MMZ AND MMZX, OR COMPRESSED MIRAMON MAP, FORMATS SPECIFICATIONS

Authors of the document: Joan Masó and Xavier Pons
First proposal: 22-03-1998
Last modification and version of the document: 05-07-2023. v. **4**

1. Background and Motivation, and Overview

Storing the different parts of a geographical dataset for preservation or dissemination purposes usually requires storing more than one file, links, etc. For example, the format in MiraMon requires a minimum of three files, as well as in the case of the Shapefile format. We could use a single ZIP file to wrap up all those files, but this ZIP file is not easy to build when, for example, multiple layers in different folders are combined; moreover, a conventional ZIP file does not provide explicit information about the structure from the geographical dataset point of view, the entry point to explore its contents, etc. Back in 1997 we were not aware of any format that fit to our needs for distribution or preservation of geographical datasets. That's why in 1997 we started the design of the Compressed MiraMon Map, or MMZ, finishing its first version in March 1998. MMZ added the possibility to include certificates for integrity check and authorship certification, being a requirement for automatic distribution of geographical information. From that moment on, the Government of Catalonia started to distribute most of their environmental datasets in MMZ, along with a free Universal MiraMon Map Reader; in 2016 they still continue to do so. In 2000, this idea won the Möbius International award special mention of the jury for the best science and technology application in the Internet.

An MMZ (pronounced M-M-Z in English) map is a standalone binary file with .mmz extension that contains all the information necessary to visualize, analyze and recover the information related to it. Information in MMZ is usually of the Geographic Information (GI) type: It aggregates raster, vector, related data tables, symbolization and files of any necessary kind, and can also include integrated metadata for datasets, tables, etc. Typically, the information an MMZ contains is aggregated and described in an MMM (MiraMon Map, an uncompressed, usually small, text file) that is also embedded in the MMZ. An MMZ file is a multipart compressed format that takes advantage of the gzip version of the deflate algorithm to compress and decompress the data without losing any information (lossless compression). The MMZ format uses a specifically designed header format to describe the table of contents of the file and some metadata about the embedded parts.

The MMZ format specification is maintained by the MiraMon team, mainly integrated into the Grumets Research Group (Universitat Autònoma de Barcelona and CREAF). Change requests can be submitted to support@miramon.uab.cat for consideration. However, in 2013 the MMZX format has been introduced to deprecate the MMZ. MMZX shares the main characteristics of the original MMZ but is based on an ISO internationally recognized standard. MiraMon continues supporting the MMZ format and has no intention to discontinue the capacity to read and write it in the near future.

The format was initially designed to compress and contain MiraMon maps but, in fact, supports the compression of any kind of MiraMon data as well as the content of a directory. It is also used to distribute compressed packages and updates of MiraMon software. The idea is complemented by an application capable of following all the links present in the different pieces of information related to MiraMon maps and many other GIS formats and to produce a list of dependencies. Once this list is created, each file in the list is read, the relations between them are rewritten to make them compatible with their relative positions in the resulting MMZ file hierarchy, and they are compressed and stored as parts of the MMZ file. In addition, the application is able to certify some files by adding some extra encrypted files with the information about the certification entity. By reading these files, the MiraMon software is able to guarantee the integrity of the original files and well as to acknowledge the authorship.

This format is able to share geospatial information in a single file that can be decompressed and visualized automatically using the MiraMon Universal Map Reader (https://www.miramon.cat/CAT/Prod-LectorUniversal.htm) or the MiraMon Professional software (https://www.miramon.cat/USA/Prod-Professional.htm).

From the users perspective, once the MiraMon Universal Map Reader or the MiraMon Professional are setup up in the computer, the file extension is registered and MMZ files are visualized with a single click giving immediate access to a identical copy of the original one. The information can be combined with other MMZ files and other local GIS files in a single view. The information can also be decompressed to be integrated as common GIS files to be analyzed professionally with GIS tools.

In MiraMon, the creation of an MMZ is done with the MMZ.exe application. In the future, other application could create MMZ files too.

2. MMZ format details

Currently two MMZ versions have been defined. The initial MMZ format was the version 1.0. The MMZ version 1.1 was produced in 2001 and added some extra characteristics to the format.

MMZ was designed for Windows platforms. Byte ordering shall be little-endian and path shall follow Windows operating system rules.

An MMZ file shall start with a package header. The order of the other sections described bellow is not mandatory, only recommended. An MMZ enabled application should be able to read the sections even if they are in a different order. An MMZ file shall not end with an EOF mark.

The use of the .mmz extension is strongly recommended. This does not preclude that other extensions can be used for specify purposes.

The MMZ MIME type shall be application/x-mmz.

2.1. Package header section

All MMZ files shall start with a package header at least 16 bytes long. For the version 1.0, the meaning of this 16 bytes header shall conform to the description of the following table. Version 1.0 could eventually support 64 bit offsets but there is no plans that MiraMon will support that (use v 1.1 or MMZX instead).

