File Formats

Sony Computer Entertainment
919 E. Hillsdale Blvd., 2nd Flr
Foster City, CA 94404

Publication date: August 1996

Table of Contents

About this Manual

About this Release

This is the first (beta) release of the File Formats manual. It is a compilation of file format specifications extracted from several other manuals in the Developer Reference Series (see Related Documentation). The purpose of this new book is to provide a single authoritative reference to PlayStation file formats.

Related Documentation

The following volumes in the Developer Reference Series contain file format informatio:

Run-time Library 3.0/Overview
Playstation Operating Systems
Sprite Editor
3D Graphics Tools
Sound Artist Tool

Typographic Conventions

Certain Typographic Conventions are used through out this manual to clarify the meaning of the text. The following details the specific conventions used to represent literals, arguments, keywords, etc.

The following conventions apply to all narrative text outside of the structure and function descriptions.

Convention	*Meaning*
|	A revision bar. Indicates that information to the left or right of the bar has been changed or added since the last release.
`courier`	Indicates literal program code.
Bold	Indicates a document, chapter or section title.

The following conventions apply within structure and function descriptions only:

Convention	*Meaning*
Medium Bold	Denotes structure or function types and names.
Italic	Denotes function arguments and structure members.
{ }	Denotes the start and end of the member list in a structure declaration.

Order Information

Additional copies of this documentation can be ordered by contacting:

Sony Computer Entertainment America
919 East Hillsdale Blvd., 2nd floor
Foster City, CA 94404
Tel (415)-655-8000

Chapter 1:
Streaming Audio and Video Data

STR: Streaming (Movie) Data

"Streaming" refers to a processing format for successive reading and processing of data from a CD-ROM. STR format is a CD-ROM data format defined to enable streaming for the PlayStation.

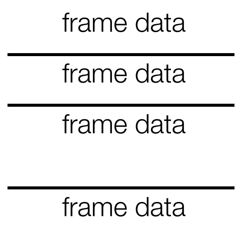
Although streaming is generally used to successively read and play back animation or audio data, it is not limited to such applications. Streaming can also be used for various other kinds of time-series data processing that involves continuous changes.

Continuously reading data from a CD-ROM is called "streaming". The streaming library is used separately from other CD-ROM functions. The bitstream used for animation and movie playback is obtained via this streaming mechanism. Image size and other such supplemental information are not included in the bitstream. For this reason, the supplemental data format (STR format) separately defines information required for animation playback in the header.

Streaming data

As shown in Figure 1-1, streaming data in an STR file is represented as a continuous array of frame data elements. Frame data is used to represent a location for streaming. The frame contents differ depending on what kind of data is used for streaming. For example, in animation, each element of frame data may contain one of the still images that makes up the animation sequence. If the application is animation of 3D modeling data, each element of frame data will contain one unit of the modeling data that makes up the animation sequence.

Figure 1–1: Streaming data

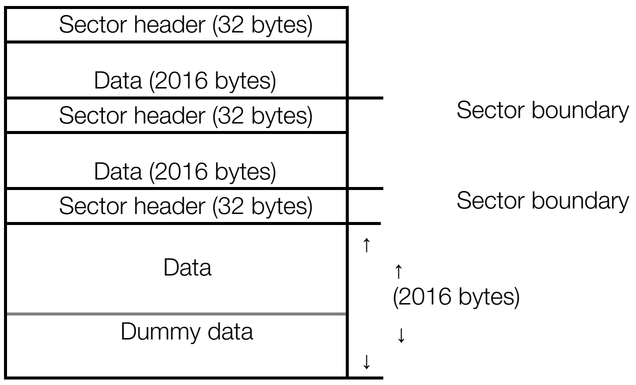


Frame data

Frame data in an STR file is divided into 2048-byte sectors, corresponding to CD-ROM data sectors. Each sector starts with a 32-byte header with information about the sector data, followed by 2016 bytes of data. Each frame must begin on a sector boundary. When necessary, the last sector of data for a frame should be padded with filler to reach the 2048-byte boundary, as shown in figure 1-2.

The type of data within each frame is not specified by the STR format, so it can be freely defined in various ways as needed. The data length and type is specified independently for each frame, so it is possible to mix various types and sizes of frame data within an STR file as needed. Continuously reading data from a CD-ROM is called "streaming". The streaming library is used separately from other CD-ROM functions.

Figure 1–2: Frame data elements



Sector headers

A sector header, located at the start of each sector, is divided into fields as shown in Figure 1-3.

ID

The ID is a two-byte field containing the STR format identifier, version, and other information.

Type

Type indicates the frame data's data type: if the MSB is a 1, the data type is a system-defined format; if the MSB is 0 it is a user-defined format.

After setting the MSB to 0, the user can set the other 15 bits as desired. This enables other frame formats to be incorporated into the STR format.

SecCount/NSectors/FrameCount

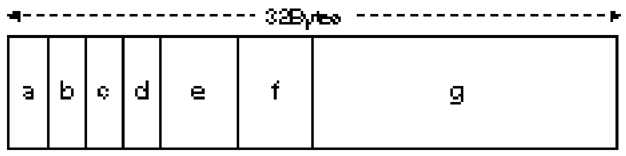
SecCount is the sector number in the frame, Nsectors is the number of sectors in the frame, and FrameCount is the frame number of the streaming data.

These values are used to ensure that the streaming library reads the frame data consecutively without missing any sectors.

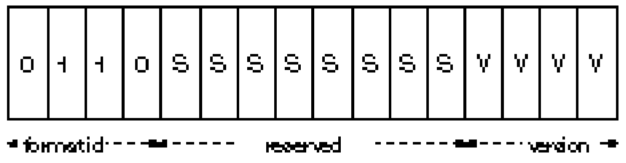
User

User is the user-defined field, which can be used as needed for various data types.

