

Migrating a Paradox database

with type 2 and 3 blob fields in DB-MB file combinations

Concept v1 – 20250208 rev. 5/JWT

Introduction

A dear friend of mine asked a couple of weeks ago my assistance in his efforts to export data with so called memoBlob fields in a Paradox database using a DB and MB file combination. His goal was to develop a C# viewer to show the genealogy contents of the Paradox files to his relatives. In his search for more information how to find ways to program this viewer, he found a [CodeProject publication by mr. Petr Bříza](#) which offers an excellent opportunity to read and export Paradox database files with C# coded applications. The C# classes he designed, works really great as long as the Paradox database you're using doesn't have any (memo) Blob or Graphic fields stored in MB files.

That part of his code was just not finished. In my efforts to support him I found on GitHub two forks of his code, with one comment to cover this subject but both forks did not offer a real solution. In addition to this, the internet offers very few sites with (free) solutions to process such MB files correctly. Those who offered free code with well-functioning paradox readers like the PdxEditor website of mr. Knabe, responded not to our inquiries for more information or help, maybe because their email addresses were not active for long period of time. We also found a Python wrapper around the C library pxlib on GitHub.

As there is apparently still a need for any open source C# or other code like Python or Golang that can handle Blob fields, we decided to give it a try ourselves.

The most difficult part of our project was to really understand what mr. Kevin Mitchell had written in his document 'Paradox 4.x File Formats, Revision 1, May 11, 1996' or in that other document 'PxFormat.txt', compiled by Randy Beck (<http://www.randybeck.com>). The 'PxFormat' version we used was from Dec 7, 2006 and contained also an example program written in Pascal.

Our understanding of the chosen file structure was also hampered by our disbelief as to why Borland's Paradox developers had come up with such, what we think, an unnecessarily complicated structure. Unexpected index inversions, applying low- and high-endian randomly, it looked as if they were afraid for their database design being stolen. But, one must realize that at the time that Mr. Mitchell and Randy Beck re-engineered Paradox, PC's worked with 16 bit and later 32 bit CPU's and what was then called 'long' consisted of only 4 bytes. Nowadays we have 64bit CPU's and do we speak of an Int32 number(s) and so forth.

Restrictions

As we were only interested in retrieving and exporting the (memo)Blob data, we did not look at the Paradox index and PX files way of retrieving the data. The reason for this is, that the investigated Paradox database had no index files.

Investigation and development process

During our search on the internet I used a commercial program during an evaluation period to export the data from the DB file, while it had no solution for the (memo)Blob in order to verify the data export of my prototype besides using the hex-viewer in my debugger.

Besides the code modifications in Petr Bříza C# application done by my friend, I tried to support him by building a prototype using the [Winbatch](#) tool from the ground up to investigate and test the

Migrating a Paradox database

with type 2 and 3 blob fields in DB-MB file combinations

Concept v1 – 20250208 rev. 5/JWT

documented way by Keith Mitchell to get the memoBlobs out of the .MB file by trial and correcting our interpretation errors of the two above mentioned documents.

In any case, a careful and accurate investigation of both Mr. Mitchell's and Beck's findings was sufficient to obtain in the end well-functioning results. Hats off to both! Since we didn't have the tools to create and edit Paradox tables ourselves, we were limited in our tests to a very limited amount of public available tables that we could find on the internet. We found later a couple on the internet.

In the following paragraphs, we will apply what both above mentioned documents told us about the structure of DB and MB file combinations. In such a combination, the MB file is very different from the format of the DB and PX (index) files. The blocks are not chained and the block length varies. The header block in the .MB file is always 4k (1000h) bytes long. Blocks that follow the header block could have a length that is a multiple of 2 or 4k and containing the information from the memo fields in the DB file. Memo Blob fields are also known as Blob fields.

(Memo)Blob fields in the DB file

According to the two documents a (Memo)Blob field in a DB record could be of a variable length with a leader buffer M with a maximum of 240 bytes followed by a buffer 10 bytes. The buffer at the end is used to hold the blob's length, its location in the MB file, and a modification number (used internally by Paradox). In general, MemoBlobs fields should be defined as M1 to minimize record size with a length of 11 bytes in total.

A M1 MemoBlob field takes 11 bytes in the record as is the case in our test Paradox database *gdpprs.DB*: the three used MemoBlob fields have a length of 11 bytes.

The 11 bytes represent:

0000h	The leader byte M1
0001h	An index value of 00–3Fh (64d) for a Type3 MB file block and FFh for a Type2 MB file block.
0002h..0005h	An unsigned long integer (32 bits) that contains the offset of the blob's data block in the MB file
0006h..0007h	An unsigned long integer that contains the length of the (memo) blob.
0008h..0009h	An unsigned short integer (16 bits) that contains the modification number from the MB file header.

The bytes are in the Little Endian format

Example from the first MemoBlob field in the first DB record:

3C 3F 10000000 0300 000100

giving

- the leader M1 byte 3Ch
- an index value of 3Fh (63d) of the array of entries in the Type3 MB file block
- an offset of 4096d in the MB file
- a length of 768d for the memo text in the MB file

An other DB/MB file combination called *album.DB/album.MB* contains records with one memoBlob field with a length of 250 bytes, Blob field and a Graphic blob field of both 11 bytes long.

Migrating a Paradox database

with type 2 and 3 blob fields in DB-MB file combinations

Concept v1 – 20250208 rev. 5/JWT

MemoBlob and Blob structure in the MB file

According to Mr. Mitchell's document, there are four types of MemoBlob or Blob blocks:

- Type0: the header block
- Type2: Single block
- Type3: Suballocated Block
- Type4: Free block

The Type1 block is not specified and we did not encounter this Type of block

Each block starts with three bytes:

0000h Block type (in the Mitchell document it is called a Record type)

0001h- Number of 4k chunks in this block.

