STRUCTURED VECTOR FILE FORMAT (TOPOLOGICAL OR NOT) IN MIRAMON (POINTS, ARCS, NODES AND POLYGONS)

Document authors: Abel Pau and Xavier Pons
Initial proposal: 18-09-2014 (04-12-2014, v. 1.1)
Last modified and version: 11-04-2024, **2.1**

1. Background, motivation and structure of this document

As MiraMon began development, Xavier Pons and Joan Masó designed and created formats for the software vector files, which had to be able to contain Points, Arcs, Nodes and Polygons, and be a balance between maximizing their coherence (reducing potential inconsistencies by using common elements between them) but without compromising their performance (something that would have happened in a fully normalized model of interdependencies). Their conception, therefore, aimed for the new files to have a structure that allowed very agile computer processing and, at the same time, to make explicit the topological relationships between entities or geographic objects when needed.

These files have an internal **computer structure** that makes them suitable **to efficiently access 2D and 3D** (from v. 1.1) **vector information**. That is why they are also used to contain vector information without topological structure. Therefore, it is important to understand that:

1. **The MiraMon structured vector format may contain files with or without explicit topological structure.**
2. **The format collects information about whether it contains vector elements with known topological relationships, or not (commonly known as "spaghetti" in the case of linear elements); more on this below.**
3. **A MiraMon structured file may contain data imported from a file without topology (such as a SHP or a DXF) and in this case it will be simply a structured file, NOT a topological file.**
4. **A MiraMon structured file may contain data with explicit topological structure (for example as a result of topological digitizing in MiraMon, resulting from a topological structuring process with certain options of applications such as LinArc, or resulting from an import of a format without topology, such as a SHP, when during that import a topology building process is made). In this case it is a proper structured topological file.**

In v. 2.0 of this document the v. 2.0 of the format is included, which essentially supports 64-bit values, and no longer 32-bit, for integers that indicate positions within the file (offsets) and for object identifiers, as well as some other modifications. This means that **from v. 2.0 both the possible files size, and the number of elements they can contain, are virtually infinite.**

This document complements the information of the MiraMon help file "*MiraMon vector formats description*", where the description of the structured formats is given from the user's point of view. Reading that help file is recommended to be familiar with MiraMon files. It is accessible directly through the MiraMon online help link: https://www.miramon.cat/help/eng/mm32/ap2.htm

Section 2 of this document describes the common part (topological header) in all the files and the specific part for each type of file. **Section 3** considers factors relating to arcs and polygons, and in **section 4** a complex polygon case is illustrated.

It is important to understand that, unlike other models of topological structure, in which the arcs can only be used to cycle (build) a single layer of polygons, in the MiraMon model a single layer of arcs can be used to cycle several layers of polygons (a layer of administrative limits can server to cycle a layer of municipalities, a layer of counties, etc); of course, in each layer of polygons, only those arcs needed to cycle the corresponding polygons are used. This causes that the classical topological information about which polygon is on the right or on the left of each arc is not in the layer of arcs, but in the polygon layer itself.

2. Description of the MiraMon structured vector file format

A MiraMon vector layer, whether it contains Points, Arcs, Nodes or Polygons, consists of several files. The main files are 3:

- **A file that contains the graphic information** (geometric and, if it is the case, topological), with coordinates, space dependencies, etc, and that has **.pnt**, **.arc**, **.nod** and **.pol** extension. This document explains the format of the files corresponding to the part of the graphic elements.
- **A main table of user attributes**, which is in DBF format, or in extended DBF format if more than 254 fields are required, if fields of more than 254 characters are needed, if field names up to 128 characters are needed, if the number of records is greater than 4200 million, etc. The DBF format is a well-known and documented format, while the documentation of the extended DBF format can also be found in the technical document "Extended DBF" format specification created by Xavier Pons and Abel Pau. **The .dbf extension is preceded by the letters 'T', 'A', 'N' or 'P', depending on whether it is a DBF relative to a points, arcs, nodes or polygons layer.** The rest of the name is the same as that of the graphic file and is stored in the same directory.
- **A text file, in INI Windows format**, described in the help section of MiraMon, **contains the layer metadata** and also describes both the **default** (optional) **symbolization** and the possible **relationships of the main table with other tables** (be DBF or others, such as tables or queries in Access files, large database managers such as SQL Server, Oracle, etc). **The extension for this file is .rel, and is preceded by the letters 'T', 'A', 'N' or 'P', in relation with a REL relative to a points, arcs, nodes or polygons layer**. The rest of the name is the same as that of the graphic file and is stored in the same directory.

Internally the n points, arcs, nodes and polygons are indexed from 0 (the first element is 0, the second one is 1, etc, up to n-1 elements that the file contains). This numbering provides what is called a graphic identifier. It is never written in the binary file and is given by the order in which the elements in the file are written. Nevertheless, from the user point of view, MiraMon usually shows a numbering from 1, which seems more natural, except when highly technical information is given. For example, when in a query by location MiraMon shows the text "Graphic element 3 of 8", internally the element corresponds to the graphic identifier 2. In polygon files, the universal polygon or polygon zero is the one that is "outside all other polygons" (or inside holes when they have no content of any kind), and

therefore the first "functional element" is the polygon with identifier 1; you will find more information about the zero polygon in section 3.

