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

Document authors: Abel Pau, reviewed by Xavier Pons
Initial proposal: 18-09-2014 (12-04-2014 v. 1.1)
Last modified and version: 13-03-2019. **1.6**

1. Background and motivation

As MiraMon began development, Xavier Pons and Joan Masó designed and created the formats of the topological files of MiraMon, which are four: Points, Arcs, Nodes and Polygons.

These files have an internal **computer structure** that makes them suitable **to efficiently access 2D and 3D 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 structured format of MiraMon 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 module options such as LinArc, or resulting from an importation of a format without topology, such as a SHP, when during that import a topology building process was made). In this case it is a proper structured topological file.**

This document complements "*APPENDIX 2. Vector formats description*" of the MiraMon help file, where the description of the structured formats is given from the user's point of view. Reading this appendix is recommended to be familiar with MiraMon files. It is accessible directly through the MiraMon online help link: http://www.creaf.uab.cat/miramon/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 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 graphical information (geometric and, if it is the case, topological), with coordinates, space dependencies, etc, and that have a .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, etc. The DBF format is a well-known and documented format, while the documentation of the extended DBF format can also be found in a technical document created by Xavier Pons and available on the MiraMon website. The .dbf extension is preceded by the letters 't', 'a', 'n' or 'p', depending on whether it is a DBF relative to a layer of point, arcs, nodes or polygons. 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 preceded by the letters 't', 'a', 'n' or 'p', in relation with a REL relative to a layer of points, arcs, nodes or polygons. The rest of the name is the same as that of the graphic file and is stored in the same directory.

Internally the points, nodes, arcs and polygons are indexed from 0 (the first element is the 0, the second one is de 1, etc). 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. For example, when in a query by location MiraMon the text "Graphic element 3 of 8" is shown, internally the element corresponds to the graphic identifier 2.

It must be kept in mind 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.

2.1.- Common structured headers to all files

2.1.1.- Header common to all structured vector files

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

Description of the 48 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	4	Element count
44	4	Reserved

File type, Version, Subversion

A chain 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 a decimal figure; these 4 characters align to the right.

As of April 15, 1997, an initial version 1.1 was designed that faithfully keeps downward compatibility with the initial 1.0 version. For example, a start type is: "PNT 1.1".
If one day existed a 12.3 version it would say: "PNT12.3".

Flag

The **byte flag** can define up to 8 logical properties in the respective bits (True: 1 or False: 0) independent 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 this version):

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 has a label on polygon 0. The MiraMon MSA "AtriTop" queries this to inherit or not attributes of the polygon 0. 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 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 0.
- Does not present incoherently re-labeled polygons.
- There are no polygons without labels.

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

Bit only valid for PNT and POL files:

bit 3

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 0.

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

Applicable to polygons, a value of 1 means that the Arcs that intervene in the cycling (unused arcs are allowed) are only used once (with the same restrictions as a VEC of polygons, but supports holes and groups). 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. 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.

Count

Indicates the total number of entities in the file.

Reserved

They are reserved for future extensions.

2.1.2.- Heights for the 3D file case.

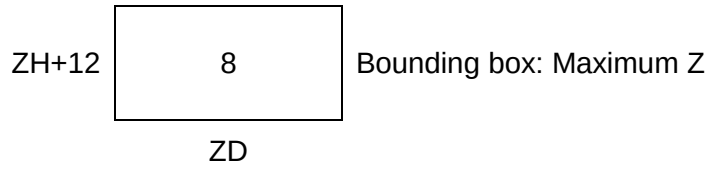
In the case of PNT and ARC, if the file is 3D, we will have two sections to define height.

Section Z.
Section Z is divided into three sections:

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

Z Header (ZH)

ZH	16	Reserved
ZH+4	8	Bounding box: Minimum Z



Reserved

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

Bounding Box: Minimum Z

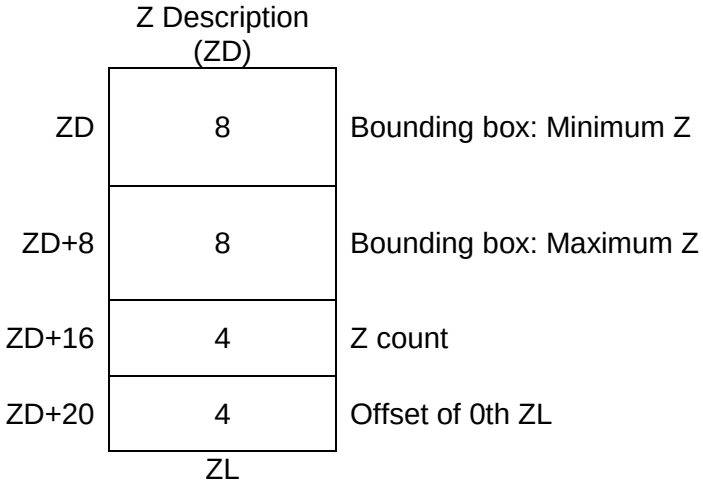
Minimum value of all z in the file. One double is used.