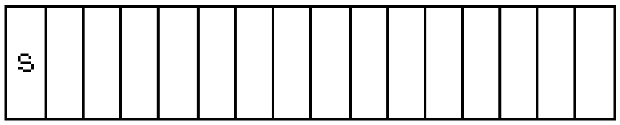
Figures 1–3: Sector header



a) ID: identifier, etc. (2 bytes)
Reserved for system



b) Type: data type (2 bytes)



s = 1 indicates system-defined format
0 indicates user-defined format

c) SecCount: sector number in frame (2 bytes)
d) NSectors: number of sectors in the frame (2 bytes)
e) FrameCount: Frame number of streaming data (4 bytes, starting from 1)
f) FrameSize: Frame size (in long words, 4 bytes)
g) User: User-defined area (16 bytes)

Note: All are little endian

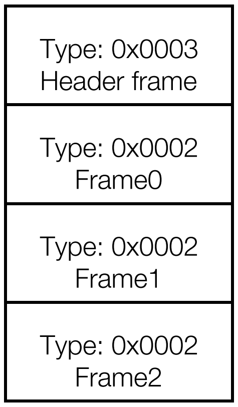
User-specified frame data

The user can set the MSB of the sector header's Type field to 0 to enable streaming of user-defined data.

If the entire run of streaming data uses the same attribute values, it is possible to use just one header (header frame) for the entire run instead of for each element of frame data, thereby reducing the total amount of data.

In streaming data that has a header frame, the Type field in the header frame may be changed to a Data field (see Figure 1-4).

Figure 1–4: Streaming data with header frame



In the example shown in the figure, the LSB in the Type field is used to distinguish frame data in a header frame from other non-header frame data.

When reading data from a CD-ROM, the Type field in the frame data read by the application can be freely accessed. This allows the interpretation of the data in the frame to change according to the type of frame.

Examples of the types of data saved in the header frame are: CLUT data that is not saved in each frame (for animation), a table of jumps to certain frames, and so on.

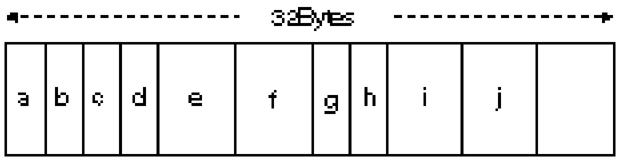
System-specified frame data

The following is currently provided as system-defined frame data.

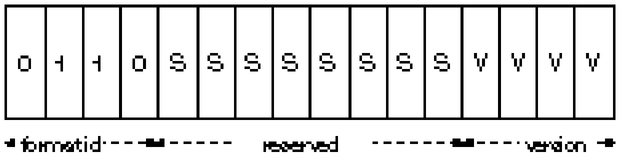
MDEC animation (Type: 0x8001)

Figure 1-5 shows an MDEC animation sector header.

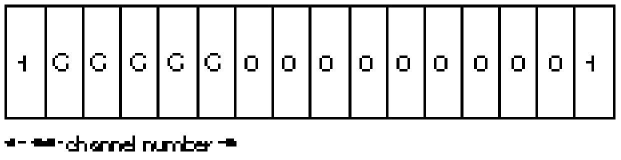
Figure 1–5: MDEC animation sector header



a) ID: Identifier, etc. (2 bytes)
Reserved for system



b) Type: data type (2 bytes)



C: Channel number (for multi-channel streaming); for ordinary streaming this number is 0.

c) SecCount: sector number in frame (2 bytes)
d) NSectors: number of sectors in the frame (2 bytes)
e) FrameCount: Frame number of streaming data (4 bytes, starting from 1)
f) FrameSize: Frame size (in long word units, 4 bytes)
g) Width: Width (2 bytes)
h) Height: Height (2 bytes)
i) HeadM: Reserved for system (4 bytes)
j) HeadV: Reserved for system (4 bytes)

Streaming data with audio

Successive audio playback

The PlayStation plays back ADPCM (Adaptive Differential Pulse Code Modulation) audio data as specified in the CD-ROM XA (eXtended Audio) standard (this is hereafter abbreviated as XA-ADPCM audio).

The PlayStation supports the following four types of XA-ADPCM audio.

Table 1–1: XA-ADPCM audio data types supported by PlayStation

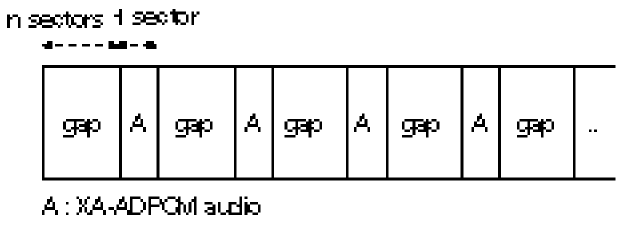
Sampling frequency	Stereo/monaural
37.8 Khz	Stereo
37.8 Khz	Monaural
18.9 Khz	Stereo
18.9 Khz	Monaural

XA-ADPCM audio is sent directly from the CD-ROM decoder to the SPU without using main memory.

The streaming format described above is used to read CD-ROM data into main memory. Consequently, it cannot be used for playing back XA-ADPCM audio.

When playing back XA-ADPCM audio, the data sectors on the CD-ROM must be arranged (interleaved) as shown in Figure 1-6.

Figure 1–6: Arrangement of data on CD-ROM for XA-ADPCM audio



The ratio of data sector size to gap size depends on the type of XA-ADPCM audio data and the CD-ROM's playback speed, as shown below.

Table 1–2: Data/gap ratios

CD-ROM playback speed	Type	Data/gap ratio
Double speed	37.8 kHz, stereo	1 sector/7 sectors
Double speed	37.8 kHz, monaural	1 sector/15 sectors
Double speed	18.9 kHz, stereo	1 sector/15 sectors
Double speed	18.9 kHz, monaural	1 sector/31 sectors
Standard speed	37.8 kHz, stereo	1 sector/3 sectors
Standard speed	37.8 kHz, monaural	1 sector/7 sectors
Standard speed	18.9 kHz, stereo	1 sector/7 sectors
Standard speed	18.9 kHz, monaural	1 sector/15 sectors

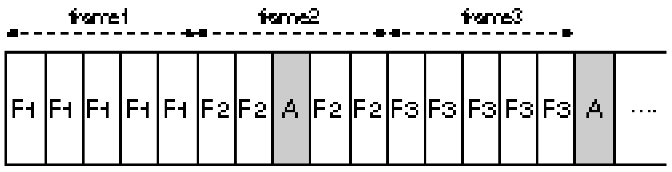
Streaming data with audio

Streaming data with audio means ordinary (non-audio) streaming data is interleaved with XA-ADPCM audio.

