SAS7BDAT Database Binary Format

by:

Matthew S. Shotwell, PhD
Assistant Professor
Department of Biostatistics
Vanderbilt University
matt.shotwell@vanderbilt.edu

Contents

Introduction

The SAS7BDAT file is a binary database storage file. At the time of this writing, no description of the SAS7BDAT file format was publicly available. Hence, users who wish to read and manipulate these files were required to obtain a license for the SAS software, or third party software with support for SAS7BDAT files. The purpose of this document is to promote interoperability between SAS and other popular statistical software packages, especially R (http://www.r-project.org/).

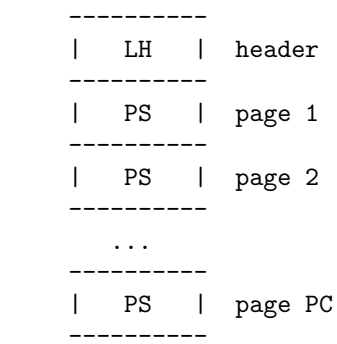
The information below was deduced by examining the contents of many SAS7BDAT databases downloaded freely from internet resources (see `data/sas7bdat.sources.RData`). No guarantee is made regarding its accuracy. No SAS software, nor any other software requiring the purchase of a license was used.

SAS7BDAT files consist of binary encoded data. Data files encoded in this format often have the extension '.sas7bdat'. The name 'SAS7BDAT' is not official, but is used throughout this document to refer to SAS database files formatted according to the descriptions below.

There appear to be significant differences in the SAS7BDAT format across operating systems and computer hardware platforms (32bit vs. 64bit). See the section on platform differences for more details. The format described below applies to the majority of the collection of test files referenced in

`data/sas7bdat.sources.RData` directory (i.e. files associated with 32bit and some 64bit builds of SAS for Microsoft Windows).

The figure below illustrates the overall structure of the SAS7BDAT database. Each file consists of a header (length := LH), followed by PC pages, each of length PS bytes (PC and PS are shorthand for 'page count' and 'page size' respectively, and are used to denote these quantities throughout this document).:



SAS7BDAT Header

The SAS7BDAT file header contains a binary file identifier (*i.e.*, a magic number), the dataset name, timestamp, the number pages (PC), their size (PS) and a variety of other values that pertain to the database as a whole. The purpose of many header fields remain unknown, but are likely to include specifications for data compression and encryption, password protection, and dates/times of creation and/or modification. Most files encountered encode multi-byte values little-endian (least significant byte first). However, at least one file has big-endian values. Hence, it appears that multi-byte values are encoded using endianness of the platform where the file was written.

The *offset table* below describes the SAS7BDAT file header as a sequence of bytes. Information stored in the table is indexed by its byte offset (first column) in the header and its length (second column) in bytes. Byte lengths having the form '%n' should read: 'the number of bytes remaining up to, but not including byte n'. The fourth column gives a shorthand description of the data contained at the corresponding offset. For example, 'uint, page size := PS' indicates that the data stored at the corresponding location is a little-endian unsigned integer representing the page size, which we denote PS. The description *????????????* indicates that the meaning of data stored at the corresponding offset is unknown. The third column represents the author's confidence (low, medium, high) in the corresponding offset, length, and description. Each offset table in this document is formatted in a similar fashion. Variables defined in an offset table are sometimes used in subsequent tables.

Header Offset Table

offset	length	conf.	description
0	32	high	binary, magic number
32	1	medium	binary, Alignment := a1 (x33-4 else-0)
32	3	low	*????????????*
35	1	medium	binary, Alignment := a2 (x33-4 else-0)
36	1	low	*????????????*
37	1	low	int, endianness (x01-little x00-big)
38	1	low	*????????????*
39	1	low	ascii, file format version (1-UNIX or 2-WIN)
40	8	low	*????????????*

... continued on next page

offset	length	conf.	description
48	8	low	*????????????*
56	8	low	repeat of 32:32+8
64	20	low	*????????????*
84	8	high	ascii 'SAS FILE'
92	64	high	ascii, dataset name
156	8+a1	medium	ascii, file type
164+a1	16	high	2x double, timestamp, secs since 1/1/60
180+a1	16	low	*????????????*
196+a2	4	high	int, length of SAS7BDAT header := LH
200+a2	4	high	int, page size := PS
204+a2	4	high	int, page count := PC
208+a1+a2	8	low	*????????????*
216+a1+a2	8	high	ascii, release
224+a1+a2	16	high	ascii, host
240+a1+a2	16	high	ascii, version
256+a1+a2	16	high	ascii, OS maker
272+a1+a2	16	high	ascii, OS name
288+a1+a2	48	low	string with timestamps, license?
336+a1+a2	%LH	medium	filler/zeros

