

[buchanan1.net]

The GDSII Stream Format

Search buchanan1.net

See also: [stream_utils](#)

[Shop at Amazon.com!]

Jim Buchanan
6/11/96

- * Home
- * Bio
- * Links
- * White/Yellow Pages
- * Computer
- * Motorcycles
- * General
- * Electronics
- * Fun
- * Woodworking
- * in.* FAQ
- * Email Me

This file is for use by people having experience with GDSII format stream files and the CAD systems that read/write them. It won't make a lot of sense without that background.

Since knowing the library structure without knowing about records and data types would be of marginal use, and knowing about records and data types without knowing about the library structure would be worse, you might have to scan through this a few times before it makes sense.

Beyond that, let me say that the stream format is quite simple. I suspect that the people at Calma put a lot of thought into creating a file that would be as easy to read in and parse as possible.

I suspect that they did this due to the modest computers that they had to work with. Their results were impressive here and with the GDSII system as a whole. A bit dated now, but in its day...

Search:
Keywords:
[In Association with Amazon.com]

Speaking of dated, Stream Format allows records to be written out to multiple reels of tape. Handy on those old 9 track drives. When a file was written to a tape, it was written in 2048 byte physical blocks. The file was padded with NULL characters so that it was always a multiple of 2048 bytes.

I've noticed that some (OK, many) stream files that were originally written to disk using more modern software also pad the file to a multiple of 2048 bytes using NULL characters.

A stream file consists of records.

These records are built up of 16 bit words. This means that all stream files should have an even number of bytes.

The first two words, or four bytes, are called the "Record Header"

A record can be as small as 4 bytes long. The GDSII Stream format manual says that a record may be infinitely long, but frankly, I don't see how it can get over 65535 bytes long, since the first two bytes of the record header are an unsigned integer that defines the length of the record. The leftmost bit of the first byte is valued at 32768, the rightmost bit of the second byte is valued at 1.

The third byte is the record type. This will tell what part of the library this record describes. There are values for such things as the beginning of a structure, the beginning of a boundary, the end of a structure, and so on. There is a section below with hexadecimal values of the various record types and a brief description of the types.

The fourth, and last, byte in the record header is the data type. This, along with the record length, tells the parser what to expect in the rest of the record. There may be more than one piece of data, but the rest of the record will be of this type. You can tell how many pieces of data are in the record by knowing the number of bytes in the record and the size of the data type.

Actually, the data type seems redundant, since each record type has only one valid data type. Perhaps the Calma people were thinking of future needs? They sure did that with the layer numbers and data types. Calma allowed only 64 layers and data types, but the Stream Format has room for 65535 of each.

There are seven data types listed in the GDSII Stream Format Manual v6.0, one is listed as not being used at the time. I doubt anyone has had the chutzpah to start using it since.

The first is "No data present". The code is 0x00. This means that the entire record is 4 bytes long. An example of an element with no data would be ENDLIB which marks the end of a library.

The second is called a "Bit array". The code is 0x01. It's simply two bytes. The meaning of each bit depends on the record type that the bit array is found in.

The third data type is a "Two-Byte Signed Integer". The code is 0x02. It is an integer between -32768 and 32767. It is stored in twos complement format, with the most significant byte first.

Some examples from the book:

```
0x0000 = 1
0x0020 = 2
0x0089 = 137
0xffff = -1
0xfffe = -2
0xff77 = -137
```

The fourth data type is a "Four-Byte Signed Integer". The code is 0x03. Same basic thing as a two byte integer, but with four bytes.

The fifth data type is the "Four-Byte Real". The code is 0x04. This is the one that seems to have never been used, so I'll describe the eight byte real in a bit more detail.

Basically though, the first bit is the sign (1 = negative), the next 7 bits are the exponent, you have to subtract 64 from this number to get the real value. The next three bytes are the mantissa, divide by 2²⁴ to get the denominator.

$$\text{value} = (\text{mantissa}/(2^{24})) * (16^{(\text{exponent}-64)})$$

In the above, we use the actual values of the fields in the stream file for the mantissa and exponent.

The sixth data type is the "Eight Byte Real". The code is 0x05. This one gets a little more use.