Table 1: Package header structure. Version 1.0

Offset	Number of Bytes	Data Type	Content
0	3	ASCII	File type magic number. Shall be “MMZ“
3	4	ASCII	Version form “ 0.0” to “99.0”. In this version shall be “ 1.0”.
7	1	Char	0x00
8	4	int32	Offset in byte of the first part header.
12	4	int32	Reserved. Shall be 0.

Version 1.1 of the MMZ files starts with a package header 32 bytes long. For the version 1.1 and 32 bit file offsets, the meaning of this 32 bytes header shall conform to the description of the following table.

Table 2: Package header structure. Version 1.1. 32 bit offset

Offset	Number of Bytes	Data Type	Content
0	3	ASCII	File type magic number. Shall be “MMz“(cast insensitive)
3	4	ASCII	Version form “ 0.0” to “99.0”. In this version shall be “ 1.1”.
7	1	char	0x00
8	4	int32	Offset in byte of the first part header.
12	4	int32	Reserved. Shall be 0.
16	4	int32	Offset of the path variables section, or -1 if not present.
20	4	int32	Reserved. Shall be 0.
24	4	int32	Offset of the group section, or -1 if not present.
32	4	int32	Reserved. Shall be 0.

Version 1.1 of an MMZ file starts with a package header 32 bytes long. For version 1.1 and 64 bit file offsets, the meaning of this 32 bytes header shall conform to the description of the following table.

Table 3: Package header structure. Version 1.1. 64 bit offset

Offset	Number of Bytes	Data Type	Content

0	3	ASCII	File type magic number. Shall be “MMz“(cast insensitive)
3	4	ASCII	Version form “ 0.0” to “99.0”. In this version shall be “ 1.1”.
7	1	char	0x00
8	8	int64	Offset in bytes of the first part header.
16	8	int64	Offset of the path variables section, or -1 if not present.
24	8	int64	Offset of the group section, or -1 if not present.

2.1. Compressed data section

After the package header, the compressed data parts usually follow. Parts are compressed using the deflate algorithm used in the gzip compression. Compressed parts shall start in offsets multiple of 8.

2.2. Part header section

After the compressed parts, part headers usually follow. The number of part headers shall be the number of compression data parts in the file. Part headers shall start in offsets multiple of 8

Table 4: Part header structure.

Offset	Number of Bytes	Data Type	Content
0	4	ASCII	Shall be “MMZ” + \0
8	4	int32	Offset of the next part header, or -1 if there is no other one[1].
12	4	int32	Reserved. Shall be 0[1].
16	4	int32	Offset of the compressed data for this part[1].
20	4	int32	Reserved. Shall be 0[1].
24	4	int32	Date of creation of the file expressed as a long.[2]
28	1	int8	Hour of creation of the file. If the hour is in UTC, add 100 to the value.
29	1	int8	Minute of creation of the file
30	1	int8	Second of creation of the file divided by 2.
31	1	bit flags	Attributes of the file. See Table 5
32	1	int8	Compression algorithm. See Table 6
33	1	bit flags	Other bit flags. See Table 7.
34	1	int8	Recommended path treatment. See Table 8.

35	1	int8	Reserved. Shall be 0.
36	4	int32	CRC code of the file data.
40	8	uint64	Size of the compressed part in bytes
48	8	uint64	Size of the original file in bytes
56	2	uint16	Size of the name of the file in bytes (does not include a final \0).
58	2	uint16	Size of the comment about the file in bytes, or 0 if not used (does not include a final \0).
60	4	uint32	Size of an extra field in bytes, or 0 if not used (does not include a final \0).
64	variable	ASCII	Name of the file expressed in OEM850 code (it is not null terminated).
var	variable	ASCII	Name of a comment expressed in OEM850 code (it is not null terminated).
var	variable	ASCII	Name of an extra field expressed in OEM850 code (it is not null terminated) [3].
[1] In case of 64 bit, offsets are required: the offset field and the next reserved field are combined to form an int64 number. [2] Dates are expressed as numbers where the 4 first digits refers to the year, the next 2 to the month and the last 2 to the day (e.g. December 31st 1991 is encoded as 19991231 [3] Reserved and never implemented			

Table 5: Attribute bit flags

Bit	Content
1	Read only
2	Hidden file
3	System file
4	Reserved [1]
5	Reserved [2]
6	Archive
[1] Reserved for volume letters but never used. [2] Reserved for folders but never used.	

Table 6: Compression

Value	Content
0	None
8	Deflated as used in gzip

Table 7: Other flags

Bit	Content

1	Entry point[1]
[1] Indicates that the part is an entry point to the data. Typically the MMM file that needs to be used to open the compressed data.	

Table 8: Recommended path treatment

Value	Content
1	Identical to the source
2	Relative to the new position of the data
3	Substituted after the current directory[1]
[1] A substituted path is a full path where the ':' has been replaced by a '$' character (e.g. C:\bob.txt -> C$\bob.txt).	

2.3. Variables of path section

After part headers, the variables of path section usually follow. This section shall be present if the file is in version 1.1 and an offset has been provided in the package header. Variables of path section shall start in offsets multiple of 8. More than one variable of path sections are allowed. This header is rarely used.

