MAIN TPS-FILE

ORGANIZATION

Header

```
     (+0)        (+4)          (+6)           (+0Ah)          (+0Eh)
 00 00 00 00    00 00      00 05 00 00     00 05 00 00     74 4F 70 53
UNSIGNED LONG   USHORT    UNSIGNED LONG   UNSIGNED LONG   UNSIGNED LONG
  Header        Header      File size       File size    TOPSPEED file label
  offset         size

 (+12h)      (+14h)          (+18h)               (+1Ch)
 00 00     00 00 00 06     07 00 00 00           00 00 00 00
 SHORT    UNSIGNED LONG   UNSIGNED LONG         UNSIGNED LONG
 zeros       Last       file modification     Upper control page
  (?)       number     (alteration?) counter   (address minus  200h,
                                                divided into 100h)
```

Then there follow two arrays which have the length = (header_length-20h)/2 of 4-byte integers (in what follows byte order is reversed):

```
0020h:   00000000  00000000  01000000  04000000  4D000000  22010000  22010000


0110h:   00000000  01000000  03000000  4C000000  22010000  22010000  22010000

            ^                                                 ^
 this pair of elements                           this is  (file_length-200h)/100h,
      is ignored (?)                      this pair of numbers fills the rest of the array
```

Pages in the file are organized in blocks and space between them is not used. The element number i in the first array refers to the beginning of each block, the element number i in the second array refers to the end of the block. The element number i is the offset of the block (of the first byte on the first page for the first array and of the first byte of the page after the last page in the block for the second array) minus 200h (header size), divided into 100h. All pages in the block - except, may be, the last one - are compressed, if this is possible.

If a page inside a block is not compressed but can be compressed, then this block is divided into two parts so that the uncompressed page is in the end of the (first?) block. Then, the following construction occurs in these arrays (in the example below the uncompressed page is located at offset 0200h and its size is 100h):

```
0020h:   00000000    00000000    01000000

0110h:   00000000    01000000    22010000
                         ^           ^
```

Unused space can’t be located in the end of file. If this happens, then the file is shortened (it is cut with the function int 21h, ah=40h, cx=0).
It is not known whether the header could be longer than 200h and if so, what would happen then with these arrays (most likely, their size is simply increased).

Format of a standard page

```
  (+0)                  (+4)                   (+6)             (+8)
 00 02 00 00            73 00                  77 00            7F 00
   ULONG                USHORT                 USHORT           USHORT
   Page             Compressed page         Page length     Page length after
  offset          length (if page is           after          decompression
(for checking)     not compressed, then    decompression       without any
                  the next field is repeated)                  shortening
                                                             (abbreviation?)

   (+0Ah)            (+0Ch)                        (+0Dh)
   0A 00               00                            05
   USHORT             BYTE                          BYTE
 The number of     standard page       first duplicator block offset
  records on          sign(?)        This byte exists only if the page is compressed,
  this page        (page level)         that is the field with offset 4 is not equal
                                           to the field with offset 6

   (+0Dh/0Eh)
```

...
Then there follow records on the page

The pages in the file have variable length. The page length depends on file driver. If after data addition/modification it occurs that a page is too long, it is divided into two pages.
Unused space after the page up to the next divisible by 0x100 offset is filled with 0xB0 byte and is not counted within the page length (fields +4, +6, +8), butt (sorry, brother) is reserved for this page and is marked in the header as belonging to this page.

Control page format

(+0)	(+4)	(+6)	(+8)
00 02 00 00	73 00	77 00	7F 00
ULONG	USHORT	USHORT	USHORT
Page offset (for checking)	Abridged page length	Abridged page length	Unabridged page length

(+0Ah)	(+0Ch)	
0A 00	00	
USHORT	BYTE	There is no +0Dh offset byte because control pages are not compressed (?)
The number of records on page	Control page level (0-standard page)	

(+0Dh)		
00 00 00 00	05 00 00 00	...
ULONG	ULONG	

Slave page array. Its size is equal to the number of records on the control page.
Array element is (slave_page_offset-200h)/100h

(+?)
...
Then follow the records. They repeat the first records on slave pages (eventually abridged, if slave pages are not control pages). To each record there corresponds an element in the slave pages array.