This kind of interleaved XA-ADPCM audio data must use the structure shown in Figure 1-6 above.

Accordingly, the structure of streaming data with audio has streaming data inserted in the "gap" sections shown in Figure 1-6. Figure 1-7 shows an example of this structure for streaming data with audio.

Figure 1–7: Streaming data with audio



Frame 1 Frame 2 Frame 3

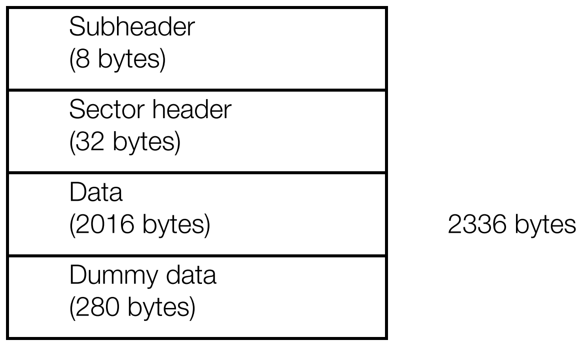
Streaming data: 30 fps (5 sectors/frame)

XA-ADPCM audio: 37.8 KHz, stereo

CD-ROM playback speed: Double speed

Note that the size of a sector of streaming data with audio is different from that of ordinary streaming data.

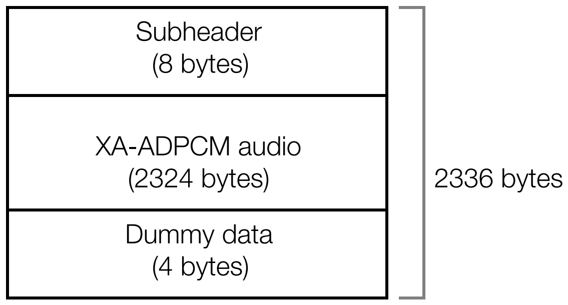
Figure 1–8: Data sector in streaming data with audio



The sector size is 2048 bytes for ordinary (non-audio) streaming data, and 2336 bytes for streaming data with audio.

Figures 1-8 and 1-9 show sectors used for streaming data with audio.

Figure 1–9: XA-ADPCM audio sectors for streaming data with audio



Note that the sector header and data areas in Figure 1-8 are exactly identical to their counterparts in Figure 1-2.

The subheader size is specified by the CD-ROM XA standard. This subheader contains information such as flags for distinguishing data sectors from audio sectors.

A dummy data section is added to the end of the sector to make the sector size for XA-ADPCM audio sectors the same as for non-XADPCM audio sectors.

XA: CD-ROM Voice Data

XA is the PlayStation CD-ROM XA voice data format. The typical extension in DOS is “.XA”.

The XA format is based on the following specifications. The XA file output by RAW2XA has a sub-header.

CD-ROM XA

SYSTEM DESCRIPTION CD-ROM XA

Copyright May 1991

Chapter 2: 3D Graphics

The tools used to create a 3D model create an RSD file (which is a text file) to store the model's data. To be used in a game this file must be converted to a binary, more compact and thus usable—called a TMD file.

RSD: Model Data

The RSD format for 3D model data is really a meta file; a collection of separate file formats that are used together to describe a single 3D model. There are four different types of files:

- RSD file: Describes relationships between PLY/MAT/GRP and texture files.
- PLY file: Describes positional information on vertices of a polygon.
- MAT file: Describes material information on a polygon.
- GRP file: Describes grouping information.

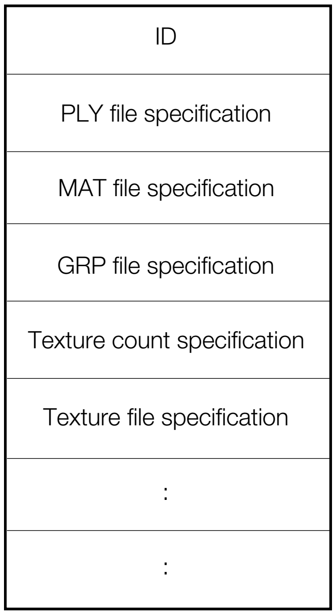
Information on how the data in the separate files is used together is specified by the RSD file. Because descriptive information is stored in multiple files, it's possible to have objects using different materials in a PLY file.

All files are ASCII text files with lines delimited by LF or CR/LF. Any line starting with a "#" is treated as a comment.

RSD File

The RSD file stores information on combinations of PLY, MAT and GRP files constituting a 3D object. A set of files is used to describe a single 3D object.

Figure 2–1: RSD File Structure



Sample RSD File Contents

The following gives a simple example of the RSD file.

```
@RSD940102
PLY=sample.ply
MAT=sample.mat
GRP=sample.grp
NTEX=3
TEX[0]=texture.tim
TEX[1]=texture2.tim
```

```
        TEX[2]=texture3.tim
```

ID

The ID is composed of a character string that indicates the version of the RSD file format, being "@RSDnnnnnn" (where nnnn is a number). The current version is "@RSD940102".

PLY File

PLY = (File name of PLY).

This is `SAMPLE.PLY` in the example.

MAT File

MAT = (File name of MAT).

This is `SAMPLE.MAT` in the example.

GRP File

GRP = (File name of GRP).

This is `SAMPLE.GRP` in the example.

Texture Count

Specifies the number of textures used.

NTEX = (Number of textures)

This is 3 in the example.

Texture File

Specifies an image data file in the TIM format to be used as the texture, and the same number of texture files as a value specified by above NTEX. (Note that while there are three in the example there can be as many as required.) This block does not exist in the RSD file for a model that uses no textures.

