SAS7BDAT Database Binary Format

Matthew S. Shotwell

Contents

Introduction

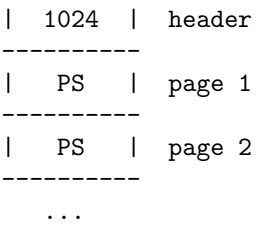
The SAS7BDAT file is a binary database storage file. At the time of this writing, no description of the SAS7BDAT file format is publicly available. Hence, users who wish to read and manipulate these files must obtain a license for the SAS software, or third party software with support for SAS7BDAT files. 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/sources.csv`). 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 (see platform differences). The format described below applies to the majority of the collection of test files referenced in `data/sources.csv` directory (i.e. files associated with Microsoft Windows).

The figure below illustrates the overall structure of the SAS7BDAT database. Each file consists of a 1024 byte header, 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).:



```
----------
|   PS   |  page PC
----------
```

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. All files encountered encode multi-byte values little-endian (least significant byte first). However, it is typical to specify endianness of multi-byte values in a file header.

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 until byte n'. The fourth column gives a short description of the data contained at this address. For example, 'LE 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 location 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	3	low	*????????????*
35	3	low	bitmasks (SAS_host, SAS_release)
38	1	low	*????????????*
39	1	low	ascii, file format version (1-UNIX or 2-WIN)
40	52	low	*????????????*
92	64	high	ascii, dataset name
156	8	medium	ascii, file type
164	16	high	2x LE double, timestamp, secs since 1/1/60
180	16	low	*????????????*
196	20	low	*????????????*
200	4	high	LE uint, page size := PS
204	4	high	LE uint, page count := PC
208	8	low	*????????????*
216	8	high	ascii, release
224	8	high	ascii, host
232	56	low	*????????????*
288	48	low	string with timestamps, license?
336	%1024	medium	filler/zeros

The bitmasks at offsets 35, 36, and 37 appear to hold information regarding the offset of the 'release' and 'host' information. The following table describes the possible polymorphisms, where the first column contains the hex values for bytes 35-37, the second column shows bytes 216-239 ('.' represents a non-

ASCII character or '0', 'a' represents an ASCII character), and the third column gives the type of platform data observed there ('WIN_*' represents various Microsoft Windows types, such as 'WIN_NT' and 'WIN_PRO'). Additional data files are needed to investigate this aspect further.

bytes 35-37	host + release data	platform
32 22 01	aaaaaaaaaaaaaaaa.....	WIN_* and Linux
33 22 00	....aaaaaaaaaaaa.....	WIN
33 33 00	.....aaaaaaaaaaaaaaaa	SunOS

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.

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
```

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`), pages containing meta-information always precede pages consisting entirely of packed binary data. In some test data files (from a single source), there is a fourth page type (04) which appears to encode additional meta information. This page usually occurs last, and appears to contain amended meta information. It's purpose is unclear.

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.

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	1	low	*???????????*
17	1	low	LE uint, page type meta/data/mix/? (0/1/2/4)
18 (meta/mix)	2	low	*???????????*
20 (meta/mix)	4	medium	LE uint, number of subheader pointers := L
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

... continued on next page

offset	length	conf.	description
18 (data)	4	medium	LE uint, page row count
24 (data)	%PS	medium	SAS7BDAT packed binary data

If a page is of type 'meta' or 'mix', 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.

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

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	LE uint, row length (in bytes)
24	12	medium	LE uint, row count := r (12 bytes?)
36	12	medium	LE uint, column count (12 bytes?)
48	4	low	*????????????*
52	4	low	LE uint, page size?
56	4	low	*????????????*
60	4	medium	LE uint, max row count on "mix" page
64	8	medium	sequence of 8 FF, end of header

... continued on next page

offset	length	conf.	description
72	%H	low	filler

Column Size Subheader

The column size subheader holds the column count.

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

Signature 00FCFFFF Subheader

The purpose of the subheader with signature 00FCFFFF is unknown. This subheader might contain pointers to column formatting information relative to the column text subheader.

offset	length	conf.	description
0	4	medium	binary, signature 00FCFFFF
4	%H	low	*????????????*

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	LE uint, length of remaining subheader
16	60	medium	ascii, proc name that generated data?
76	%H	high	ascii, combined column names, labels, formats

Column Name Subheader

The column name subheader contains a sequence of column name pointers to the offset of each column name **relative to the 'column text subheader'**_.

offset	length	conf.	description
0	4	medium	binary, signature FFFFFFFF
4	8	medium	LE uint, length of remaining subheader
12	8*CC	medium	column name pointers (see below)
12+8*CC	8	medium	filler

Column Name Pointers

offset	length	conf.	description
0	1	low	LE uint, offset relative to page 04 subheader
0	1	low	*????????????*
2	2	medium	LE uint, column name offset w.r.t. FDFFFFFF
4	2	medium	LE uint, column name length
6	2	low	binary, zeros

If the first byte in the column name pointer is 01 (it is usually 00), this indicates that the column name offset is relative to an 'amendment subheader' (i.e. a subheader with the same signature, but found on an amendment page (page type 04).

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	LE uint, length of remaining subheader
12	12*CC	medium	column attributes (see below)
12+12*CC	8	medium	filler

Column Attributes

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

Column Label Subheader

The column label subheader contains a column label pointer to the offset of a column label **relative to the 'column text subheader'**_. Since the column label subheader only contains information regarding a single column, there are typically as many column label subheaders as columns.

offset	length	conf.	description
0	4	medium	binary, signature FEFBFFFF
4	38	low	*????????????*
42	2	medium	LE uint, column label offset wrt FDFFFFFF
44	2	medium	LE uint, column label length
46	6	low	*????????????*

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 ^ e`. Under the IEEE 754 floating point standard, the sign requires 1 bit, the exponent requires 11, and the mantissa requires 52 bits, for a total of 8 bytes. In SAS7BDAT file, numeric quantities can be 3, 4, 5, 6, 7, or 8 bytes in length. For numeric quantities using less than 8 bytes, some number of bytes are absent from the most significant part of the mantissa. The smaller width mantissa means that the range of possible values is restricted. The table of numeric binary formats below describes how bits are distributed among the six possible column widths in SAS7BDAT files.

Numeric Binary Formats

size	**24bit**	**32bit**	**40bit**	**48bit**	**56bit**	**64bit**[1]
bytes	3	4	5	6	7	8
sign	1	1	1	1	1	1
exponent	11	11	11	11	11	11
mantissa	12	20	28	36	44	52

Platform Differences

The test files referenced in `data/sources.csv` 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. The format of SAS7BDAT files generated on other platforms are formatted differently.

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.

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.

[1] Only 64bit is IEEE 754 compliant!

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

- 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 subheader 00FCFFFF
- identify purpose of unknown header quantities
- determine other bytes in subheader with signature FEFBFFFF
- can SAS7BDAT files use non-ASCII encoding?
- 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.