It must be considered that the order in which the byte bits are written always follows the Intel convention, not the Motorola one. In this document the name "double" refers to a 64-bit real number (termed double in the C programming language); doubles have enough numerical precision and range to store the coordinates used in geographic information.

Comment on offsets: In addition, a general comment should be noted with reference to any offset that appear in these specifications:

Up to and including version 1.1, the offset value that documents where to look for an element (an X,Y coordinate, a Z coordinate...) was stored in a 4-byte unsigned integer variable (the maximum value that could be stored was 4 294 967 295), meaning that the first Z coordinate of the last stored point could be, at most, at the byte position corresponding to slightly before the end of the file size of about 4.29 Gbyte (in this document we use the prefixes of mega (M), giga (G) or tera (T) as in SI, not as in computing: dividing by multiples of 1000, not 1024).

From version 2.0 this number is expanded to 8 bytes, which can reach **files of 18 million Tbytes**. It will hardly need to be expanded further in our lifetime (or so must have thought those who believed that 4 Gbyte of memory was "virtually infinite", like the creators of the C language, who foresaw that the fseek() function only needed an integer of 32 bits to indicate an offset). The same goes for **object identifiers**: they become encoded with a 64-bit unsigned integer and, therefore, **the number of possible objects is virtually infinite.**

2.1. Structured headers common to all files

2.1.1. Header common to all structured vector files

All structured vector files in MiraMon have a common part at the beginning: the topological header. This header has a size **sizeof_TH**, which is **56 bytes (48 bytes in version 1.x)**. The structure and content of the rest of the file depends on whether it contains points, arcs, nodes or polygons.

Description of the sizeof_TH bytes header:

	Topo Header (TH)	
0	3	File type (PNT, ARC, NOD, POL)
3	2	Version (" 0"-"99")
5	1	"."
6	1	Subversion ("0"-"9")
7	1	Flag (1 byte)
8	8	Bounding box: Minimum X
16	8	Bounding box: Maximum X
24	8	Bounding box: Minimum Y
32	8	Bounding box: Maximum Y
40	8 (4 in v. 1.x)	Element count
48 TH+44 in v. 1.x	8 (4 in v. 1.x)	Reserved

File type, Version, Subversion

It is a string consisting of **3 characters** declaring the file type (PNT, NOD, ARC or POL and that matches the file extension) and **4 characters** that denote the format version, with one decimal place; these 4 characters align to the right. For example, a typical start is: "PNT 1.1". If one day a 12.3 version would exist, the string will be: "PNT12.3".

In version 1.x the number of bytes intended to indicate the (integer) number of objects is 4 and therefore up to 2^{32} elements (~4200 million elements) can be saved.

As of April 15, 1997, an version 1.1 to support 3D coordinates (X,Y,Z) was designed that faithfully kept downward compatibility with the initial 1.0 version, which only supported 2D (X,Y) coordinates.

On March 21, 2023, a second version was designed, 2.0, which faithfully preserves the philosophy of 1.1, but no longer supports reading by applications compiled without knowing the specifications of 2.0. This version arises from the need to expand both the number of graphic elements that a file can contain, as well as the location of the coordinates of these objects within the files; these values are encoded in unsigned 8 byte (64-bit) integers. Since 8 bytes are used, in version 2.0 the total number of possible elements is 2^{64} elements (~18 million Tbyte).

Later in this document we explain how to access the coordinate location for versions 1.x and 2.0.

Flag

The **byte flag** can define up to 8 logical properties in the respective bits (True: 1 or False: 0) with different meanings depending on the nature of the geometric objects of the file. The following bits are defined at the date of this document.
Bits valid for PNT, ARC, NOD and POL files:

bit 0

Indicates that the topology has been verified by an application considered reliable. A value of 1 informs that the file has been generated with a MiraMon application that guarantees that the indicated topology is correct, or that the file has been imported from another format where topological relationships were present and considered reliable.

bit 1

Indicates that the file has been generated with a MiraMon application. Note that this bit can be set to 1 even if the file does NOT contain topology at this time.

Bit only valid for PNT, ARC and POL files (in NOD files it has to be 0 in versions 1.x and 2.0).

bit 2

For PNT: A value labeled as 1 indicates that the point file comes from a POL file via the MiraMon support application (MSA) "Etiqueta" ("Label") and that contains a label on polygon zero. The MiraMon MSA "AtriTop" checks it to decide whether to inherit attributes of the polygon zero. When in doubt, write a 0 in the bit.

For ARC: A value of 1 indicates that the arc file contains only edges of polygons. This means that a total cycling process (in which all arcs are involved) has been possible, which guarantees that the file does not contain either end nodes or arcs with the same polygon on both sides (dumbbells). If in doubt, write a 0 in the bit.

For POL: A value of 1 indicates that the file has been correctly tagged with the MiraMon "AtriPol" or "AtriTop" MSA, or with another MiraMon MSA such as "RasTop". Specifically this means:

- There is no label in polygon zero.
- Does not present incoherent re-labeled polygons.

- There are no polygons without labels.

In case of doubt or in the NOD case, write a 0 in the bit.

bit 3

Bit only valid for PNT and POL files:

For PNT: A value of 1 indicates that the point file comes from a POL file via the MiraMon "Etiqueta" ("Label") MSA and does **not** have a label on polygon zero.

For POL: A value of 1 indicates that the polygon file contains groups (or regions) in polygons different from the polygon zero. If topology is not verified bit 0 must be turned off.