Compressed pages

If a page is compressed, then the page header fields (+4) and (+6) are different. In this case the byte with offset (+0Dh) indicates offset over the byte (+Eh) of the first duplicator block. The format of such a block will be:

00	05	03
BYTE	BYTE	BYTE
Which byte Should be repeated	The number of repetitions minus one	Next duplicator block offset

Next duplicator block offset is the offset of the next such block over the byte, which is the last in this block. If this block is the last one, then the offset of the next block refers to the last byte of the page (not to the byte 0xB0 which follows the page).
If the number of repetitions > 127, directly after this byte there follows another one:

3E	85	10	03
BYTE	BYTE	BYTE	BYTE
Which byte To repeat	First byte Number of repetitions minus one	Second byte	Next block offset

Then the number of repetitions is calculated according to the formula:

```
((second_byte + ((first_byte & 7F) << 1)) >> 1) + 1
```

If the offset of the next block > 127, then directly after this byte there follows another one and the offset over the last byte of the block is calculated according to the formula:

```
(second_byte + ((first_byte & 7F) << 1)) >> 1
```

The page header is not compressed. Pages are usually (normally?) compressed. However, the page is not compressed if it can’t be compressed (its length after the compression should strictly be less than the uncompressed page length). If the page is not compressed, but can be compressed, it is mentioned in the header. If the page can’t be compressed, it is not mentioned in the header.

```
                      General format of the record on a page

      (+0)            (+1)          (+1/3)          (+1/3/5)
       C0             2A 00         09 00            . . .
  Identifier       New length     New length of
                 of the record    the record on       Data
                                  control page
```

```
   Identifier format:            1 1 0 0 0 1 0 0
                                 ¦ ¦ L----T-----
The presence of the new length --- ¦      ¦
        of the record              ¦      |
                                   |   How many initial bytes
The presence of the new length -----   of this record should be
of the record on control page          taken from the previous record
```

New record length is present only if (identifier & 0x80) <> 0, otherwise, the length is taken from the previous record. The length of the new record on control page is present only if (identifier & 0x40) <> 0, otherwise, the length is taken from the previous record. For the first record on the page both of these values should be present, that is (identifier & 0xC0 = 0xC0). The last six bits of the identifier indicate the number of bytes copied from previous record.

```
                             General format of a file

 ----   Header  ------¬  ---->---   Block 1
 ¦ ...                ¦  ¦    ¦--  page 1
 ¦    upper page      ¦  ¦    ¦¦  .....--
 ¦ ...                ¦  ¦    ¦L-
 ¦                    ¦  ¦    ¦--   page 2
 ¦ 020h:  00 00 00 00 ¦  ¦    ¦¦  ...........------
 ¦    beginning 1 --------    ¦L-
 ¦    beginning 2 ---------¬  ¦...
 ¦    beginning 3 -------¬ ¦  ¦--   page M
 ¦        ...         ¦  ¦ ¦  ¦¦  ..........---
 ¦ 110h:  00 00 00 00 ¦  ¦ ¦  ¦L-
 ¦        end 1  --------+-+->L----
 ¦        end 2  --------+ ¦       Unused space
 ¦        end 3  ------¬ ¦ L->---  block 2
 ¦        ...         ¦¦ ¦    ¦--  page 1
 L--------------------¦  ¦    ¦¦  ...
                      ¦  ¦    ¦L-
                      ¦  ¦    ¦--  page 2
                      ¦  ¦    ¦¦  ...
                      ¦  ¦    ¦L-
                      ¦  ¦    ¦...
                      ¦  ¦    ¦--  page N – not decompressed
                      ¦  ¦    ¦¦  ...
                      ¦  ¦    ¦L-
                      ¦  ¦    L----
                      ¦  L--->---  block 3
                      ¦       ¦--  page 1
                      ¦       ¦¦ ...
                      ¦       ¦L-
                      ¦       ¦...
                      ¦       ¦--  page K
                      ¦       ¦¦ ...
                      ¦       ¦L-
                      ¦       L----
                      L----->  Unused space
                              --- Block L
                              ...

                              L----
                              End of file
```

```
   -- Header -------------------¬
   ¦  . . . . . . . . . . . .   ¦
 ---<  upper level page         ¦
 ¦ ¦  . . . . . . . . . . . .   ¦
 ¦ L-----------------------------
 ¦
 ¦    Level N
 ¦    page                      level N-1
 L->T-----------------¬         pages                       Level zero
    ¦  slave page 11 -------->T-------------------¬         pages
    ¦  slave page 12 ------¬  ¦slave page 21 ------->...->--------------------¬
    ¦ ...             ¦    ¦  ¦slave page 22      ¦     --->recordN1= record11|
    ¦                 ¦    ¦  ¦ ...               ¦     ¦ ¦ recordN2          ¦
    ¦-----------------¦    ¦  ¦                   ¦     ¦ ¦ ...               ¦
    ¦ record11 ------------+-¬¦-------------------¦     ¦ L--------------------
    ¦ record12 -----------¬¦ L->record21=record11 ------- ...
    ¦ ...             ¦   ¦¦  ¦ record22          ¦
    L------------------   ¦¦  ¦ ...               ¦  ...->--------------------¬
                          ¦¦  L--------------------       ->recordM1          ¦
                          ¦¦                              ¦ recordM2          ¦
                          ¦¦                              ¦                   ¦
                          ¦L->--------------------¬       ¦                   ¦
                          ¦   ¦ slave page 31 ------>...  L--------------------
                          ¦   ¦ ...               ¦       ...
                          ¦   ¦-------------------¦  ...->--------------------¬
                          L---->record31=record12---------->recordK1          ¦
                              ¦ ...               ¦       ¦ recordK2          ¦
                              L--------------------       ¦ ...               ¦
                              ...                         L--------------------
                                                          ...
```