TEX[n] = (n-th texture file name)

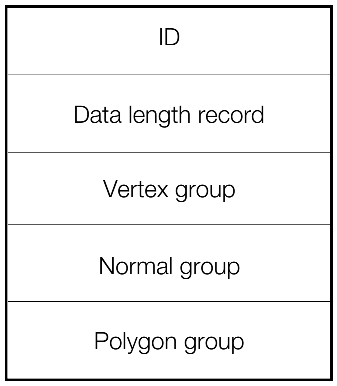
This is “TEXTURE.TIM”, “TEXTURE2.TIM”, and “TEXTURE3.TIM” in our example. The filename specifications must follow the appropriate format for the development system being used (MS-DOS, UNIX, Macintosh, etc.). Because of this, care should be taken when transferring files between platforms.

PLY File

The PLY file stores the positions of the vertices of polygons. The coordinate system for the PLY file is the same as for the extended library (libgs), with the X axis (forward) representing the right screen, the Y axis the bottom, and the Z axis the depth.

The direction (obverse or reverse) of a single-faced polygon is determined by the order in which the vertices are described in a polygon group. The obverse of the polygon is defined as the plane for which the vertices of a polygon are described clockwise.

Figure 2–2: PLY File Structure



Sample PLY File Contents

The following gives a simple example of a PLY file:P

```
@PLY940102
# Number of Items
8 12 12
# Vertex
0  0  0
0  0 100
0 100  0
0 100 100
100  0  0
100  0 100
100 100  0
100 100 100
# Normal
0.000000E+00 0.000000E+00 -1.000000E+00
0.000000E+00 0.000000E+00 -1.000000E+00
1.000000E+00 0.000000E+00 -0.000000E+00
1.000000E+00 0.000000E+00 0.000000E+00
0.000000E+00 0.000000E+00 1.000000E+00
0.000000E+00 0.000000E+00 1.000000E+00
-1.000000E+00 -0.000000E+00 -0.000000E+00
-1.000000E+00 0.000000E+00 0.000000E+00
-0.000000E+00 1.000000E+00 0.000000E+00
0.000000E+00 1.000000E+00 0.000000E+00
0.000000E+00 -1.000000E+00 0.000000E+00
0.000000E+00 -1.000000E+00 0.000000E+00
# Polygon
0 6 2 0 0 0 0 0 0
0 6 0 4 0 1 1 1 0
0 7 6 4 0 2 2 2 0
0 7 4 5 0 3 3 3 0
0 3 7 5 0 4 4 4 0
0 3 5 1 0 5 5 5 0
0 2 3 1 0 6 6 6 0
0 2 1 0 0 7 7 7 0
0 7 3 2 0 8 8 8 0
0 7 2 6 0 9 9 9 0
0 4 0 1 0 10 10 10 0
0 4 1 5 0 11 11 11 0
```

ID

This is a character string representing the version of a PLY file format, being "@PLYnnnnnn" (where nnnn is a number). The present version is "@PLY940102".

Data Length Record

Describes the number of data lines for the subsequent three data blocks. Items on each line are delimited by a tab or space character. In our sample, we specify 8 lines of data for the VERTEX group, and 12 lines each for the NORMAL and POLYGON groups.

Figure 2–3: PLY File Data Length Record

Number of vertices	Number of normals	Number of polygons

Vertex Group

A vertex group is composed of three floating-point values representing coordinates of a vertex. One line serves one vertex.

Figure 2–4: Vertex Descriptor for PLY File

x component	y component	z component

Normal Group

The normal is the direction used to calculate light shading. Thus the normal can be either perpendicular to a polygon's flat surface, the 'flat normal' (to give flat light shading), as in this example, or perpendicular to the vertex, the 'vertex normal', where three polygons join (giving gouraud shading).

A normal group is composed of three floating-point values representing the components of a normal vector.

Figure 2–5: Normal Descriptor for PLY File

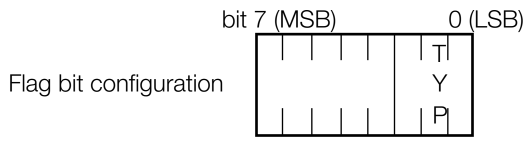
x component	y component	z component

Polygon Group

A polygon group is composed of a flag for representing the type of a polygon, and eight parameters constituting the polygon. The meaning of the parameters varies with the type of polygon specified in the flag.

Figure 2–6: PLY File Polygon Descriptor

Flag	Parameter #1	Parameter #2	. . .	Parameter #3



Polygon (Triangle or Quadrangle)

The parameter section describes the vertices and normals for the polygon. Each vertex value is an integer index, numbered from zero, to the proper position of the vertex data within the vertex group. Normal values are a similar index into the normal group.

For a polygon to be subjected to flat shading, the normal of each vertex has the same value, and the value of the first vertex is adopted. For a polygon to be subjected to smooth shading gourand, the normal of each vertex has a different value.

The flag is a hexadecimal integer value (although not prefixed with “0x”, as would be expected) that specifies the type of polygon. For a triangular polygon, the data for the fourth vertex and normal are assigned a value of zero. For a quadrangular polygon, the vertices are described in the proper order so that the first three vertices form a triangle, and the second through fourth vertices form another triangle (i.e. to subdivide the quad as shown in Figure 2-7).

Figure 2–7: Vertex ordering for quad subdivision

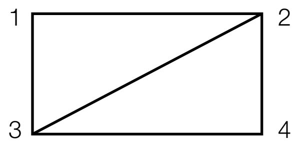


Figure 2–8: Polygon

Flag	Vertex 0	Vertex 1	Vertex 2	Vertex 3	Normal 0	Normal 1	Normal 2	Normal 3

Straight line

The parameter section describes the vertex numbers of two end points.

Figure 2–9: Straight Line

Flag	Vertex 0	Vertex 1	Vertex 2	Vertex 3	Normal 0	Normal 1	Normal 2	Normal 3

Sprite

A sprite in model data is rectangular image data located in a 3D space. It can be considered to be a textured polygon always facing the visual point.

The parameter section describes vertices indicating sprite positions, and the width and height of images (sprite patterns).