When in doubt, write a 0 in the bit.

bit 4

Only for PNT and ARC: The file presents 3D coordinates. POL and NOD files can be 3D, but their Z coordinates are always contained in the corresponding ARC file.

bit 5

Only applies to polygons. A value of 1 means that the arcs involved in the cycling (remember that unused arcs are allowed) are used only once. It cannot be combined with bit 0 since this last flag would imply that the arcs would be used twice (at least against the polygon zero) and that overlays would be prohibited, situations that we want to allow in the case of the explicit polygons (not topological). See also the "Note on explicit polygons" at the end of the document.

bit 6

For POL: A value of 1 indicates that the polygon file contains groups (or regions) in the polygon zero. This can be interpreted as: some polygon different from polygon zero has one or more holes inside. This bit together with bit 3 allows to know all the information regarding groups in a polygon file. If topology is not verified, bit 0 must be turned off.

When in doubt, write a 0 in the bit.

Bounding box

Indicates the total bounding area in the order minX, maxX, minY, maxY, as in the documentation REL file (all the bounding boxes of the binary files respect this agreement). A double (real 64-bit) value is used for each member of the bounding box.

Element count

Indicates the total number of entities in the file. An unsigned __int64 is used (unsigned __int32 in v. 1.x).

Reserved

They are reserved for future extensions or for internal use. Documented internal uses are:

- During intensive file structure modification operations, such as during digitizing or during other formats import:
    - Bit 0 of the first reserved byte will be 1 if the XY coordinates of the ARC (or, more rarely, PNT) files are not in the zone after AH (TH in points), but in a file located in the same directory, with the same name, and with the "extension" "*#.~xy" (where # is A for arc layers and T for point layers). In this situation, and in the case of ARC/PNT files, the offsets recorded in AH/TH will refer to positions in this "*#.~xy" file, which will be rewritten to offsets in the ARC/PNT file when the modification of the file is finished and the part of coordinates is written in the ARC/PNT file, and the file "*#.~xy" is removed.
    - For the polygons case, bit 0 of the first reserved byte will be 1 if the arc indices of the POL files are not in the zones after PH but in a file located in the same directory, with the same name, but with "extension" "*P.~idx". In this situation the offsets recorded in PH will refer to positions in this "*P.~idx" file, which will be rewritten to offsets in the POL file when the modification of the file is finished and the indices part is written in the POL file, and the "*P.~idx" file is deleted.
    - For the nodes case, bit 0 of the first reserved byte will be 1 if the indices of confluent arcs at each node of the NOD files are not in the zones after NH, but in a file located in the same directory, with the same name, but with the "extension" "*N.~idx". In this situation the offsets recorded in NH will refer to positions in this "*N.~idx" file, which will be rewritten to offsets in the NOD file when the modification of the file is considered complete, and the index part is written in the NOD file and the "*N.~idx" file is deleted.
    - Bit 1 of the first reserved byte will be 1 if the Z coordinates of the PNT or ARC files are not in the zones after TL or AL, respectively, but in a file located in the same directory, with the same name, but

with the "extension" "*#.~z" (where # is T for point layers and A for arc layers). In this situation the offsets recorded in ZD will refer to positions in this "*#.~z" file, which will be rewritten to offsets in the PNT or ARC file when the modification of the file is finished and the coordinates' part is written in PNT or ARC files, and the "*#.~z" file is deleted.

2.1.2. Altitudes for the 3D file case.

In the case of PNT and ARC, if the file is 3D, there are two sections to define altitudes.

Section Z.

Section Z is divided into three subsections:

Subsection ZH. **32-byte** header that is defined below.

Z Header (ZH)

ZH	16	Reserved
ZH+16	8	Bounding box: Minimum
ZH+24	8	Bounding box: Maximum

ZD

Reserved

16 bytes that are reserved for future extensions. Filled with 0.

Bounding box: Minimum Z

Minimum value of all Z in the file. One double is used.

Bounding box: Maximum Z

Maximum value of all the Z in the file. One double is used.

Subsections ZD.

For EACH ITEM (point or arc) a **32-byte** header is written **(24 bytes in version 1.1)** with the following structure.

Z Description (ZD)

ZD	8	Bounding box: Minimum Z
ZD+8	8	Bounding box: Maximum Z
ZD+16	4	Z count
ZD+20 (non-existent in v. 1.1)	4 (non-existent in v. 1.1)	Reserved
ZD+24	8 (4 in v. 1.1)	Offset of 0th ZL

ZL

Bounding box: Minimum Z

Minimum value of all the Z of the point.

Bounding box: Maximum Z

Maximum value of all the Z of the point.

Z count

In case of PNT: Number of point altitudes (value expressed as a negative number). The number of altitudes is always a negative or a zero value in point files and indicates the number of point altitudes. In the arcs section we describe the meaning of a "Z count" when it is a positive value (which in points does not make sense).

In the case of ARC: Number of arc altitudes. If the **number of altitudes is positive** this indicates the number of altitudes for each vertex of the arc, and all the altitudes of the vertex 0 are written first, then those of the vertex 1, etc. If the **number of altitudes is negative** this indicates the number of arc altitudes, understanding that all the vertices have the same altitude (it is the case of a contour line, for example). Use -1.0E+300 as NoData if one of the altitudes of any vertex is not known (implementation note in the MiraMon software: this value will correspond to the "const double NODATA_COORD_Z_**v**_**2**" variable of PrjMMVGl.h/c).