The first (most significant) bit of the first byte is the sign, one means negative, 0 means positive.

The 7 least significant bits of the first byte are the exponent in "excess 64" notation. You must subtract 64 to get the true value. I'll show the subtraction in the formula below.

The remaining 7 bytes are the mantissa, with a binary point to the left of the most significant figure. The formula below uses the unsigned integer value of these 7 bytes as the numerator of a fraction.

$$\text{value} = (\text{mantissa}/(2^{56})) * (16^{(\text{exponent}-64)})$$

The seventh and final data type is the "ASCII String". The code is 0x06. The length of this string is always equal to the length of the record minus the four bytes used for the record header. If this number is not even, a NULL character (0x00) is added to the end. This is another artifact of the 16 bit words that the stream file format assumes.

This is the format of a stream file. The records shown within square brackets '['] are optional. The or bar '|' indicates one or the other. The structure '{}+' is used to indicate one or more instances. The angle brackets '<>' indicate that further definition is below. Sort of an "include" element.

The actual stream file:

```
HEADER
BGNLIB
[LIBDIRSIZE]
[SRFNAME]
[LIBSECUR]
LIBNAME
[REFLIBS]
[ FONTS]
[ATTRTABLE]
[GENERATIONS]
[FORMAT | FORMAT {MASK}+ ENDMASKS]
UNITS
[ {BGNSTR STRNAME [STRCLASS] [ {<element>}+ ] ENDSTR}+ ]
ENDLIB
```

An element portion of a stream file:

```
<boundary> | <path> | <sref> | <aref> | <text> | <node> | <box>
[ {PROPATTR PROPVALUE}+ ]
ENDEL
```

Boundary portion of an element:

```
BOUNDARY
[ELFLAGS]
[ PLEX]
LAYER
DATATYPE
XY
```

Path portion of an element:

PATH
[ELFLAGS]
[PLEX]
LAYER
DATATYPE
[PATHTYPE]
[WIDTH]
[BGNEXTN]
[ENDEXTN]
XY

SREF portion of an element:

SREF
[ELFLAGS]
[PLEX]
SNAME
[STRANS [MAG] [ANGLE]]
XY

AREF portion of an element:

AREF
[ELFLAGS]
[PLEX]
SNAME
[STRANS [MAG] [ANGLE]]
COLROW
XY

Text portion of an element:

TEXT
[ELFLAGS]
[PLEX]
LAYER
TEXTTYPE
[PRESENTATION]
[PATHTYPE]
[WIDTH]
[STRANS [MAG] [ANGLE]]
XY
STRING

Node portion of an element:

NODE
[ELFLAGS]
[PLEX]
LAYER
NODETYPE
XY

Box portion of an element:

BOX
[ELFLAGS]
[PLEX]
LAYER

BOXTYPE
XY

A stream file may be broken up over multiple reels of tape. I haven't seen this since the days of 9 track tapes on reels, but just in case, here we go...

Tape 1:

HEADER
several complete stream records
TAPENUM
TAPECODE
LIBNAME

Intermediate tape(s):

TAPENUM
TAPECODE
LIBNAME
more complete stream records
TAPENUM

last tape:

TAPENUM
TAPECODE
LIBNAME
more complete stream records
ENDLIB

A concatenation of all of the tapes, without the tape id stuff (and I presume w/o the extra LIBNAMEs) should be a valid stream file as described above.

OK, here's the part you've been waiting for, what the records mean...

Record type	Data type	
0x00 HEADER	0x02 INTEGER_2	Start of stream, contains version number of stream file. < v3.0 0x0000 0 v3.0 0x0003 3 v4.0 0x0004 4 v5.0 0x0005 5 v6.0 0x0258 600
0x01 BGNLIB	0x02 INTEGER_2	Beginning of library, plus mod and access dates. Modification: year, month, day, hour, minute, second Last access: year, month, day, hour, minute, second
0x02 LIBNAME	0x06 STRING	The name of the library, supposedly following

Calma DOS conventions. Using later tools, such as ISS LTL-100, it seems more flexible than that, but it won't allow any old thing you want. If memory serves, Calma DOS allowed 6 characters in a file name, with a 2 character extension.