Figure 2–10: Sprite

Flag	Vertex 0	WIDTH	HEIGHT	0	0	0	0	0

MAT File

The MAT file defines the color and shading for each polygon.

Figure 2–11: MAT File Structure

ID
Number of materials
Material descriptor
:
:
:

Sample MAT File Contents

The following gives a simple example of the MAT file:

```
MAT940801
Number of Items
10
# Materials
0-50 F C 255 255 255
6  0 G T 1 10 0 25 71 40 25 0 0
7  0 G T 1 10 30 20 75 40 25 0 0
8  0 G T 1 18 73 30 79 40 25 0 0
9  0 G T 1 12 23 29 77 40 25 0 0
10 0 F T 1 18 13 75 72 40 25 0 0
11 0 F T 0 22 10 24 74 40 25 0 0
12 0 F T 0 30 39 41 79 40 25 0 0
13 1 F D 0 116 47 118 77 69 46 69 77 30 187 187
14 1 F H 0 69 46 69 77 17 45 15 77 101 210 138 52 211 188 101 210
```

ID

This is a character string representing the version of a MAT file format, being "@MATnnnnnn" (where nnnn is a number). The present version is "@MAT940801".

New attributes of colored texture and gradation texture unavailable to the past format (@MAT940102) are supported in @MAT940801.

Number of Items

Describes the number of subsequent material descriptors (lines).

Material Descriptor

Specifies a polygon and describes material information on the polygon.

Figure 2–12: Material Descriptor

Polygon no.	Flag	Shading	Material information

Polygon number

This is an index (starting from zero) for a polygon group described in a PLY file. Using a range specification allows two or more polygons to be described in one line. See Table 2-1.

Table 2–1: Polygon number

Description	Polygon of interest
1	1 only
0-5	0 1 2 3 4 5
2,4,6	2 4 6

Flag

This is a hexadecimal integer representing the type of a polygon. The flag is not provided with a prefix of '0x'. The following gives the meaning of each bit.

Bit 0: Light source calculation mode
0: Light source calculation supported
1: Fixed color

With light source calculation supported, the rendering color is determined by the angle between the direction of the light source and the surface of the polygon. Note that for fixed color, the color is constant irrespective of the direction of the light source.

Bit 1: Flag for Back face Culling
0: Single-faced polygon
1: Double-faced polygon

Bit 2: Flag for Semitransparent
0: Opaque
1: Semitransparent

With the flag set at 1, the polygon with no texture is always made to be semitransparent, and the polygon with texture is made to be semitransparent/opaque/ transparent depending on the STP bit of texture data.

Bits 3 to 5: Rate of semitransparency
000: 50% back + 50% polygon
001: 100% back + 100% polygon
010: 100% back - 100% polygon
011: 100% back + 25% polygon
1XX: reserved

The current library does not provide the capability to change the semitransparency rate of a polygon with no texture.

Bits 6 to 7: Reserved (Must be 0)

Shading

This is an ASCII character indicating the shading mode.

“F” = Flat shading (shading is based on the normal for the first vertex of the polygon, as specified in the PLY file)
“G” = Smooth shading

Material information

The format of the remainder of each line is different dpending on the material type. There are several different material types. Each is designated by a special type code, as follows:

Table 2–2

Type	Meaning
C	Colored polygon/straight line, no texture
G	Gradient filled polygon/straight line, no texture

T	Textured polygon/sprite
D	Colored textured polygon
H	Gradient (shaded) textured polygon

Figure 2–13: Texture not Supported (Colored Polygon/Straight Line)

TYPE	R	G	B

TYPE:	Material type, whose value is "C"
R, G, B:	RGB components of polygon color (0 to 255)

Figure 2–14: Texture not Supported (Gradation colored polygon/straight line)

TYPE	R0	G0	B0	R1	G1	B1	...	R3	G3	B3

TYPE:	Material type, whose value is "G"
Rn, Gn, Bn:	RGB components of the n-th vertex. For a triangular polygon, the RGB value of the fourth vertex is 0, 0, 0.

Figure 2–15: Textured Polygon/Sprite

TYPE	TWO	U0	V0	U1	V1	U2	V2	U3	V3

TYPE:	Material type, whose value is "T"
TNO:	TIM data file to be used (Texture number described in the RSD file)
Un, Vn:	Position of vertex n in the texture space. For a triangular polygon, the value (U3, V3) of the fourth vertex is zero.

Figure 2–16: Colored Textured Polygon

TYPE	TNO	U0	V0	U1	V1	U2	V2	U3	V3	R	G	B

TYPE:	Material type, whose value is "D"
TNO:	TIM data file to be used. (Texture number described in the RSD file)
Un, Vn:	Position of vertex n in the texture space. For a triangular polygon, the value (U3, V3) of the fourth vertex is zero.
R, G, B:	RGB components of polygon color (0 to 255)

* The colored textured polygon is used to make the texture of a polygon bright without light source calculation. This type allows the three-dimensional drawing of a textured object without light source calculation. It is valid only in the fixed color light source calculation mode.

Figure 2–17: Gradation Textured Polygon

TYPE	TNO	U0	V0	U1	V1	U2	V2	U3	V3

R0	G0	B0	R0	G1	B1	...	R3	G3	B3

TYPE:	Material type, whose value is "H"
TNO:	TIM data file to be used. (Texture number described in the RSD file)
Un, Vn:	Position of vertex n in the texture space. For a triangular polygon, the value (U3, V3) of the fourth vertex is zero.
Rn, Gn, Bn:	RGB components of the n-th vertex (N = 0 to 3). For a triangular polygon, the RGB value of the fourth vertex is 0, 0, 0.

* The gradation textured polygon is used to provide the same effect as textured smooth shading without light source calculation. This type is valid only in the fixed color light source calculation mode.

GRP File

A group of polygons in the PLY file can be assigned a name. For example, the polygons used to make up a steering wheel can be grouped and given the name ‘wheel’.

Thus, a group of polygons can be operated by the material editor, and certain polygons can be accessed from the program.

Figure 2–18: GRP File Structure

ID
Number of groups
Group descriptor
. . . .