> Note on NoData in Z: As a result of a research about the reliability in the comparison of double precision reals done in 2017 by Xavier Pons (and which is part of the document PrecisioEnReals_*.doc) it

was proposed, **for version 2.0 only**, a "const double NODATA_COORD_Z_**v**_**2**" that defines, like -VALOR_ESTAD_INDEFINIT, at 2.9E+301 for the reasons of direct comparison and reliable printing explained in NOTE_XP_30-08-2017_30 of the EstadStr.h file of the MiraMon code, at the time 1.0E+300 was abandoned as a VALOR_ESTAD_INDEFINIT. If in reading and writing the comparison were easy and there was no need to strongly fork the code, the new value could be adopted, but this fork would delay the implementation of the v. 2.0 of MiraMon structured vector files and, moreover, in practice the old value 1.0E+300 has not yet been problematic for direct comparisons with NODATA_COORD_Z (*i.e.*, it has not been necessary to use, in the MiraMon code the very slow functions of the family "double_diferents..."). Another possibility, much better, which is left here written but which will not be applied yet in v. 2.0, is reserving an 8-byte space for Z data in the disk format, in the first reserved bytes of the Z Header (ZH), and loading this value into the memory structure, so that, when reading a file into memory there would be 1.0E+300 in versions where this reservation does not exist (and the value 1.0E+300 is assumed) while, in versions already having NoData written to this disk position, the memory would contain whatever it is set on disk (by default: NODATA_COORD_Z_**v**_**2**). In the code it would be elegant because it will be as in raster files, where a member of the structure specifies the NoData value. When writing, the value indicated in the structure member (NODATA_COORD_Z in versions 1.x or 2.0 of the file and, later, NODATA_COORD_Z_**v**_**2**) should be used. This step will be finished in an ulterior version (perhaps 2.x) in order to be able to explicitly indicate still more reliable NoData values (which support a comparison with an equal and which do not have problems when their decimals are written) and to make potential problems disappear both in the MiraMon code and in the code from the GDAL libraries.

Since in the MiraMon structured model polygon edges are described by arcs, the explanations in this paragraph apply to polygon edges. Similarly, Z values of nodes are contained in the ARC file.

Example:

A 4-vertices arc with number of altitudes 2: the altitudes of the first vertex will be written, then those of the second, then those of the third, and finally those of the fourth. Total: 4x2=8 altitudes.

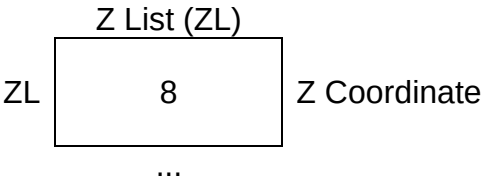
An arc of 4 vertices with number of altitudes -2: the two altitudes of the arc are written. Each vertex will have these two altitudes in the same order in all the vertices, although they will only be written once.

Offset of 0th ZL

Indicates the offset where the first Z coordinate (typically an altitude) is written. This is only relevant if the number of altitudes of this point is different from zero.

Subsections ZL.

The ZL section contains a list of altitudes of each point or arc vertex.



Z Coordinate

PNT case: Altitude of the point that is represented. One double is used.

ARC case: Altitude of the indicated vertex of the arc that is represented, or of the whole arc. One double is used.

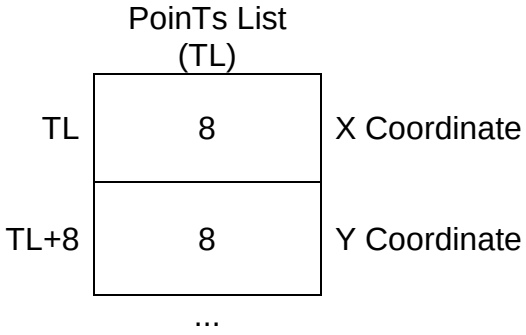
2.2. Points file .pnt

The point file format (PNT) contains two sections (three in the 3D case), described below:
Section TH. Common **header** in all files (**sizeof_TH**), previously described. The "File type" field corresponds to the "PNT" string.

Section TL. For each **POINT** the coordinates are written (**16 bytes**).

The first point is written in offset **sizeof_TH**, which is where the header always ends.

Description of the 16 bytes:



X Coordinate

X coordinate of the point that is represented. One double is used.

Y Coordinate

Y coordinate of the represented point. One double is used.

When flag 4 of the "Topological header common to all files" section is activated (1), a section 3, described below, exists.

Section Z (applies only in case the file is 3D).