0002h The maximum size is FFFFh x 1000h = 65,535 x 4096. This is 256 megabytes (the maximum length of a blob).

We describe only two of them: the Type2, containing single larger text blobs for blobs longer than 2048 bytes. Type3 used for multiple small text blobs. Used to hold up to 64 small (under 2k) blobs.

The first occurring memoFields in the Paradox DB file *gdpprs.db* pointing at Type2 and Type3 blocks in the Paradox MB file *gdpprs.mb* are selected hereafter to explain how the corresponding data might be retrieved from a Paradox database.

Type2 blocks

Description of the Type2 block: A single blob block can appear anywhere in the MB file. A "long" blob is stored in this kind of block. The block length is the smallest multiple of 4k that is greater than or equal to the length of the blob.

The Type2 block header contains the following fields:

0000h Record (Block) type = 02h (block contains one blob)

0001h- Size of block divided by 4k

0003h- Length of the blob.

0007h- Modification number. This is reset to 1 by a table restructure.

0009h- Blob data starts here

Example: The first 11_byte MemoBlobArray which occurs in the DB file *gdpprs.db* and points at a Type2 block is 3C FF 40000022 0E00 000100'.

It tells us that with a value of FFh it is a Type 2 block with an Offset (4000h/16384d) in corresponding MB file *gdpprs.mb* a blob with length 3618d can be found:

Also the byte value at address 4000h/16384d in the MB file is 2, indicating the start of a Type2 block.

Type3 blocks

A Type3 block Suballocated block can appear anywhere in the MB file. Up to 64 short blobs may be stored in this type of block. A suballocated block is 4k bytes long. It has a 12-byte header followed by an array of up to 64 5-byte blob pointers (or entries).

The DB field that "owns" a blob contains the offset of the block (from the start of the MB file) and the index of one of the entries in the blob pointer array. The array entry points to the blob data.

Migrating a Paradox database

with type 2 and 3 blob fields in DB-MB file combinations

Concept v1 – 20250208 rev. 5/JWT

The 12-byte Type3 block header contains the following two fields:

0000h Record type = 03h (Suballocated block)

0001h Size of block divided by 4k, value of 1 because the block size is 4k

The blob pointer array of entries follows the header starting at position 00012h. The array has 64 entries numbered from 00h to 3Fh. Entries are used in **reverse** order. The 3Fh entry is used first. Then the 3Eh entry, then 3D, and so on.

The offset (from the start of the Type 3 block) of the entry indexed by i is calculated as follows:

offset = 12 + (5 * i)

Each entry is 5 bytes long and has the following format:

0000h Data offset divided by 16. The offset is measured from start of the Type3 block. If this offset is zero, then the blob was deleted and the space has been reused for another blob (which is associated with another entry in the array).

0001h Data length divided by 16 (rounded up)

0002h- Modification number from blob header. This is reset to 1 by a table restructure.

0004h Data length modulo 16. If this is zero, then the associated blob has been deleted and the space can be reused.

For an active blob, this value will be between 01h and 10h

Example: The first 11_byte MemoBlobArray which occurs in the DB file gdpprs.db and points at a Type3 block is 3C 3F 10000000 0300 000100.

It tells us that with an Offset (00001000h/4096d) in the corresponding MB file gdpprs.mb, a blob with length 0300h/768d can be found with the Array of entries index 3Fh/63d:

The byte value at address 1000h/4096d in the MB file is 3 indicating the start of a Type3 block.

If the ArrayofEntries[63] looks like 15-30-01-00-10, then the blob offset should be calculated as follows:

Offset in current Type3 block = $16 * 0x15 = 16 * 21d = 336d$.

The data length = $0x10 * 0x30 = 768d$, happens to be equal to the value the value in the 11_byte MemoBlobArray.

The Data length modulo value in the 4th byte has a value xx, giving different Data Length.

Our conclusion: The derived data length value from the 11_byte MemoBlob should prevail.

Remarks

During the prototype development I stumbled on the following observations.

Database field *Timestamp*: from the description of Mitchel and also from mr Knabe (<http://nknabe.dk/database/pdxeditor/help/index.html?paradox-table-format.html>) the Timestamp is stored as double precision floating 8-bytes (Stored as Float value: Integer part as days since 0001-01-01 and fractional part as fraction of a day in milliseconds). In the end after converting it to floating point value we concluded that this value represented the number milliseconds since 0001-01-01 and it was also stored in the Little Endian format. Also you have to subtract one day from the total number of days. This is also the case in C# code of Petr Bříza.

Migrating a Paradox database

with type 2 and 3 blob fields in DB-MB file combinations

Concept v1 – 20250208 rev. 5/JWT

Both the used DB/MB file combinations were not very large, so in my Winbatch prototype script I just read the DB and the MB file in two separate memory buffers (or arrays). If you are exporting larger files with thousands of records may be you should read the files block by block.

Limitations and conclusion

We had only one Paradox DB-MB table combination available for testing. This combination contained a lot of Type3 memo blocks, but only 2 of Type2.

We also had no (graphical) Blobs in the test MB file like pictures or music parts.

The album.DB/album.MB Paradox combination had several memoBlobs with a length of 250 bytes, and also a Blob and Graphic fields, both with a length of 11 bytes.

Our conclusion: It is not a KISS design (Keep it Simple and Sound) and other user experiences stated that these Paradox databases can easily be corrupted. So we did not try to update or insert data records or any memoBlob data.