0x03 UNITS	0x05 REAL_8	Size of db unit in user units, size of db unit in meters. To calculate the size of a user unit in meters, divide the second number by the first.
0x04 ENDLIB	0x00 NO_DATA	End of the library.
0x05 BGNSTR	0x02 INTEGER_2	Begin structure, plus create and mod dates in the same format as the BGNLIB record.
0x06 STRNAME	0x06 STRING	Name of a structure. Up to 32 characters in GDSII, A-Z, a-z, 0-9, _, ?, and \$ are all legal characters.
0x07 ENDSTR	0x00 NO_DATA	End of a structure.
0x08 BOUNDARY	0x00 NO_DATA	The beginning of a BOUNDARY element.
0x09 PATH	0x00 NO_DATA	The beginning of a PATH element.
0x0a SREF	0x00 NO_DATA	The beginning of an SREF element.
0x0b AREF	0x00 NO_DATA	The beginning of an AREF element.
0x0c TEXT	0x00 NO_DATA	The beginning of a TEXT element.
0x0d LAYER	0x02 INTEGER_2	Layer specification. On GDSII this could be 0 to 63, LTL allows 0 to 255. Of course a 3 byte integer allows up to 65535...
0x0e DATATYPE	0x02 INTEGER_2	Datatype specification. On GDSII this could be 0 to 63, LTL allows 0 to 255. Of course a 3 byte integer allows up to 65535...
0x0f WIDTH	0x03 INTEGER_4	Width specification, negative means absolute In data base units.
0x10 XY	0x03 INTEGER_4	An array of XY coordinates. An array of coordinates in data base units. Path: 2 to 200 pairs in GDSII Boundary: 4 to 200 pairs in GDSII Text: Exactly 1 pair SREF: Exactly 1 pair AREF: Exactly 3 pairs 1: Array reference point 2: column_space*columns+reference_x 3: row_space*rows+reference_y Node: 1 to 50 pairs in GDSII Box: Exactly 5 pairs
0x11 ENDEL	0x00 NO_DATA	The end of an element.
0x12 SNAME	0x06 STRING	The name of a referenced structure.

0x13	COLROW	0x02	INTEGER_2	Columns and rows for an AREF. Two 2 byte integers. The first is the number of columns. The second is the number of rows. In an AREF of course. Neither may exceed 32767
0x14	TEXTNODE	0x00	NO_DATA	"Not currently used" per GDSII Stream Format Manual, v6.0. Would be the beginning of a TEXTNODE element if it were.
0x15	NODE	0x00	NO_DATA	The beginning of a NODE element.
0x16	TEXTTYPE	0x02	INTEGER_2	Texttype specification. On GDSII this could be 0 to 63, LTL allows 0 to 255. Of course a 3 byte integer allows up to 65535...
0x17	PRESENTATION	0x01	BIT_ARRAY	Text origin and font specification. bits 15 to 0, 1 to r bits 0 and 1: 00 left, 01 center, 10 right bits 2 and 3: 00 top 01, middle, 10 bottom bits 4 and 5: 00 font 0, 01 font 1, 10 font 2, 11 font 3,
0x18	SPACING		UNKNOWN	"Discontinued" per GDSII Stream Format Manual, v6.0.
0x19	STRING	0x06	STRING	Character string. Up to 512 char in GDSII
0x1a	STRANS	0x01	BIT_ARRAY	Bits 15 to 0, 1 to r 15=refl, 2=absmag, 1=absangle, others reserved for future use.
0x1b	MAG	0x05	REAL_8	Magnification, 1 is the default if omitted.
0x1c	ANGLE	0x05	REAL_8	Angular rotation factor in ccw direction. If omitted, the default is 0.
0x1d	UINTEGER		UNKNOWN	User integer, used only in V2.0, when instreamed, should be converted to property attribute 126.
0x1e	USTRING		UNKNOWN	User string, used only in V2.0, when instreamed, should be converted to property attribute 127.
0x1f	REFLIBS	0x06	STRING	Names of the reference libraries. Starts with name of the first library and is followed by the second. There are 44 bytes in each, NULLS are used for padding, including filling in an entire unused field.
0x20	FONTS	0x06	STRING	Names of the textfont definition files. 4 44 byte fields, padded with NULLS if a field is unused or less than 44 bytes.
0x21	PATHTYPE	0x02	INTEGER_2	Type of path ends. 0: Square ended paths 1: Round ended 2: Square ended, extended 1/2 width 4: Variable length extensions, CustomPlus