The description of this section corresponds to section Z of the section "2.1.2. Altitude for the case of 3D file**".**

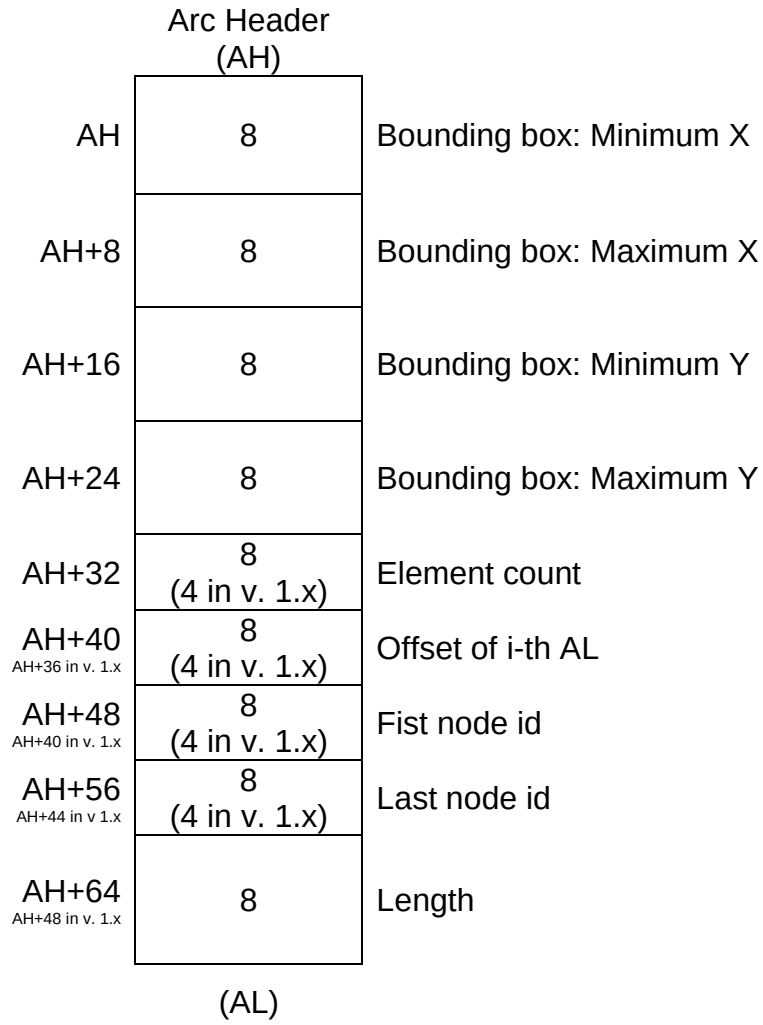
2.3. Arcs files .arc

The arc file format (ARC) contains three sections (four in the 3D case) described below:

Section TH. Common **header** in all files (**sizeof_TH**), previously described. The "File type" field corresponds to the "ARC" string.

Sections AH. For each **ARC**, a header of size **sizeof_AH** is written, which is **72 bytes (56 bytes in version 1.x)**. The first header is written in offset **sizeof_TH**, which is where the common TH header always ends. The rest of AH is in the offset sizeof_TH+sizeof_AH*id_arc.

Description of the sizeof_AH bytes of an arc header:



Bounding box

Indicates the bounding box of the arc described in this header in the order minX, maxX, minY, maxY. One double is used for each member of the bounding box.

Element count

Indicates the total number of arc vertices described in this header. One unsigned __int64 (unsigned __int32 in v. 1.x).

Offset of i-th AL

Offset of the first arc vertex described in this header. One unsigned __int64 is used (unsigned __int32 in v. 1.x).

First node id

Identifier of the initial node of the arc. This identifier refers to the list of node identifiers in the node file associated with the arc file. One unsigned __int64 is used (unsigned __int32 in v. 1.x).

Last node id

Identifier of the final node of the arc. This identifier refers to the list of node identifiers in the node file associated with the arc file that. One unsigned __int64 is used (unsigned __int32 in v. 1.x).

Length

Length of the arc that is described, in the same reference system as the coordinates. One double is used. Note that lengths over the ellipsoid are also computed, but stored in the main table (*A.dbf).

AL

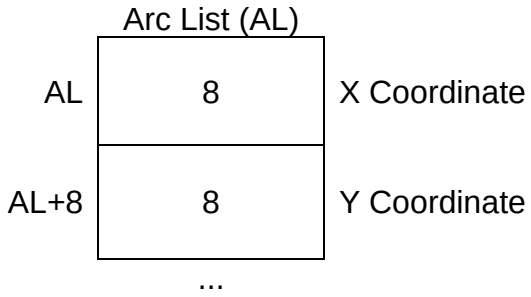
Immediately after the succession of AH it would be appropriate for AL sections to start appearing, but it is not necessary since each AL section can be found from the offset indicated in the corresponding AH section.

Section AL.

List of arc coordinates. These are the coordinates corresponding to each individual arc from the number of arc vertices defined in the 32nd byte of the corresponding AH *and* of the offset defined in the 40th byte (36 in v. 1.x) of the corresponding AH.

For each **VERTEX** of the arc, its coordinates **(16 bytes)** are written.

Description of the 16 bytes:



X Coordinate

X coordinate of the vertex that is represented. One double is used.

Y Coordinate

Y coordinate of the represented vertex. One double is used.

When flag 4 of the "Topological header common to all files" section is on (1), a Section Z is found, described below.

Section Z (only applies in case the file is 3D)

The description of this section corresponds to section 1 of the section "2.1.2. Altitude for the case of 3D file".

2.4. Nodes file .nod

The format of node files (NOD) contains three sections, described below:

Section TH. Common **header** in all files (**sizeof_TH**), previously described. The "File type" field corresponds to the "NOD" string.

Sections NH. For each **NODE**, a header of size **sizeof_NH** is written, which is **12 bytes (8 bytes in version 1.x).**

The first header is written in offset **sizeof_TH**, where the common TH header always ends. The rest of NH are in the offset sizeof_TH+sizeof_NH*id_nod.