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 **24-byte** header is written that has the following structure.



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

In the case of ARC: Number of arc heights. If the **number of heights is positive** this indicates the number of heights for each vertex of the arc. First all the heights of the vertex 0, then those of the 1, etc are written. If the **number of heights is negative** this indicates the number of arc heights, understanding that all the vertices have the same height or heights (in the case of a countour line, for example). Use -1.0E+300 as no data if one of the heights of any vertex is not known.

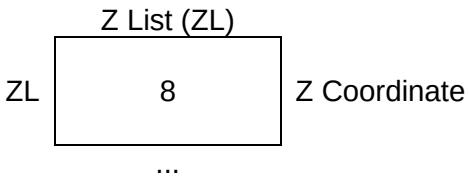
Example:
A 4-vertices arc with number of heights 2: the heights of the first vertex will be written, then those of the second, then those of the third vertex, and finally those of the fourth. Total: 4x2 = 8 heights.

An arc of 4 vertices with number of heights -2: the four heights of the arc are written. Each vertex will have these four heights in the same order in all the vertices.

Offset of 0th ZL

Indicates the offset where the first height is written. This is only relevant if the number of heights of this point is different from zero.

Subsections ZL.
The ZL section contains a list of heights of each point or arc.



Z Coordinate

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

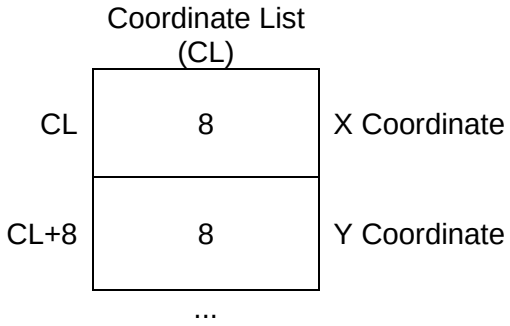
ARC case: Height of the indicated vertex of the arc that is represented or of the whole arc. One double is used.

2.2.- Points file .PNT

The point file format (PNT) contains two sections (three in the 3D case) and are described below:

Section TH. Common **header** in all files (**48 bytes**), previously described. The "File type" field corresponds to the "PNT" string.

Section CL. For each **POINT** the coordinates are written (**16 bytes**).
The first point is written in offset **48**, 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, must exist.

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

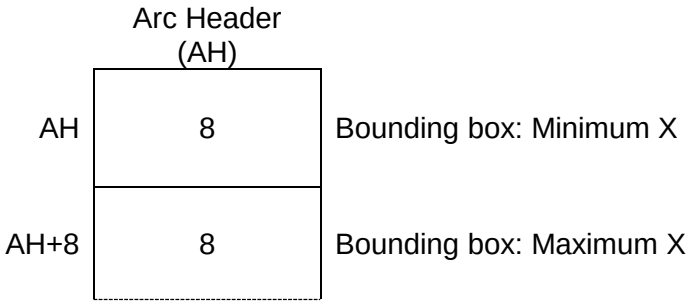
The description of this section corresponds to section Z of the section "**2.1.2.- Height for the case of 3D file**".

2.3 .ARC files

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

Section TH. Common **header** in all files (**48 bytes**), previously described. The "File type" field corresponds to the "ARC" string.

Sections AH. For each **ARC**, a **56-byte** header is written.
The first header is written in offset **48**, which is where the common TH header is always finished. The rest of AH is in the offset 48+56*id_arc.
Description of the 56 bytes of the header of an arc:



AH+16	8	Bounding box: Minimum Y
AH+24	8	Bounding box: Maximum Y
AH+32	4	Element count
AH+36	4	Offset of i-th AL
AH+40	4	Fist node id
AH+44	4	Last node id
AH+48	8	Length

(AL)

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 __int32 is used.

Offset of i-th AL

Offset of the first arc vertex described in this header. One __int32 is used.

First node id

Initial node index of the arc that refers to the list of nodes in the node file associated with the arc file that is described. One unsigned __int32 is used.

Last node id

Index of the final node of the arc that refers to the list of nodes in the node file associated with the arc file that is described. One unsigned __int32 is used.

Length

Length of the arc that is described, in the same reference system as the coordinates. One double is used.