The 8 bytes beginning at offset 32 appear to hold information regarding the offset of the 'release' and 'host' information. The following table describes some of the possible polymorphisms, where the first column contains the hex values for bytes 32-39, the second column shows bytes 216-239 ('.' represents a non-ASCII character or '0'). The byte at offset 39 appears to distinguish the file format type, where '1' indicates that the file was generated on a UNIX-like system, such as Linux or SunOS, and '2' indicates the file was generated on a Microsoft Windows platform. Additional data files are needed to investigate these aspects further.

filename	bytes 32-39	bytes 216-239
`compress_no.sas7bdat`	`22 22 00 32 22 01 02 32`	`9.0101M3NET_ASRV........`
`compress_yes.sas7bdat`	`22 22 00 32 22 01 02 32`	`9.0101M3NET_ASRV........`
`lowbwt_i386.sas7bdat`	`22 22 00 32 22 01 02 32`	`9.0202M0W32_VSPRO.......`
`missing_values.sas7bdat`	`22 22 00 32 22 01 02 32`	`9.0202M0W32_VSPRO.......`
`obs_all_perf_1.sas7bdat`	`22 22 00 32 22 01 02 32`	`9.0101M3XP_PRO..........`
`adsl.sas7bdat`	`22 22 00 33 33 01 02 32`	`....9.0202M3X64_ESRV....`
`eyecarex.sas7bdat`	`22 22 00 33 22 00 02 31`	`....9.0000M0WIN.........`
`lowbwt_x64.sas7bdat`	`22 22 00 33 33 01 02 32`	`....9.0202M2X64_VSPRO...`
`natlterr1994.sas7bdat`	`33 22 00 33 33 00 02 31`	`........9.0101M3SunOS...`
`natlterr2006.sas7bdat`	`33 22 00 33 33 00 02 31`	`........9.0101M3SunOS...`
`txzips.sas7bdat`	`33 22 00 33 33 01 02 31`	`........9.0201M0Linux...`

The binary representation for the hexadecimal values present in the table above are given below.

hexadecimal	decimal	binary
01	001	00000001

... continued on next page

hexadecimal	decimal	binary
02	002	00000010
22	034	00010010
31	049	00011001
32	050	00011010
33	051	00011011

Alignment

In files generated by 64 bit builds of SAS, 'alignment' means that all data field offsets should be a factor of 8 bytes. For files generated by 32 bit builds of SAS, the alignment is 4 bytes. Because SAS7BDAT Packed Binary Data potentially consist of doubles, it seems that all data rows are 64 bit aligned, regardless of whether the file was written with a 32 bit or 64 bit build of SAS. Alignment of data structures according to the platform word length (4 bytes for 32 bit, and 8 bytes for 64 bit architectures) facilitates efficient operations on data stored in memory. It also suggests that parts of SAS7BDAT data file format are platform dependent. One theory is that the SAS implementation utilizes a common C or C++ structure or class to reference data stored in memory. When compiled, these structures are aligned according to the word length of the target platform. Of course, when SAS was originally written, platform differences may not have been forseeable. Hence, these inconsistencies may not have been intentional.

Magic Number

The SAS7BDAT magic number is the following 32 byte (hex) sequence.:

```
00 00 00 00   00 00 00 00
00 00 00 00   c2 ea 81 60
b3 14 11 cf   bd 92 08 00
09 c7 31 8c   18 1f 10 11
```

In all test files except one, the magic number above holds. The one anomalous file has the following magic number:

```
00 00 00 00   00 00 00 00
00 00 00 00   00 00 00 00
00 00 00 00   00 00 00 00
00 00 00 00   18 1f 10 11
```

In addition, the file is associated with the SAS release "3.2TK". Indeed, this file may not have been written by SAS. Otherwise, the anomalous appears to be similar to other test files.

Other Notes

From Clint Cummins (yet to be incorporated properly into this document, or the prototype reader):

> 1A. If byte at offset 35 = 33h, there is a 4 byte filler 00 00 00 00 inserted at offset 164 (between "file type" and "time stamp") 1B. If byte at offset 32 = 33h, there are 4 extra bytes inserted somewhere between "time stamp" and "release". All these files are Linux or SunOS with IOA=8, and none of them have valid PS or PC at the expected positions. So all we really know about them is where the release and host fields are.