All records are sorted according to the lexicographical order.

All records in a file are sorted according to the lexicographical order. This is achieved by sorting out records on each page, including control pages. Records on control pages repeat the first records on the corresponding slave pages (or slightly shortened). The organization of a TPS file is very similar to that of key/index files of the format CLARION 2.1.

Note: to all appearances, there should be neither in a file nor on any page two identical records (see records of keys/indexes and DUP attributes).

While searching for the record one goes through the chain of pages starting from the upper control page down to zero level page, one page on each level. The search on a page starts with the beginning of the page: it is searched the last record which is lexicographically less than the sought one. Then, for the found record and for every record, the first N symbols of which coincide with the sought record (N is the length of the sought record), the record is searched on the slave page (or is assumed to be found, if this is a zero level page). If the first record on page is lexicographically greater than the sought one, then the sought record is absent on both this page and the slave pages. If the record on the zero level page is less than the sought one and after the last record, the first N symbols of which are less than the sought one, there follows the record, which is lexicographically greater the sought one, then the sought record is absent both on this page and in the whole file.

While adding a new record to the file one goes through the chain of pages, which has the length M, where M is the level of upper control page, one determines the place where to insert the new record, so that the order of the sorting would not be disturbed.

DATABASE LOGICAL ORGANIZATION

TPS file can contain several tables of data and data, keys/indexes and memo-fields for every table.

Data types and formats

00-F2	- keys/indexes
F3	- data bases records
F6	- information record, contains the number of records of the given type
FA	- table description
FC	- memo-fields data

Empty record

The first record in a file is always the empty record: C0 00 00 00 00

Data base record

```
                                                  (+0)            (+4)
     C0           3F 00         09 00          00 00 00 01         F3
    BYTE          USHORT        USHORT            ULONG           BYTE
 Identifier      record     record length on   table number    record code -
                length+9    control page, is                   DB record
                            always equal to 9

        (+5)           (+9)
     00 00 00 1B       ...
        ULONG         data
   Record number,
   unique number
   for each record
```

While adding a new record, the last record number in the file header is increased by one (firstly, the last byte is increased, as in Motorola processors), and then is given to the new record. In such a way uniqueness is achieved. While the file is created the DOS number of the last record is established to 1 (in Motorola format). (The last phrase is ambiguous, may be it should be read as "While the DOS file is created the number of the last record is established to 1" – *A.Z.*).

Record length on control page is always equal to 9.

