs>

### Standards and specifications

- [10] ISO/IEC 646:1991, Information technology – ISO 7-bit coded character set for information interchange
- [11] ISO 639-1:2002 – Codes for the representation of names of languages – Part 1: Alpha-2 code

### Sources of sample files

- [12] <https://www.pinns.co.uk/osm/gpi.html>
- [13] <https://www.bordatlas.de/overlays.php>
- [14] <http://www.gmaptool.eu/en/content/gpi-device>
- [15] <https://www.garmin.com.sg/richpoi/>
- [16] <https://www.garmin.com/de/extras/poi2/>
- [17] <https://www.garmin.ru/download/extras/poi.php>
- [18] <http://gps.maroufi.net/gpi-pois.shtml>
- [19] <https://www.tourenfahrer.de/reise/pois-fuers-motorradnavi/motorradhotels-als-poi/tf-partnerhaeuser-als-pois-fuer-garmin/>
- [20] <https://www.promobil.de/gps-daten/stellplaetze-gps-daten-navi/>
- [21] <https://bikeaway.info/gps-navigation/poi-gps-navigation/bmw-motorradhaendler-poi/>

- 
- [22] <https://www.paesse.info/pois-points-of-interest-paesse-schweiz-und-alpen-fuer-garmin-und-tomtom/>
  - [23] <https://github.com/gpsbabel/gpsbabel/tree/master/reference>
  - [24] <https://sites.google.com/a/smail.nchu.edu.tw/chemistry-education/gps/garmin/speed-camera>