SAS7BDAT Pages

Following the SAS7BDAT header are pages of data. Each page can be one of (at least) four types. The first three are those that contain meta-information (e.g. field/column attributes), packed binary data, or a combination of both. These types are denoted 'meta', 'data', and 'mix' respectively. Meta-information is required to correctly interpret the packed binary information. Hence, this information must be parsed

first. In test files (see `data/sources.csv`), 'meta' and 'mix' pages always precede 'data' pages. In some test data files, there is a fourth page type, denoted 'amd' which appears to encode additional meta information. This page usually occurs last, and appears to contain amended meta information.

The page offset table below describes each page type. Byte offsets appended with one of '(meta/mix)', '(mix)', or '(data)' indicate that the corresponding length and description apply only to pages of the listed type. For now, the internal structure of the 'amd' page type is considered identical to the 'meta' page type.

Page Offset Table

offset	length	conf.	description
0	4	low	*????????????* (sometimes repeated)
4	8	low	*????????????* (not critical)
12	4	low	*????????????* row/col related (not critical)
16	2	medium	int, bit field page type
18 (meta/mix)	2	low	*????????????*
20 (meta/mix)	2	medium	int, number of subheader pointers := L
22 (meta/mix)	2	low	*????????????*
24 (meta/mix)	L*12	medium	L subheader pointers, 24+L*12 := M
M (meta)	%PS	medium	subheader data
M+M%8 (mix)	%PS	medium	SAS7BDAT packed binary data
18 (data)	4	medium	int, page row count
24 (data)	%PS	medium	SAS7BDAT packed binary data

Page Type

There are at least four page types 'meta', 'data', 'mix', and 'amd'. These types are encoded in the most significant byte of a two byte bit field at offset 16. If no bit is set, the following page is of type 'meta'. If the first, second, or third bits are set, then the page is of type 'data', 'mix', or 'amd', respectively. Hence, if the two bytes are interpreted as an unsigned integer, then the 'meta', 'data', 'mix', and 'amd' types correspond to 0, 256, 512, and 1024, respectively. In compressed files, other bits (and sometimes multiple bits) have been set (e.g., `1 << 16 | 1 << 13`, which is `-28672` signed, or `36864` unsigned). However, the pattern is unclear.

If a page is of type 'meta', 'mix', or 'amd', data beginning at offset byte 24 are a sequence of L 12-byte subheader pointers, which point to an offset farther down the page. SAS7BDAT Subheaders stored at these offsets hold meta information about the database, including the column names, labels, and types.

If a page is of type 'mix', then **packed binary data begin at the next 8 byte boundary following the last subheader pointer**. In this case, the data begin at offset 24+L*12 + (24+L*12) % 8, where '%' is the modulo operator.

If a page is of type 'data', then packed binary data begin at offset 24.

Subheader Pointers

The subheader pointers encode information about the offset and length of subheaders relative to the beginning of the page where the subheader pointer is located. The purpose of the last four bytes of the subheader pointer are uncertain, but may indicate that additional subheader pointers are to be found on the next page, or that the corresponding subheader is not crucial.

... continued on next page

offset	length	conf.	description

offset	length	conf.	description
0	4	high	int, offset from page start to subheader
4	4	high	int, length of subheader := H
8	1	low	int, optional (0/1)?
9	1	low	int, continue next page (0/1)?
10	2	low	*????????????*

H is sometimes zero, which indicates that no data is referenced by the corresponding subheader pointer. When this occurs, the subheader pointer may be ignored.

SAS7BDAT Subheaders

Subheaders contain meta information regarding the SAS7BDAT database, including row and column counts, column names, labels, and types. Each subheader is associated with a four-byte 'signature' that identifies the subheader type, and hence, how it should be parsed.

Row Size Subheader

The row size subheader holds information about row length (in bytes), their total count, and their count on a page of type 'mix'.

offset	length	conf.	description
0	4	medium	binary, signature F7F7F7F7
4	16	low	*????????????*
20	4	medium	int, row length (in bytes)
24	12	medium	int, row count := r (12 bytes?)
36	4	medium	int, partial column count := CC1
40	4	medium	int, partial column count := CC2
44	8	low	*????????????*
52	4	low	int, page size?
56	4	low	*????????????*
60	4	medium	int, max row count on "mix" page
64	8	medium	sequence of 8 FF, end of header
72	%H	low	filler

The partial column counts CC1 and CC2 usually sum to CC (i.e., CC1+CC2=CC). Usually, CC1 is equal to CC, and CC2 is zero, but there are some exceptions. Their exact purpose is not clear.