Key/index record

```
                                        (+0)                 (+4)
      C0          19 00     19 00    00 00 00 01              01
     BYTE         USHORT    USHORT      ULONG                BYTE
  identifier                         table number       record code,
                                                      in the same time
                                                      key serial number
       (+5)                    (+?)
   80 02 80 05 05  ...     00 00 00 1B
                               ULONG
     data               DB record number
```

Sorting in the key/index is done because of the necessary (obligatory?) sorting of all records in the file. The length of this record on control page is equal to the length of the record on ordinary page, if key has DUP attribute, otherwise it is less than it by 4 (of course, TPS file should not contain two identical records).

Note: if the number of the record in the index refers to a nonexistent DB record (if it was manually changed), then it may happen that during the next construction of the index by Clarion Database Manager it will not be deleted (why?).

Memo-field record

```
                            (+0)           (+4)             (+5)
  C0   0C 01    0C 00    00 00 00 01        FC           00 00 00 02
 BYTE  USHORT   USHORT      ULONG          BYTE             ULONG
                         Table number   record code-    DB record number,
                                          memo data    to which this memo-field
                                                            belongs

          (+9)                  (+0Ah)               (+0Ch)
           00                    00 01                ...
          BYTE                   USHORT
    Memo-field serial       memo block number,    memo-field data
         number             the last byte is
   (a record can have       changed first
    more than one memo-fiald
```

Memo data form blocks of 256 bytes (the last block may have less). Block number is indicated in field (+0Ah) (the last byte of which is changed first). The length of such a record on control page is always equal to 12.

Information record

```
                            (+0)           (+4)               (+5)
  C0   0E 00   06 00    00 00 00 01         F6                 01
 BYTE  USHORT  USHORT      ULONG           BYTE               BYTE
                        Table number   record code -     code of the records, to which
                                        information      the information belongs:
                                           record        00-F2 – keys/indexes
                                                         F3    - data

        (+6)                (+0Ah)
    05 00 00 00         00 00 00 00
       ULONG               ULONG
  the number of       the record, which
   records with        was accessed
  this (the same?)
      code
```

For each key/index and table data one record of this type is created. The code of the records, for which this record is created, is indicated in the field with offset +5. The length of this record on control page is always equal to 6.

The record which, was accessed is equal to 0, if the index does not need to be rebuild and contains the number of DB record, which was first accessed after the last index rebuild. It is always equal to 0 for keys and table data.

Table structure description

```
                            (+0)         (+4)        (+5)         (+7)
  C0   1A 00   07 00    00 00 00 01       FA         00 00        01 00
 BYTE  USHORT  USHORT      ULONG         BYTE        USHORT       USHORT
                        Table number record code -  description   minimal driver
                                      structure      block        version (for
                                     description     number       work)

  3F 00      07 00    01 00         07 00
  USHORT     USHORT   USHORT        USHORT
record      number of  number of   number of
length in    fields    memo        keys/indexes
 table
```

Structure description data form blocks of 512 bytes (the last block may have less) (like memo). The block number is indicated in the field (+5). The length of each record on (of?) control page is always equal to 7. The minimal driver version for work with file is in the field (+7):

1 = TopSpeed 1.0 from Clarion 3.1
2 = TopSpeed from Clarion for Windows 1.5

Fields description

```
  --------------

   (+0)          (+1)           (+3)         (+n)           (+n+2)
    12          00 00         FIELD1 00      01 00           14 00
   BYTE         USHORT         CSTRING       SHORT           SHORT
   field      fields offset    field     number of       size of the whole
   type     with respect to    name    elements in array     array
         the beginning of the
              record

        (+n+4)               (+n+6)
         00 00                01 00
         USHORT               USHORT
            ?
   equal to 1, if this      serial number of the field
  field overlaps another        in the record.
  field (OVER attribute);
  equal to 0 otherwise.
```

Field (+n) contains 1, if this field is not an array, [and] the length, if it is an one-dimensional array, and the product of lengths off all dimensions for a multi-dimensional array. The next field contains the size of the whole array, that is, the length of one element multiplied by the value of the field (+n) (it seems that the author of the Russian text made a mistake – A. Z.)

If a file has a prefix, it is indicated in every name of a field, e.g.: “TST:FIELD1”. If it has no prefix, it is simply written “FIELD1”. After the creation the prefix of the fields is not taken into account.