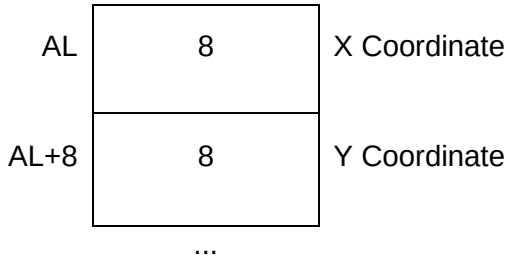
AL

Generally, 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 36th byte of the corresponding AH.

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

Arc List (AL)



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.- Height for the case of 3D file**".

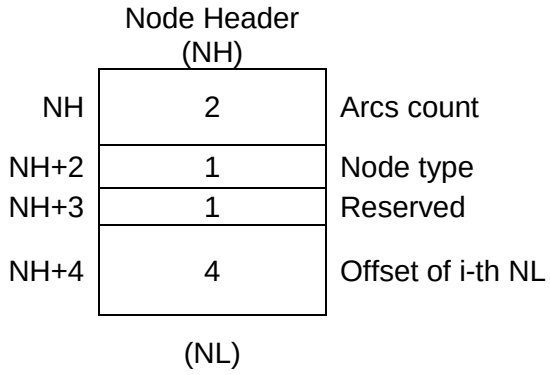
2.4.- .NOD node file

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

Section TH. Common **header** in all files (**48 bytes**), previously described. The "File type" field corresponds to the "NOD" string.

Sections NH. For each **NODE**, an **8-byte** header is written.
The first header is written in offset **48**, where the common TH header is always finished.
The rest of NH are in the offset 48+8*id_nod.

Description of the 8 bytes node header:



Arcs count

Indicates the total number of arcs that converge in 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 takes the value 0 at the moment.

Offset of i-th NL

Offset the first of the arcs that converge to the described node. One __int32 is used. 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.

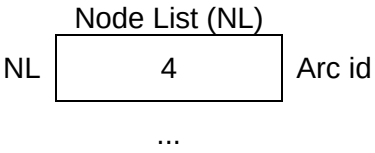
NL

Generally, immediately after the succession of NH, NL sections 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 index arcs that connect to different nodes. Access is granted to each one from the offset of i-th NL.

For each NODE, arcs' indexes that converge to the described node are written.

Description of the 4 bytes:



Arc id

Index of the arc in the arc file that converges to the described node. One unsigned __int32 is used.

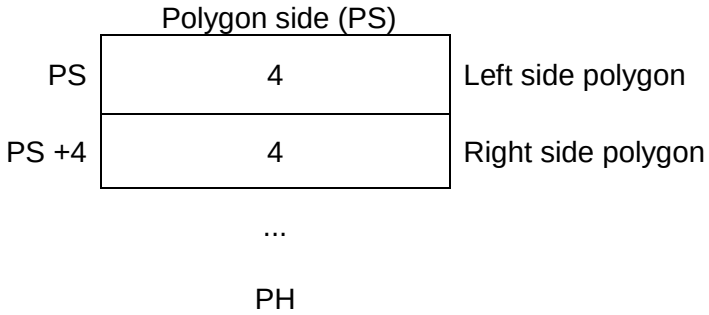
2.5. - Polygon file .POL

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

Section TH. Common **header** in all files (**48 bytes**), previously described. The "File type" field corresponds to the "POL" string.

Sections PS. They contain, for each **ARC**, the indexes 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 **48**, which is where the common header is always finished. 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 the polygons of the second layer.

Description of the 8 bytes:



Left side polygon

Polygon located on the left of the described arc. One unsigned __int32 is used.

Right side polygon

Polygon located on the right of the described arc. One unsigned __int32 is used.

The arcs that do not participate in the cycling will have the value at "0xFFFFFFFF" which is the maximum value for an unsigned __int32.

Sections PH. For each **POLYGON**, a **64-byte** header is written. The first section is foundin 48+8*n_arc and the other 48+8*n_arc+64*id_pol.

Description of the 64 bytes of the head of an arc:

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	4	Arcs count
PH+36	4	Arcs in external rings count
PH+40	4	Ring count
PH+44	4	Offset of i-th PL
PH+48	8	Perimeter
PH+56	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 that constitute the polygon described in this header. One unsigned __int32 is used.

Arcs in external rings count

Indicates the total number of outer arcs that constitute the polygon described in this header. If "0xFFFFFFFF" 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 __int32 is used.

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 polypoligones with holes, the holes count like inner rings of the polypolygon. One unsigned __int32 is used.

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 __int32 is used.