Column Size Subheader

The column size subheader holds the column count.

offset	length	conf.	description
0	4	medium	binary, signature F6F6F6F6
4	8	medium	int, column count := CC

Subheader Counts Subheader

This subheader contains information on the first and last appearances of at least 7 common subheader types. Any of these subheaders may appear once or more. Multiple instances of a subheader provide information for an exclusive subset of columns. The order in which data is read from multiple subheaders corresponds to the reading order (left to right) of columns. The subheader counts subheader is always 304 bytes in length. The structure of this subheader was deduced and reported by Clint Cummins.

offset	length	conf.	description
0	4	medium	binary, signature 00FCFFFF
4	4	low	length or offset, usually >= 48d (30h)
8	4	low	usually 4d (4 decimal, 04000000 hex)
12	4	low	usually 7d
76	8	low	usually zeros
84	11*20	medium	11 subheader count vectors, 20 bytes each

Subheader Count Vectors

The subheader count vectors encode information for each of 7 common subheader types, and potentially 11 total subheader types.

offset	length	conf.	description
0	4	medium	binary signature (see list below)
4	4	medium	int, page where this subheader first appears := PAGE1
8	2	medium	int, position of subheader pointer in PAGE1 := LOC1
10	2	low	*??????????*
12	4	medium	int, page where this subheader last appears := PAGEL
16	2	medium	int, position of subheader pointer in PAGEL := LOCL
18	2	low	*??????????*

The LOC1 and LOCL give the positions of the corresponding subheader pointer in PAGE1 and PAGEL, respectively. That is, if there are L subheader pointers on page PAGE1, then the corresponding subheader pointer first occurs at the LOC1'th position in this array, enumerating from 1. If PAGE1=0, the subheader is not present. If PAGE1=PAGEL and LOC1=LOCL, the subheader appears exactly once. If PAGE1!=PAGEL or LOC1!=LOCL, the subheader appears 2 or more times. In all test files, PAGE1 <= PAGEL, and the corresponding subheaders appear only once per page.

The first 7 binary signatures in the Subheader Count Vectors array are always:

hex	decimal	description
FCFFFFFF	-4	Column Attributes
FDFFFFFF	-3	Column Text
FFFFFFFF	-1	Column Names
FEFFFFFF	-2	Column List
FBFFFFFF	-5	unknown signature #1
FAFFFFFF	-6	unknown signature #2
F9FFFFFF	-7	unknown signature #3

The remaining 4 out of 11 signatures are zeros in the observed source files. Presumably, these are for subheaders not yet defined, or not present in the collection of test files.

Column Text Subheader

The column text subheader contains all text associated with columns, including the column name, label, and formatting. However, this subheader is not sufficient to parse these information. Other subheaders (e.g. the column name subheader), which point to specific elements relative to this subheader are also needed.

offset	length	conf.	description
0	4	medium	binary, signature FDFFFFFF
4	12	medium	int, length of remaining subheader
16	60	medium	ascii, proc name that generated data?
76	%H	high	ascii, combined column names, labels, formats

This subheader sometimes appears more than once; each is a separate array. If so, the "column name index" field in column name pointers selects a particular text array - 0 for the first array, 1 for the second, etc. Similarly, "column format index" and "column label index" fields also select a text array. For compressed files, the type of compression is indicated within the field at offset 16 of the first column text subheader. In particular, if the first eight bytes are ascii "SASYZCRL", then the file was generated with the option COMPRESS=YES, and data are apparently compressed using a simple run-length encoding (RLE) algorithm.

Column Name Subheader

Column name subheaders contain a sequence of column name pointers to the offset of each column name **relative to a 'column text subheader'**. There may be multiple column name subheaders, indexing into multiple column text subheaders.

offset	length	conf.	description
0	4	medium	binary, signature FFFFFFFF
4	8	medium	int, length of remaining subheader
12	8*CMAX	medium	column name pointers (see below), CMAX=(H-12-8)/8
12+8*CMAX	8	low	filler

Each column name subheader hold CMAX column name pointers. When there are multiple column name subheaders, CMAX will be less than CC.