Note: If an arbitrary value, which is not equal to 0 is put in the field (+n+4), that will act in the same way as one. But standard tools put 1.

Fields types
----

type	size	title	description
01	1	BYTE	unsigned number
02	2	SHORT	signed number in Intel 8086 format
03	2	USHORT	unsigned SHORT
06	4	LONG	signed number in Intel 8086 format
07	4	ULONG	unsigned LONG
08	4	SREAL	format “single” of Intel 8087 coprocessor
09	8	REAL	format “double” of Intel 8087 coprocessor
0A	?	DECIMAL	additional data:

```
                    (+n+8)                (+n+9)
                      02                    05
                     BYTE                  BYTE
             Number of digits after   the size of one element of
               the decimal point           the array
```

DECIMAL is a number with fixed point in BCD format, one byte contains two digits, the elder nibble of the elder byte contains the sign (0 – plus, other - minus).

Note: Clarion uses 0xF as minus. The elder byte is stored first (with the least offset).

Data size is calculated according to the formula:

(the_number_of_digits_defore_the_point + the_number_of_digits_after_the_point)/2 + 1

The maximal length is 16 bytes.

```
 12     ?      STRING      additional data:

                              (+n+8)        (+n+10)
                              14 00          00 00
                              USHORT           ?
                         The size of one    pattern
                       Element of the array   picture
```

If the described field is an array, then (+n+8) is the size of one of its elements. If a string has a pattern, it should be indicated in the field (+n+10), without @, and trailing with zero. If it has no pattern, then the field (+n+10) contains two bytes: The first one is zero, the second is an arbitrary number (Clarion 3.1 writes 0). It is not clear why there are two bytes.

```
 12     ?      PICTURE       additional data:

                              (+n+8)           (+n+10)
                              09 00         p####-####p 00
                              USHORT            CSTRING
                          The size of one       picture
                          element of the array

 13     ?      CSTRING     additional data: see STRING or PICTURE
                           CSTRING – a string trailing with zero.

 14     ?      PSTRING     additional data: see STRING or PICTURE
                           PSTRING – a string, the first byte of which is its length
                           The size of the field PSTRING is the maximal length of the
                           string + 1.

 16     ?      GROUP       no additional data.
```

GROUP is represented as a field, which is independent from its nested fields.

It simply overlaps other fields (using offset and size notification).

A number is ascribed to it as to other fields, independently from other fields.

It is located directly before slave fields, which follow it. Numbers are ascribed to slave fields as if they aren’t included in GROUP.

If a group has a prefix, it (this?) is indicated in slave fields names and substitutes the file prefix, e.g., "GRP:FIRST_FIELD".

If a group has no prefix, then its fields have the same prefix as the main file.
If the main file has no prefix either, then the group fields have no prefix at all.
After the creation, the prefix of the group fields is not taken into account.
While creating GPOUP array (array GROUP?), its size is indicated in the (+n) field in the description of the field GROUP. There are no more references to the fact that this is an array. The offset for slave elements is indicated as for the first element of the array. The slave field from the first element of the array can even be a part of a key.

```
  Memo description
  -------------

        (+0)               (+n)          (+m)        (+m+2)
     DATA.MEM 00       FILE_MEMO 00      10 27       01 00
       CSTRING            CSTRING        USHORT      USHORT
  The name of the      memo-field      memo-field   attributes
  external file for       name            size
       memo
```

If there exists external file name, it is stored in the (+0) field and trails with zero. If the external file name is absent, then the (+0) field contains two bytes: the first is zero and the second is arbitrary (Clarion 3.1 stores 1).
Example:

```
         (+0)          (+n)           (+m)     (+m+2)
        00 01      FILE_MEMO 00      10 27     01 00
```

Note: A program written with Clarion 3.1 stores 1 in field (+m+2), if the memo-field does not have BINARY attribute, and 2, if it has. Clarion Database Manager 3.1 always stores here 1. Neither the program, nor Database Manager recognizes this field (here an arbitrary value can be stored).
For the driver version 2 (Clarion for Windows 1.5) the attribute byte is constructed in the following way:

```
        0 0 0 0 0 1 0 1
                  ¦ ¦ L-- always 1
                  ¦ L---- 1 = There is a BINARY attribute
                  L------ 1 = this is a BLOB, 0 = this is a MEMO
```

The memo-field length for a BLOB is 0.