Perimeter

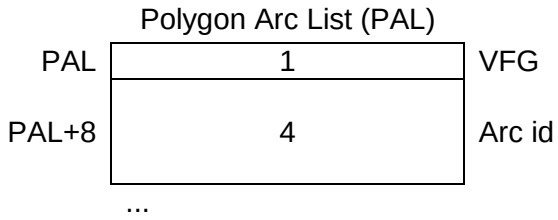
Perimeter of the polygon described in this header, in the same reference system as the coordinates. One double is used.

Area node

Area of the polygon described in this header, in the same reference system as the coordinates. One double is used.

Sections PAL. For each **POLYPOLYGON** the arcs that compose it (**5 bytes**) are written.

Description of the 5 bytes:



VFG

The byte VFG (standing for "*Vora* (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 that make up 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 vertices drawn direction in the ARC file.

b) In the case of constructing 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 has not to be flipped (value 0); this is due to the fact that in explicit polygons rings 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 that describe the ring explicitly. This criterion coincides with the criterion of ArcInfo. This means that the outer rings cycle clockwise and the internal rings counterclockwise.

Arc id

Index of the arc from the arcs file which is the base of this polygon file.

2.6 Format summary for all file types

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

PNT	
TH	48
CL	16
CL	...
ZH	20
ZD	24
ZD	...
ZL	8
ZL	...

ARC	
TH	48
AH	56
AH	...
AL	16
AL	...
ZH	20
ZD	24
ZD	...
ZL	8
ZL	...

NOD	
TH	48
NH	8
NL	4
NL	...

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

3. Some considerations about the files of Arcs and 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 was 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 an explicit polygon file, 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 where each group has a polygon ID and 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
    4. inner rings contained in the previous outer ring
    5. ...
- Each inner edge counts on the total count of the 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.

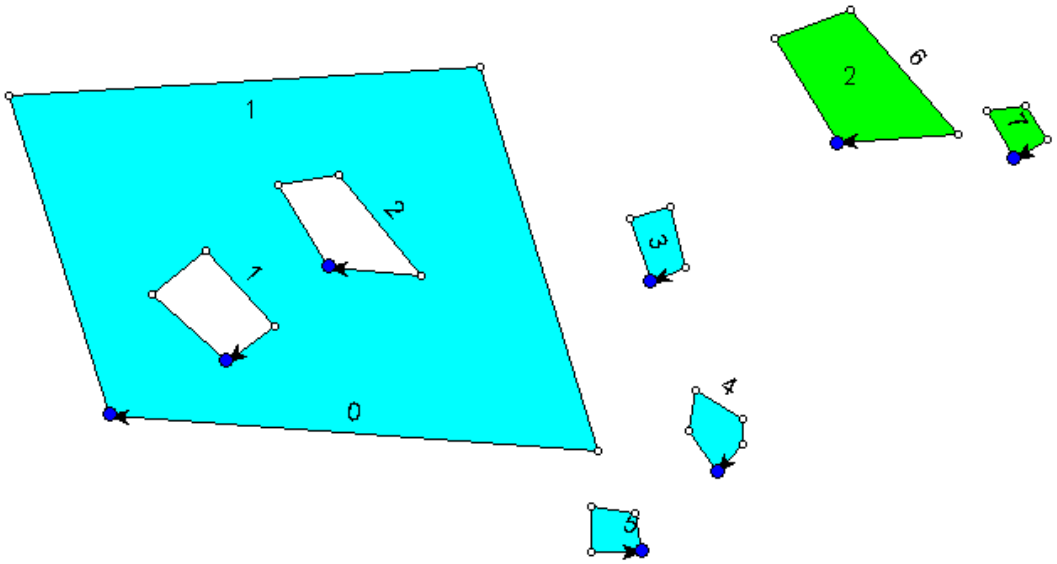
  Note on explicit polygons: A file of **explicit polygons**, whether of groups or not, should be defined as a POL file with the following particulars:
    - In the *flag* of the section TH, bit 0 is not set and bit 5 is set. Bit 3 of this *flag* has the value as is convenient.
    - The section PS (of the polygons of 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 0 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 particulars:

- 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 0xFFFFFFFF (maximum value of type unsigned __int32 value).
- The polygon 0 is formed by 0 arcs and does not contain a PAL section.

4. Illustrative example of a complex polygon

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 index 1 polygon, because polygon zero "cannot be seen" (it is not painted). The polygon on the right part is the one with index 2. The index of the arcs is clearly seen.

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 4 bytes).

- The byte 7 (flag) header of the polygon file described in the section PH will have set, at least, bit 3.
- The 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 the section PH (headers of the polygons) it should be noted that the header of polygon 1 will have 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, the 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