ID

This is a character string representing the version of a GRP file, being "@GRPnnnnnn" (where nnnn is a number). The current version is "@GRP940102".

Number of Groups

Covers the number of subsequent group descriptors.

Group Descriptor

Defines the configuration of a group. A group descriptor is composed of two or more lines.

Start line

Figure 2–19: GRP Descriptor (Start line)

Group name	Polygon No. line count	Number of polygons

Group name:	Name assigned to a group
Polygon No. line count:	Number of subsequent lines for polygon No. description
Number of polygons:	Number of polygons belonging to a group

Subsequent line (for polygon No. description)

Specifies the numbers of polygons belonging to a group. The value indicates the position of a polygon in the PLY file. Range specification allows two or more polygons to be described in one line.

Table 2–3

Description	Polygon of interest
1	1 only
3-7	3 4 5 6 7
2,4,6	2 4 6

TMD: Modeling Data for OS Library

The TMD format contains 3D modeling data which is compatible with the PlayStation expanded graphics library (libgs). TMD data is downloaded to memory and may be passed as an argument to functions provided by LIBGS. TMD files are created using the RSDLINK utility, which reads an RSD file created by the SCE 3D Graphics Tool or a comparable program.

The data in a TMD file is a set of graphics primitives—polygons, lines, etc.—that make up a 3D object. A single TMD file can contain data for one or more 3D objects.

Coordinate Values

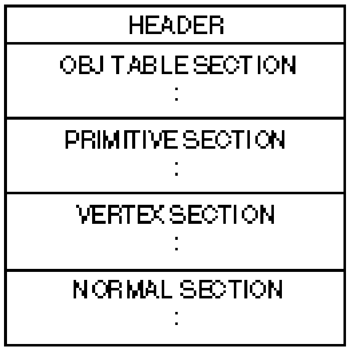
Coordinate values in the TMD file follow the 3D coordinate space handled by the 3D graphics library. The positive direction of the X axis represents the right, the Y axis the bottom, and the Z axis the depth. The spatial coordinate value of each object is a signed 16-bit integer value ranging from -32768 to +32767.

In the 3D object design phase and within the RSD format, the vertex information is stored as a floating point value. Conversion from RSD into TMD involves converting and scaling vertex values as needed. The scale used is reflected in the object structure, described later, as the reference value. This value can provide an index for mapping from object to world coordinates. The current version of LIBGS ignores the scale value.

File Format

TMD files are configured by 4 blocks. They have 3 dimensional object tables, and 3 types of data entities—PRIMITIVE, VERTEX, and NORMAL—which configure these.

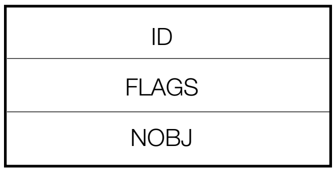
Figure 2–20: TMD File Format



HEADER

The header section is composed of three word (12 bytes) data carrying information on data structure.

Figure 2–21: Structure of Header

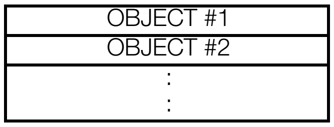


ID:	Data having 32 bits (one word). Indicates the version of a TMD file. The current version is 0x00000041.
FLAGS:	Data having 32 bits (one word). Carries information on TIM data configuration. The least significant bit is FIXP. The other bits are reserved and their values are all zero. The FIXP bit indicates whether the pointer value of the OBJECT structure described later is a real address. A value of one means a real address. A value of zero indicates the offset from the start.
NOBJ:	Integral value indicating the number of objects

OBJ TABLE

The OBJ TABLE block is a table of structures holding pointer information indicating where the substance of each object is stored. Its structure is as shown below.

Figure 2–22: OBJ TABLE structure



The object structure has the following configuration:

```
struct object
{
   u_long *vert_top;
   u_long n_vert;
   u_long *normal top;
   u_long n_normal;
   u_long *primitive top;
   u_long n_primitive;
   long scale;
}
```

(Explanation of members)

vert_top:	Start address of a vertex
n_vert:	Number of vertices

normal_top:	Start address of a normal
n_normal:	Number of normals
primitive_top:	Start address of a primitive
n_primitive:	Number of primitives

Among the members of the structure, the meanings of the pointer values (vert_top, normal_top, primitive_top) change according to the value of the FIXP bit in the HEADER section. If the FIXP bit is 1, they indicate the actual address, and if the FIXP bit is 0, they indicate a relative address taking the top of the OBJECT block as the 0 address.

The type of the scaling factor is "signed long", and its value raised to the second power is the scale value. That is to say, if the scaling factor is 0, the scale value is an equimultiple; if the scaling factor is 2, the scale value is 4; if the scaling factor is -1, the scale value is 1/2. Using this value, it is possible to return to the scale value at the time of design.

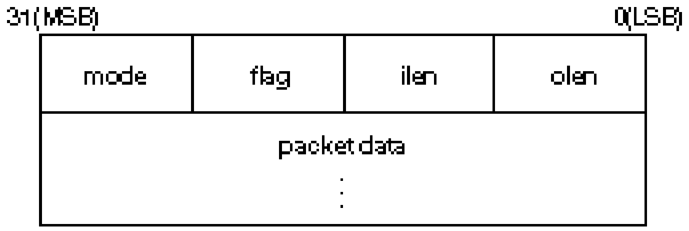
PRIMITIVE

The PRIMITIVE section is an arrangement of the drawing packets of the structural elements (primitives) of the object. One packet stands for one primitive (see Figure 2-23).

The primitives defined in TMD are different from the drawing primitives handled by libgpu. A TMD primitive is converted to a drawing primitive by undergoing perspective transformation processing performed by the libgs functions.

Each packet is of variable length, and its size and structure vary according to the primitive type.