Description of the sizeof_NH bytes node header:

Node Header
(NH)

NH	2	Arcs count
NH+2	1	Node type
NH+3	1	Reserved
NH+4	8 (4 in v. 1.x)	Offset of i-th NL

(NL)

Arcs count

Indicates the total number of arcs that converge at the described node. An unsigned short int (16-bit) is used.

Node type

Indicates the node type. It is used from version 1.1. Possible types of nodes are: typical node (Node type=0), line node (Node type=1), ring node (Node type=2) and end node (Node type=3). An unsigned char (8-bit) is used.

Reserved

It remains reserved for future extensions. A byte is used and currently takes the value 0.

Offset of i-th NL

Offset the first of the arcs that converge at the described node. One unsigned __int64 is used (unsigned __int32 in version 1.x). Offsets must be aligned to a multiple of 8 bytes. Thus, a ring node (Arcs count=1+reserved) occupies the same as a line node (Arcs count=2 and no filling is required). For this reason, the transformation of a ring node to a line node or vice versa does not alter the offsets of the file.

NL

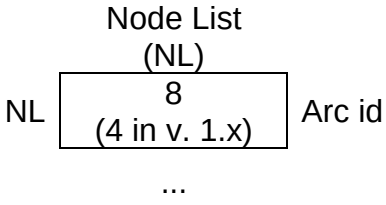
Immediately after the succession of NH, NL sections could begin to appear, but it is not necessary since each NL section can be found from the offset indicated in the corresponding NH section.

Sections NL

List of arc identifiers that connect to different nodes. Access is granted to each one from the offset of i-th NL.

For each **NODE**, arc identifiers that converge at the described node are written. Each item in this sequence of identifiers (as many as arcs converge at the node) occupies 8 bytes (4 in v. 1.x):

Description of the 4 bytes:



Arc id

Arc identifier in the arc file that converges at the described node. One unsigned __int64 is used (unsigned __int32 in v. 1.x).

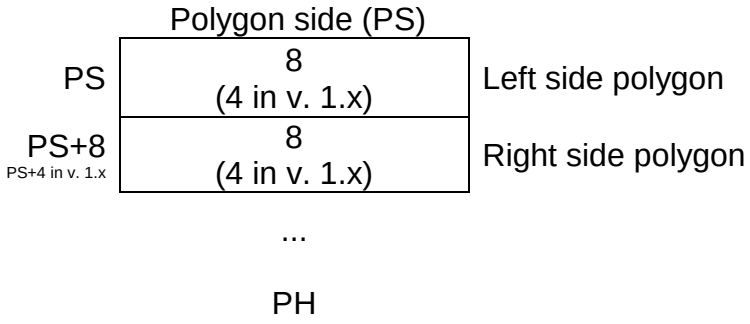
2.5. Polygons file .pol

The polygon file format (POL) contains four sections, described below:

Section TH. Common **header** in all files (**sizeof_TH**), previously described. The "File type" field corresponds to the "POL" string.

Sections PS. They contain, for each **ARC**, the identifiers of the polygons being on the left and on the right (in this order) of the described arc. The topological information of the first arc is written in offset **sizeof_TH** (where the common header always ends). Note, as explained in the introduction, that a second layer of polygons based on the same layer of arcs will refer to other arcs, specifically those that are needed to cycle (build) the polygons of the second layer. This header has a size of sizeof_PS, which is **16 bytes (8 bytes in version 1.x)**.

Description of the sizeof_PS bytes:



Left side polygon

Polygon located on the left of the described arc. One unsigned __int64 is used (unsigned __int32 in v. 1.x).

Right side polygon

Polygon located on the right of the described arc. One unsigned __int64 is used (unsigned __int32 in v. 1.x).

In v. 2.0, the arcs that do not participate in cycling will be set to "0xFFFFFFFFFFFFFFFF" (maximum value for an unsigned __int64). In v. 1.x, the arcs not participating in the cycling have the value at "0xFFFFFFFF" (maximum value for an unsigned __int32).

Sections PH. For each **POLYGON**, a header of size sizeof_PH is written, which is **80 bytes (64 bytes in version 1.x)**. The first header is at sizeof_TH+sizeof_PS*n_arc and the rest at sizeof_TH+sizeof_PS*n_arc+sizeof_PH*id_pol.

Description of the sizeof_PH bytes of a polygon header:

	Polygon Header (PH)	
PH	8	Bounding box: Minimum X
PH+8	8	Bounding box: Maximum X
PH+16	8	Bounding box: Minimum Y
PH+24	8	Bounding box: Maximum Y
PH+32	8 (4 in v. 1.x)	Arcs count
PH+40 PH+36 in v. 1.x	8 (4 in v. 1.x)	Arcs in external rings count
PH+48 PH+40 in v. 1.x	8 (4 in v. 1.x)	Ring count
PH+56 PH+44 in v. 1.x	8 (4 in v. 1.x)	Offset of i-th PL
PH+64 PH+48 in v. 1.x	8	Perimeter
PH+72 PH+56 in v. 1.x	8	Area

Bounding box

Indicates the bounding box of the polygon described in this header in the order minX, maxX, minY, maxY. A double is used for each member of the bounding box.