Note: While importing the structure of a file, CfW 1.5 never takes the attributes byte into account.

```
   Key/index description
  ----------------------

   (+0)       (+n)      (+m)     (+m+1)         r=============================¬
   00 01     KEY1 00     21      02 00          ¦     KEY/INDEX ATTRIBUTES    ¦
     ?       CSTRING    BYTE     USHORT         ¦                             ¦
external     key    attributes   the number     ¦      0 0 1 0 0 0 0 1        ¦
file name,   name                of fields in   ¦        L-+      ¦ ¦ L- DUP  ¦
see memo                         a key          ¦  0 = KEY        ¦ L--- OPT  ¦
                                                ¦  1 = INDEX     L-- NOCASE   ¦
                                                ¦  2 = Dynamic index          ¦
  Then, for every field in key/index             L=============================-
there follows a record


    01 00     00 00
    USHORT    USHORT
    Field   attributes      Attributes: 0    = ASCENDING
    number                              non 0 = DESCENDING
```

For Dynamic index the number of fields is always equal to 0. It is unknown, where the Dynamic index data are stored. If the attributes are not equal to 0(any arbitrary number), then the field is considered as DESCENDING.
Note: If 00 00 is contained in the field (+0), then Clarion 3.1 while importing the structure assumes that the key/index has an external name "". Other values of the second byte (00 02, 00 03 etc.) are correctly recognized. The value 00 00 for memo-fields is correctly recognized.

Table title

The last records in a file are titles of the tables

			(+0)	(+1)	(+n)
C0	0C 00	08 00	FE	UNNAMED	00 00 00 01
BYTE	USHORT	USHORT	BYTE	STRING	ULONG
	Record length	record length in control page	indicates that this is a table name	table name	file code for this table

Byte with offset (+0) indicates that this record is a table name. No table number should begin with this byte. Partly for this reason while augmenting the record number the last byte is changed first. The length of table name is calculated as the_length_of_the_record minus 5, and the length of the record on control page as the_length_of_the_record minus 4.

While the table is constructed by means of the CREATE function its number is calculated as the number of the next record (the number of the last record in the header is augmented by 1, is inscribed back into the header and is considered as the number of the table).

Fields representation in keys and indexes

ASCENDING

BYTE, STRING, PICTURE, GROUP — Are not changed, GROUP is considered as STRING, even the bytes of the numbers stored in it are not swapped.

CSTRING — the unused bytes on the right hand side are replaced with zeros

PSTRING — The byte of length is not indicated, unused symbols on the right hand side are replaced with zeros, one more zero is added on the right hand side in order that the length of the string in the key would be equal to the length of the string in the table record.

USHORT, ULONG — Bytes are swapped in the reverse order

SHORT, LONG — The elder bit of the elder byte is inverted, bytes are swapped

REAL, SREAL — If the number is positive, then the elder bit of the elder byte is inverted. Otherwise, all bits are inverted. Bytes are swapped.

DECIMAL — If the number is positive, then the elder bit of the elder byte is inverted. Otherwise, the older nibble is equal to 7, and the other bits are inverted.

DESCENDING

BYTE, SHORT, LONG, USHORT, ULONG, REAL, SREAL, STRING, CSTRING, PSTRING, DECIMAL — The same as ACSENDING, but every bit is inverted

Note: if the number DECIMAL is negative, then the elder nibble is equal to 8, all other bits remain unchanged. We actually change the sign of the number, then construct as for ASCENDING.

Note: if the number REAL/SREAL is negative, then bytes are simply swapped. We actually change the sign of the number, then construct as for ASCENDING.

Commentary

RECLAIM, CREATE attributes don't change the file content, as well as the procedures LOCK, UNLOCK, HOLD, RELEASE.

It is not altogether clear, for what purpose two lengths of the file are used in file header. In processing of transaction file header is copied just after the last page of the file (that is, into the address indicated in the field (+6)). After the header, there follow some other data (pages). In this new header the field (+6) remains the same, and the field (+0Ah) indicates the new length of the file together with the recorded data. That is, the first of the lengths is the length without the "unused" space in the end of the file.