			The default is 0
0x22	GENERATIONS	0x02 INTEGER_2	Number of deleted or backed up structures to retain. Seems a bit odd in an archive... From 2-99, default is 3.
0x23	ATTRTABLE	0x06 STRING	Name of the attribute definition file. Max size 44 bytes.
0x24	STYPTABLE	0x06 STRING	"Unreleased feature" per GDSII Stream Format Manual, v6.0.
0x25	STRTYPE	0x02 INTEGER_2	"Unreleased feature" per GDSII Stream Format Manual, v6.0
0x26	ELFLAGS	0x01 BIT_ARRAY	Flags for template and exterior data. bits 15 to 0, 1 to r 0=template, 1=external data, others unused
0x27	ELKEY	0x03 INTEGER_4	"Unreleased feature" per GDSII Stream Format Manual, v6.0.
0x28	LINKTYPE	UNKNOWN	"Unreleased feature" per GDSII Stream Format Manual, v6.0.
0x29	LINKKEYS	UNKNOWN	"Unreleased feature" per GDSII Stream Format Manual, v6.0.
0x2a	NODETYPE	0x02 INTEGER_2	Nodetype specification. On GDSII this could be 0 to 63, LTL allows 0 to 255. Of course a 3 byte integer allows up to 65535...
0x2b	PROPATTR	0x02 INTEGER_2	Property number.
0x2c	PROPVALUE	0x06 STRING	Property value. On GDSII, 128 characters max, unless an SREF, AREF, or NODE, which may have 512 characters.
0x2d	BOX	0x00 NO_DATA	The beginning of a BOX element.
0x2e	BOXTYPE	0x02 INTEGER_2	Boxtype specification. On GDSII this could be 0 to 63, LTL allows 0 to 255. Of course a 3 byte integer allows up to 65535...
0x2f	PLEX	0x03 INTEGER_4	Plex number and plexhead flag. The least significant bit of the most significant byte is the plexhead flag. Because of this, you can "only" have 2^{24} plex groups. Or is that $2^{24}-1$? I'm not sure if 0 is a valid plex group in a stream file.
0x30	BGNEXTN	0x03 INTEGER_4	Path extension beginning for pathtype 4 in CustomPlus. In database units, may be negative.
0x31	ENDTEXTN	0x03 INTEGER_4	Path extension end for pathtype 4 in CustomPlus. In database units, may be negative.
0x32	TAPENUM	0x02 INTEGER_2	Tape number for multi-reel stream file.

0x33 TAPECODE	0x02 INTEGER_2	Tape code to verify that the reel is from the proper set. 12 bytes that are supposed to form a unique tape code.
0x34 STRCLASS	0x01 BIT_ARRAY	Calma use only. In stream files created by non-Calma programs, this should be missing or all field should be 0.
0x35 RESERVED	0x03 INTEGER_4	Used to be NUMTYPES per GDSII Stream Format Manual, v6.0.
0x36 FORMAT	0x02 INTEGER_2	Archive or Filtered flag. 0: Archive 1: filtered
0x37 MASK	0x06 STRING	Only in filtered streams. Layers and datatypes used for mask in a filtered stream file. A string giving ranges of layers and datatypes separated by a semicolon. There may be more than one mask in a stream file.
0x38 ENDMASKS	0x00 NO_DATA	The end of mask descriptions.
0x39 LIBDIRSIZE	0x02 INTEGER_2	Number of pages in library director, a GDSII thing, it seems to have only been used when Calma INFORM was creating a new library.
0x3a SRFNAME	0x06 STRING	Sticks rule file name.
0x3b LIBSECUR	0x02 INTEGER_2	Access control list stuff for CalmaDOS, ancient. INFORM used this when creating a new library. Had 1 to 32 entries with group numbers, user numbers and access rights.