Column Name Pointers

offset	length	conf.	description
0	2	medium	int, column name index to select Column Text Subheader
2	2	medium	int, column name offset w.r.t. FDFFFFFF
4	2	medium	int, column name length
6	2	low	binary, zeros

Column Attributes Subheader

The column attribute subheader holds information regarding the column offsets within a row, the column widths, and the column types (either numeric or character). The column attribute subheader sometimes occurs more than once (in test data). In these cases, column attributes are applied in the order they are

parsed.

offset	length	conf.	description
0	4	medium	binary, signature FCFFFFFF
4	8	medium	int, length of remaining subheader
12	12*CMAX	medium	column attributes (see below), CMAX=(H-12-8)/12
12+12*CMAX	8	medium	filler

Column Attributes

offset	length	conf.	description
0	4	medium	int, column offset in w.r.t. row
4	4	medium	int, column width
8	2	low	name length flag
10	1	medium	int, column type (01-num, 02-chr)
11	1	low	*????????????*

Observed values of name length flag in the source files:

name length flag	description
4	name length <= 8
1024	usually means name length <= 8 , but sometimes the length is 9-12
2048	name length > 8
2560	name length > 8

Column Format and Label Subheader

The column format and label subheader contains pointers to a column format and label **relative to the 'column text subheader'**. Since the column label subheader only contains information regarding a single column, there are typically as many of these subheaders as columns. The structure of column format pointers was contributed by Clint Cummins.

offset	length	conf.	description
0	4	medium	binary, signature FEFBFFFF
4	30	low	*????????????*
34	2	medium	int, column format index to select Column Text Subheader
36	2	medium	int, column format offset wrt FDFFFFFF
38	2	medium	int, column format length
40	2	medium	int, column label index to select Column Text Subheader
42	2	medium	int, column label offset wrt FDFFFFFF
44	2	medium	int, column label length
46	6	low	*????????????*

Column List Subheader

The purpose of this subheader is not clear. But the structure is partly identified. Information related to this subheader was contributed by Clint Cummins.

offset	length	conf.	description
0	4	medium	binary, signature FEFFFFFF
4	2	medium	int, length of remaining subheader
6	6	low	*????????????*
12	2	medium	int, length of remaining subheader
14	2	low	*????????????*
16	2	low	int, usually equals CC
18	2	medium	int, length of column list := CL
20	2	low	int, usually 1
22	2	low	int, usually equals CC
24	6	low	*????????????*
30	2*CL	medium	column list values (see below)
30+2*CL	8	low	usually zeros

Column List Values

These values are 2 byte, little-endian signed integers. Each value is between -CC and CC. The significance of signedness and ordering is unknown. The values do not correspond to a sorting order of columns.

SAS7BDAT Packed Binary Data

SAS7BDAT packed binary data are stored by rows, where the size of a row (in bytes) is defined by the row size subheader. When multiple rows occur on a single page, they are immediately adjacent. When a database contains many rows, it is typical that the collection of rows (i.e. their data) is evenly distributed to a number of 'data' pages. However, in test files, no single row's data is broken across two or more pages. A single data row is parsed by interpreting the binary data according to the collection of column attributes contained in the column attributes subheader. Binary data can be interpreted in two ways, as ASCII characters, or as floating point numbers. The column width attribute specifies the number of bytes associated with a column. For character data, this interpretation is straight-forward. For numeric data, interpretation of the column width is more complex.

The common binary representation of floating point numbers has three parts; the sign (`s`), exponent (`e`), and mantissa (`m`). The corresponding floating point number is `s * m * b ^ e`, where `b` is the base (2 for binary, 10 for decimal). Under the IEEE 754 floating point standard, the sign, exponent, and mantissa are encoded by 1, 11, and 52 bits respectively, totaling 8 bytes. In SAS7BDAT file, numeric quantities can be 3, 4, 5, 6, 7, or 8 bytes in length. For numeric quantities of less than 8 bytes, the remaining number of bytes are truncated from the least significant part of the mantissa. Hence, the minimum and maximum numeric values are identical for all byte lengths, but shorter numeric values have reduced precision.

Reduction in precision is characterized by the largest integer such that itself and all smaller integers have an exact representation, denoted `M`. At best, all integers greater than `M` are approximated to the nearest multiple of `b`. The table of numeric binary formats below lists `M` values and describes how bits are distributed among the six possible column widths in SAS7BDAT files, and lists.