Arcs count

Indicates the total number of arcs needed to cycle the polygon described in this header. One unsigned __int64 is used (unsigned __int32 in v. 1.x).

Arcs in external rings count

Indicates the total number of outer arcs that cycle the polygon described in this header. In v. 2.0, if "0xFFFFFFFFFFFFFFFF" ("0xFFFFFFFF" in v. 1.x) is indicated, it means that anything is known about which arcs are defining internal rings (internal borders) and which are external rings; in this case the bit 0 of the corresponding element from the Polygon Arc List VFG (see the PAL section that follows) is always 0. One unsigned __int64 is used (unsigned __int32 in v. 1.x).

Ring count

A ***polypolygon*** is a polygonal entity that can be formed by more than one ring. This set of bytes indicates the number of polypolygon rings described in this header. In case of polypolygons with holes, the holes count like inner rings of the polypolygon. One unsigned __int64 is used (unsigned __int32 in v. 1.x).

Offset of i-th PL

Offset to the first arc of the polygon described in this header. It is advised to use multiples of 8. One unsigned __int64 is used (unsigned __int32 in v. 1.x).

Perimeter

Perimeter of the polygon described in this header, in the same reference system as the coordinates. One double is used. Note that perimeters over the ellipsoid are also computed, but stored in the main table (*P.dbf).

Area

Area of the polygon described in this header, in the same reference system as the coordinates. One double is used. Note that areas over the ellipsoid are also computed, but stored in the main table (*P.dbf).

PAL sections. For each **POLYPOLYGON** their arcs are written in a header of size sizeof_PAL, which is **9 bytes** (**5 bytes** in version 1.x).

Description of the sizeof_PAL bytes:

Polygon Arc List (PAL)

PAL	1	VFG
PAL+8	8 (4 in v. 1.x)	Arc id
	...	

VFG

The byte VFG (standing for "*Vora* (external edge) – *Fi* (end) – *Gir* (flip)") is used to determine characteristics of the arc that composes the polygon. When a bit is set (value 1) the property it defines is considered to be TRUE; otherwise, FALSE.

The following bits are defined at the date of this document.

bit 0 (V): Indicates whether the arc is part of an outer ring (value 1) or of an inner ring (0) of the polypolygon.

bit 1 (F): Indicates whether the current arc ends the ring (value 1) or if it is necessary to bind with another arc to close the ring (value 0). Therefore, if the number of 1's (value 1) of a polypolygon is counted, this coincides with the number of rings forming the polypolygon.

bit 2 (G): It can have two interpretations:

a. This Boolean indicates whether the polygon in question is in the left side (value 1) or in the right side (value 0) of the arc, according to the drawn direction of vertices in the ARC file.
b. In the case of defining explicit polygons (non-topological), this bit indicates whether the plot order of the vertices written in the arc file must be flipped (value 1) to write this ring fragment in the sequence of coordinates that describe the entire ring explicitly, or if the order do not have to be flipped (value 0); this is due to the fact that rings of explicit polygons must be constructed making the polygon itself remaining to the right, which allows the outer rings to be automatically calculated with a positive area and the inner rings with a negative area, and to be constructed in this manner it may be necessary to have to flip (invert) the order of the sequence of the vertices.

NOTE: The polygon always stands to the right of the succession of coordinates explicitly describing the ring. This criterion coincides with the criterion of ArcInfo. This means that the outer rings cycle clockwise and the internal rings counterclockwise.

Arc id

Arc identifier in the arcs file which is the base of this polygon file. One unsigned __int64 is used (unsigned __int32 in v. 1.x).

2.6. Format summary for all file types

Below is a summary table of all formats of MiraMon structured vector files.

Version 1.x:

	PNT
TH	48
TL	16
TL	...
ZH	32
ZD	24
ZD	...
ZL	8
ZL	...

	ARC
TH	48
AH	56
AH	...
AL	16
AL	...
ZH	32
ZD	24
ZD	...
ZL	8
ZL	...

	NOD
TH	48
NH	8
NH	…
NL	4
NL	...

	POL
TH	48
PS	8
PS	...
PH	64
PH	...
PAL	5
PAL	...

Version 2.0:

	PNT
TH	56
TL	16
TL	...
ZH	32
ZD	32*
ZD	...
ZL	8
ZL	...

	ARC
TH	56
AH	72*
AH	...
AL	16
AL	...
ZH	32
ZD	32*
ZD	...
ZL	8
ZL	...

	NOD
TH	56
NH	12*
NH	...
NL	8*
NL	...

	POL
TH	56
PS	16*
PS	...
PH	80*
PH	...
PAL	9*
PAL	...

* Marks that there is a difference from v. 1.x *vs* version 2.0. The difference is mainly due to two facts: an expansion in the number of bytes for the object identifiers, which go from 4 to 8 bytes, and an expansion in the offsets where the coordinates or identifiers of the described objects start, also moving from 4 to 8 bytes. In addition, there is some additional reserve of space that, at the same time, allows a more generalized 8-byte alignment (not total because there are structures, such as the NH, that have a smaller size).

3. Some considerations about the files containing arcs or polygons

- There is always a polygon, which is called polygon zero (or universal polygon). This makes sense in a file with guaranteed topology, but not in a file of explicit polygons. The polygon zero is composed of all arcs that form rings of all other polygons in the file, provided these arcs are not in contact with any other polygon. For example, in an archipelago in which the sea is the polygon zero, the different inner rings would be the outer

edges of the islands of the archipelago. In the case of a file composed of a single polygon with a hole (not with another polygon within the hole) the polygon zero has as arcs all the arcs of the file. When the zero polygon does not have a topological meaning (typically in layers of explicit polygons) it consists of zero arcs.