Table 9: Variable of path section structure.

Offset	Number of Bytes	Data Type	Content
0	4	ASCII	Shall be "MMZ" + \0
8	4	int32	Offset of the next variable of path section, or -1 if there is no other one[1].
12	4	int32	Reserved. Shall be 0[1].
16	2	uint16	Size of the name of the variable of path in bytes (does not include a final \0).
18	2	uint16	Size of the descriptor of the variable of path in bytes, or 0 if not used (does not include a final \0).
20	2	uint16	Number of candidates of a value for the path variable
22	2	uint16	Size of the first candidate of a value for the path variable (does not include a final \0).
24	2	uint16	Size of the second candidate...
...			
var	variable	ASCII	Name of the name variable of a path in OEM850 code (it is not null terminated).
var	variable	ASCII	Descriptor of the path variable expressed in OEM850 code (it

			is not null terminated).
var	variable	ASCII	First candidate of a value for the path variable expressed in OEM850 code (it is not null terminated).
			Second candidate of...
...			
[1] In case that 64 bit offsets are required, the offset field and the next reserved field are combined into an int64 number.			

2.4. Group section

After variables of path section, group section usually follows. This section shall be present if the file is in version 1.1 and an offset has been provided in the package header. Group section shall start in offsets multiple of 8. Only one group section is allowed. This header is rarely used.

Table 10: Group section structure.

Offset	Number of Bytes	Data Type	Content
0	4	ASCII	Shall be “MMZ” + \0
4	2	uint16	Size of the name of the group in bytes (does not include a final \0).
6	variable	ASCII	Name of the group in OEM850 code (it is not null terminated).

3. Standardization and evolution of the format: MMZX: ISO 19165

In 2016 MiraMon complemented the MMZ file with the MMZX file format that fully follows the OPC standard. OPC (also known as OOXML) is a packaging strategy that integrates elements of the ZIP compression, XML documents, and the web MIME types into an open standard that makes it easier to organize, store, and transport data. It was defined by Microsoft and approved as ISO 29500-2 and ECMA-376. It is used by Office 2007 and newer versions of Word (.docx), Excel (.xlsx), and PowerPoint (.pptx), along with XPS (.xps), Autodesk AutoCAD (.dwfx), etc. An OPC package can contain several files with a directory structure in it. Each file is called a “part” and all OPC package part names have to follow the URI restrictions and conventions. The format adds some extra files to increase interoperability. It incorporates a standardized way of recording file relations or links. It also incorporates a file with a few elements of metadata and a thumbnail with a small image for presentation purposes. All these extra parts allow some basic independent data maintenance, such as the extraction of a fragment of the package, thus guaranteeing that all the related resources will be extracted without needing to understand the actual encoding of the parts included in the package.

OPC is conceptually very similar to the KMZ and MMZ format, but offers some advantages. KMZ is a compressed file that uses a ZIP file format following the solution reuse criteria, but its structure is deeply related to the original KML. Therefore, it does not comply with the completeness criteria, and KMZ strategy is not accurately documented (it is not included in the KML specification) and thus, it does not comply with the formality and metamodel identity criteria. MMZ was never submitted for

standardization or adopted by other vendors and has not reached a high popularity outside the GIS community. OPC uses an accepted popular compression schema and header (ZIP compression) but combines other advantages such as it can be directly manipulated by other OPC compatible tools that recognize the file structure and relations even if they are not able to identify or understand the format of some compressed parts. Phillips and Allemang (2010) prefer it over other alternatives as a file format for data archiving.

OPC format offers an opportunity for designing an standardized way to pack geospatial information in a single file that internally uses the original data files that can be later recovered as they were produced. Due to the similarities between the old MMZ format and the OPC standard, we call this approach MMZX. Most GIS software offer the capability to save a session composed by different information layers, symbolization, presentation, metadata, etc. This "composition" format (e.g., the MiraMon Map, or MMM) does not contain the information, but links to it. Recently the OGC has introduced a new standardized format that also can to this: the OWS context. It is possible to create a software that is able to collect the "context" file plus all the data files and symbolization and configuration files and create an OPC package. Apart from the geospatial data parts of the file, OPC specifies how to explicitly relate different parts using .rels files (relations are composed of a source file and a target file, and the purpose of this relation is like in the RDF format). These files are XML files that have the same name as its respective source part adding ".rels" and placed in a "rels" folder. Each of these files lists the target parts related to its source and the semantics of this relation.

A complete description of the MMZX is available in this open access paper:

- Pons X, Masó J (2016) **A comprehensive open package format for preservation and distribution of geospatial data and metadata**. *Computers & Geosciences*, 97:89-97. DOI: 10.1016/j.cageo.2016.09.001. http://dx.doi.org/10.1016/j.cageo.2016.09.001.

In 2018, ISO 19165 standard, "Geographic information -- Preservation of digital data and metadata", has been approved, and the MMZX has become the first implementation of it.