Numeric Binary Formats

size	bytes	sign	exponent	mantissa	M
24bit	3	1	11	12	8192
32bit	4	1	11	20	2097152
40bit	5	1	11	28	536870912
48bit	6	1	11	36	137438953472
56bit	7	1	11	44	35184372088832
64bit	8	1	11	52	9007199254740990

Dates, Currency, and Formatting

Column formatting infomation is encoded within the Column Text Subheader and Column Format and Label Subheader. Columns with formatting information have special meaning and interpretation. For example, numeric values may represent dates, encoded as the number of seconds since midnight, January 1, 1960. The format string for fields encoded this way is "DATETIME". Using R, these values may be converted using the as.POSIXct or as.POSIXlt functions with argument `origin="1960-01-01"`. The most common date format strings correspond to numeric fields, and are interpreted as follows:

Format	Interpretation	R Function
DATE	Number of days since January 1, 1960	chron::chron
TIME	Number of seconds since midnight	as.POSIXct
DATETIME	Number of seconds since January 1, 1960	as.POSIXct

There are many additional format strings for numeric and character fields.

Platform Differences

The test files referenced in `data/sas7bdat.sources.RData` were examined over a period of time. Files with non-Microsoft Windows markings were only observed late into the writing of this document. Consequently (but not intentionally), the SAS7BDAT description above is specific to SAS datasets generated on the most commonly observed platform: Microsoft Windows. SAS7BDAT files generated on other platforms have differenct structure.

In particular, the files `natlerr1944.sas7bdat`, `natlerr2006.sas7bdat` appear to be generated on the 'SunOS' platform. The header in these files appear to be 8196 bytes, rather than the 1024 seen on Microsoft Windows platforms.

The files `cfrance2.sas7bdat`, `cfrance.sas7bdat`, `coutline.sas7bdat`, `gfrance2.sas7bdat`, `gfrance.sas7bdat`, `goutline.sas7bdat`, `xfrance2.sas7bdat`, `xfrance.sas7bdat`, `xoutline.sas7bdat` appear to be generated on a 'Linux' system.

Text may appear in non-ASCII compatible, partially ASCII compatible, or multi-byte encodings. In particular, Kasper Sorenson discovered some text that appears to be encoded using the Windows-1252 'code page'.

Compression Data

The table below presents the results of compression tests on a collection of 142 SAS7BDAT data files (sources in `data/`). The 'type' field represents the type of compression, 'ctime' is the compression time (in seconds), 'dtime' is the decompression time, and the 'compression ratio' field holds the cumulative disk usage (in megabytes) before and after compression. Although the `xz` algorithm requires significantly more time to compress these data, the decompression time is on par with gzip.

type	ctime	dtime	compression ratio
gzip -9	76.7s	2.6s	541M / 30.3M = 17.9
bzip2 -9	92.7s	11.2s	541M / 19.0M = 28.5
xz -9	434.2s	2.7s	541M / 12.8M = 42.3

Software Prototype

The prototype program for reading SAS7BDAT formatted files is implemented entirely in R (see file `src/sas7bdat.R`). Files not recognized as having been generated under a Microsoft Windows platform are rejected (for now). Implementation of the `read.sas7bdat` function should be considered a 'reference implementation', and not one designed with performance in mind.

There are certain advantages and disadvantages to developing a prototype of this nature in R. Advantages:

1. R is an interpreted language with built-in debugger. Hence, experimental routines may be implemented and debugged quickly and interactively, without the need of external compiler or debugger tools (e.g. gcc, gdb).
2. R programs are portable across a variety of computing platforms. This is especially important in the present context, because manipulating files stored on disk is a platform-specific task. Platform-specific operations are abstracted from the R user.

Disadvantages:

1. Manipulating binary (raw) data in R is a relatively new capability. The best tools and practices for binary data operations are not as developed as those for other data types.
2. Interpreted code is often much less efficient than compiled code. This is not major disadvantage for prototype implementations because human code development is far less efficient than the R interpreter. Gains made in efficient code development using an interpreted language far outweigh benefit of compiled languages.

ToDo

- what are CC1 and CC2 for?
- experiment further with 'amendment page' concept
- consider header bytes -by- SAS_host
- check that only one page of type "mix" is observed. If so insert "In all test cases (`data/sources.csv`), there are exactly zero or one pages of type 'mix'." under the Page Offset Table header.
- identify all missing value representations: missing numeric values appear to be represented as '0000000000D1FFFF' (nan) for numeric 'double' quantities.
- identify purpose of unknown header quantities
- determine other bytes in subheader with signature FEFBFFFF
- identify how non-ASCII encoding is specified
- identify SAS7BDAT compression and encryption methods (this is not the same as 'cracking', or breaking encryption): data files may be compressed using the RLE (CHAR) and RDC (BINARY) algorithms.
- implement options to read just header (and subheader) information without data, and an option to read just some data fields, and not all fields.