- In the case of being a file of explicit polygons, the polygon zero is documented with the header filled with 0 and has no PH or PAL section.
- Group files contain polygons grouped into groups of polygons. In these files object identifiers can refer to conventional polygons (with or without holes) or to groups of polygons. Each object has a single PH section.
- The order in which the arcs that constitute a polygon is written (polypolygon) is:
    1. outer ring
    2. inner rings contained in the previous outer ring
    3. outer ring (if there are other rings)
    4. inner rings contained in the previous outer ring.
    5. ...
- Each inner edge counts on the total count of rings.
- The perimeter of the polygon zero is the sum of lengths of all edges and has a positive sign.
- The area of the polygon zero is the sum of the areas of all polygons and with a forced negative sign.

Note on explicit polygons: A file of **explicit polygons**, whether of groups or not, should be defined as a POL file with the following particulars:

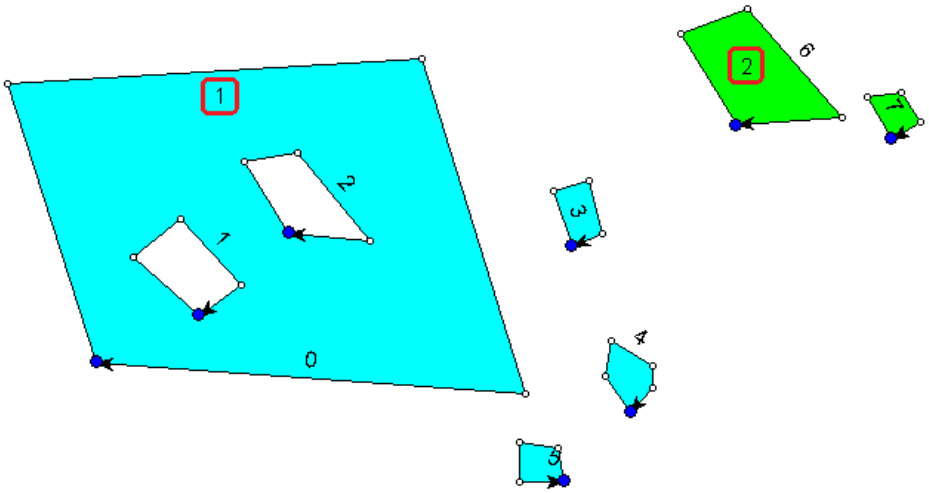
- In the flag of the section TH, bit 0 is not set and bit 5 is set. Bit 3 of this flag has the value as convenient.
- The section PS (of the polygons at each side of each arc) contains the graphic identifier of the polygon on one side and the polygon zero on the other, depending on the polygon being in the right or left side.
- Polygon zero is constituted by 0 arcs and has no PAL section.

A file of **NON-TOPOLOGICAL groups** (which support overlays) must be defined as a POL file with the following particularities:

- In the flag of the section TH, bit 0 is not set and bit 3 is set.
- The PS section (of the polygons of each side of each arc) is filled with 0xFFFFFFFFFFFFFFFF (maximum value of type unsigned __int64) in v. 2.0 (0xFFFFFFFF (maximum value of type unsigned __int32) in v. 1.x).
- The polygon zero is formed by 0 arcs and does not contain a PAL section.

4. Illustrative example of complex polygons

The following example illustrates one of the most complex cases: two polygons, the first with two holes and three enclaves, and the second without a hole and one enclave.



The polygon on the left part of the example (blue) is the polygon with identifier 1, because polygon zero "cannot be seen" (it is not painted). The polygon on the right part is the one with index 2 (green). The identifier of the arcs is clearly seen. The identifier of the two polygons is highlighted with a red mark. The rest of the identifiers correspond to the identifiers of the arcs, from the first, with value 0, which encircles the largest polygon, to the last, with value 7, which is located on the right side of the figure.

In order to clarify this example, we describe some interesting questions to highlight:

- The number of elements in this polypolygon is 2, not 6 (which is the number of rings). This is specified in the header of the polygon file described in the section TH, specifically in byte 40 (and occupy 8 bytes [4 in v. 1.x]).
- The byte 7 (flag) of the header of the polygon file described in section PH will have set, at least, bit 3.
- Section PS, which determines which polygon is in the left and right side, looks like this: 01-10-10-01-01-01-01-02-02 (hyphens are only visual aids).
- In section PH (headers of the polygons) it should be noted that the header of polygon 1 has 6 **Arcs count**, 4 **Arcs in external rings count** and 6 **Ring count**, while the head of polygon 2 will have 2 **Arcs count**, 2 **Arcs in external rings count** and 2 **Ring count**.
- Finally, section PAL will have the following aspect (VFG is shown as a set of three bits):

0: 1-1-0, 0
0-1-**1**, 1 (it must be flipped so that the direction of a hole is counterclockwise)
0-1-**1**, 2 (it must be flipped so that the direction of a hole is counterclockwise)
1-1-0, 3
1-1-0, 4
1-1-0, 5

1: 1-1-0, 6
   1-1-0, 7