Figure 2–23: Drawing Packet General Structure

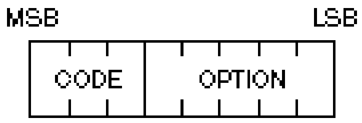


Each item in Figure 2-23 is as follows:

Mode (8 bit)

Mode indicates the type of primitive and added attributes. They have the following bit structure:

Figure 2–24: Mode



CODE: 3 bit code expressing entities
001 = polygon (triangle, quadrilateral)
010 = Straight line
011 = Sprite

OPTION: varies with the option, bit and CODE values
(Listed with the list of packet data configurations described later)

Flag (8 bit)

Flag indicates option information when rendering and has the following bit configuration:

Figure 2–25: Flag

MSB LSB

0	0	0	0	0	GRD	FCE	LGT

GRD: Valid only for the polygon not textured, subjected to light source calculation
1: Gradation polygon
0: Single-color polygon

FCE: 1: Double-faced polygon
0: Single-faced polygon
(Valid, only when the CODE value refers to a polygon.)

LGT: 1: Light source calculation not carried out
0: Light source calculation carried out

Ilen (8 bit)

Indicates the length, in words, of the packet data section.

Olen (8 bit)

Indicates the word length of the 2D drawing primitives that are generated by intermediate processing.

Packet Data

Parameters for verices and normals. Content varies depending on type of primitive. Please refer to “Packet data configuration” which will be discussed later.

VERTEX

The vertex section is composed of a set of structures representing vertices. The following gives the format of one structure.

Figure 2–26: Vertex Structure

MSB	LSB
VY	VX
--	VZ

VX, VY, XZ: x, y and z values of vertex coordinates (16-bit integer)

NORMAL

The normal section is composed of a set of structures representing normals. The following gives the format of one structure.

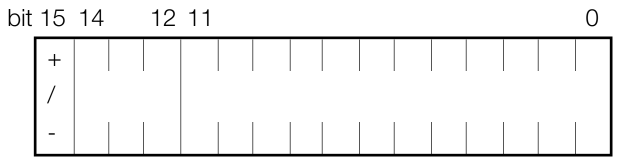
Figure 2–27: Normal Structure

MSB	LSB
NY	NX
--	NZ

NX, NY, NZ: x, y and z components of a normal (16-bit fixed-point value)

NX, NY and NZ values are signed 16-bit fixed-point values where 4096 is considered to be 1.0.

Figure 2–28: Fixed-Point Format



Sign:	1 bit
Integral part:	3 bits
Decimal part:	12 bits

Packet Data Composition Table

This section lists packet data configurations for each primitive type.

The following parameters are contained in the packet data section:

Vertex(n):

Index value of 16-bit length pointing to a vertex. Indicates the position of the element from the start of the vertex section for an object covering the polygon.

Normal(n)

Index value of 16-bit length pointing to a normal. Same as Vertex.

Un, Vn

X and Y coordinate values on the texture source space for each vertex

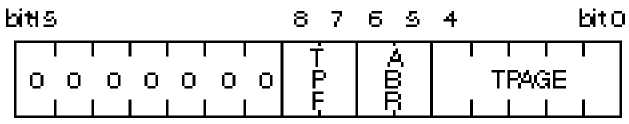
Rn, Gn, Bn

RGB value representing polygon color being an unsigned 8-bit integer. Without light source calculation, the predetermined brightness value must be entered.

TSB

Carries information on a texture/sprite pattern.

Figure 2–29: TSB

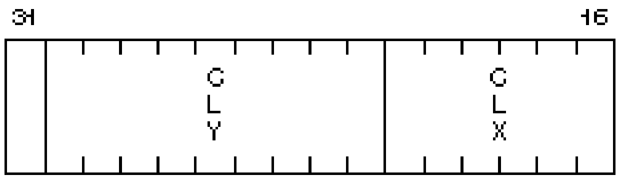


TPAGE:	Texture page number (0 to 31)
ABR:	Semitransparency rate (Mixture rate). Valid, only when ABE is 1. 00 50%back + 50%polygon 01 100%back + 100%polygon 10 100%back - 100%polygon 11 100%back + 25%polygon
TPF:	Color mode 00 4 bit

01	8 bit
10	15 bit

CBA: Indicates the position where CLUT is stored in the VRAM.

Figure 2–30: CBA



CLX: Upper six bits of 10 bits of X coordinate value for CLUT on the VRAM

CLY: Nine bits of Y coordinate value for CLUT on the VRAM

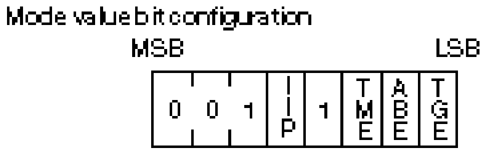
Packet Data Configuration Example-3 Vertex Polygon with Light Source Calculation

A 3 vertex polygon with light source calculation is shown below. The mode and flag values in this example express a one sided polygon with translucency in the OFF state.

Bit Configuration of Mode Value

The mode value bit configuration of the primitive section is as follows:

Figure 2–31: Mode Structure



IIP:Shading mode
- 0: Flat shading
- 1: Gouraud shading

TME: Texture specification
- 0: Off
- 1: On

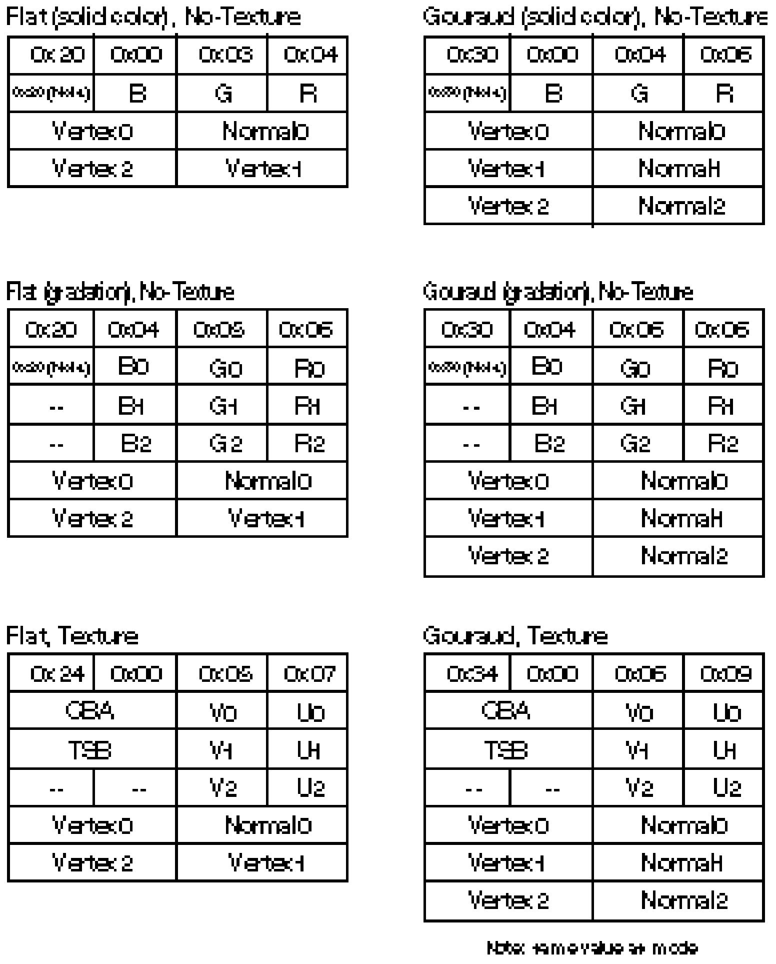
ABE: Translucency processing
- 0: Off
- 1: On

TGE: Brightness calculation at time of texture mapping
- 0: On
- 1: Off (Draws texture as is)

Packet Data Configuration

Packet data configuration is as follows:

Figure 2–32: Packet Data for Polygons



Flat (solid color), No-Texture

0x20	0x00	0x03	0x04
0x20 (N+4)	B	G	R
Vertex0		Normal0	
Vertex 2		Vertex1	

Gouraud (solid color), No-Texture

0x30	0x00	0x04	0x06
0x30 (N+4)	B	G	R
Vertex0		Normal0	
Vertex1		Normal1	
Vertex 2		Normal2	

Flat (gradation), No-Texture

0x20	0x04	0x05	0x06
0x20 (N+4)	B0	G0	R0
--	B1	G1	R1
--	B2	G2	R2
Vertex0		Normal0	
Vertex 2		Vertex1	

Gouraud (gradation), No-Texture

0x30	0x04	0x06	0x06
0x30 (N+4)	B0	G0	R0
--	B1	G1	R1
--	B2	G2	R2
Vertex0		Normal0	
Vertex1		Normal1	
Vertex 2		Normal2	

Flat, Texture

0x24	0x00	0x05	0x07
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
Vertex0		Normal0	
Vertex 2		Vertex1	

Gouraud, Texture

0x34	0x00	0x06	0x09
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
Vertex0		Normal0	
Vertex1		Normal1	
Vertex 2		Normal2	

Note: same value as mode

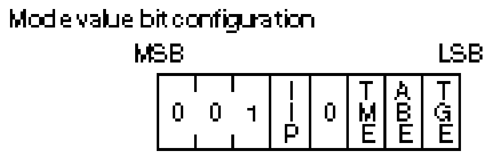
In the above example, the values of mode and flag indicate a single-faced polygon and semitransparency processing not carried out.

Packet Data Configuration Example-Polygon with 3 Vertices and No Light Source Calculation

Bit Configuration of Mode Value

The primitive section mode value bit configuration is shown below. For the value of each bit please refer to "3 vertex polygon with light source calculation."

Figure 2–33: Mode Byte



Mode value bit configuration

MSB							LSB
0	0	1	IIP	0	TME	ABE	TGE

Packet Data Configuration

Packet date configuration will be as follows:

Figure 2–34

Flat, No Texture

0x21	0x01	0x03	0x04
Note	B	G	R
Vertex1		Vertex0	
--		Vertex2	

Gouraud, No Texture

0x31	0x01	0x05	0x06
Note	B0	G0	R0
--	B1	G1	R1
--	B2	G2	R2
Vertex1		Vertex0	
--		Vertex2	

Flat, Texture

0x25	0x01	0x06	0x07
CBA		V0	U0
TSB		V1	U1
--	–	V2	U2
--	B	G	R
Vertex1		Vertex0	
--		Vertex2	

Note: the same value as mode.

Gouraud, Texture

0x35	0x01	0x08	0x09
CBA		V0	U0
TSB		V1	U1
--	–	V2	U2
--	B0	G0	R0
--	B1	G1	G1
--	B2	G2	G2
Vertex1		Vertex0	
--		Vertex2	

Packet Data Configuration Example-Polygon with 4 Vertices and Light Source Calculation

Bit Configuration of Mode Value

The primitive section mode value bit configuration is shown below. For the value of each bit please refer to "3 vertex polygon with light source calculation."

Figure 2–35: Mode Byte

Mode value bit configuration

MSB							LSB
0	0	1	IIP	1	TME	ABE	TGE

Note: Bit 3 is set to 1 to designate a 4-vertex primitive.

Packet Data Configuration

Packet data configuration is as follows:

Figure 2–36: Mode

Flat (solid color), No-Texture

0x28	0x00	0x04	0x08
0x28 (N+4)	B	G	R
Vertex0		Normal0	
Vertex 2		Vertex1	
--		Vertex3	

Gouraud (solid color), No-Texture

0x38	0x00	0x06	0x08
0x38 (N+4)	B	G	R
Vertex0		Normal0	
Vertex1		Normal1	
Vertex1		Normal2	
Vertex 2		Normal3	

Flat, (gradation), No-Texture

0x28	0x04	0x07	0x08
0x28 (N+4)	B0	G0	R0
--	B1	G1	R1
--	B2	G2	R2
--	B3	G3	R3
Vertex0		Normal0	
Vertex 2		Vertex1	
--		Vertex3	

Gouraud (gradation), No-Texture

0x38	0x04	0x08	0x08
0x38 (N+4)	B0	G0	R0
--	B1	G1	R1
--	B2	G2	R2
--	B3	G3	R3
Vertex0		Normal0	
Vertex1		Normal1	
Vertex 2		Normal 2	
Vertex3		Normal3	

Flat, Texture

0x2c	0x00	0x07	0x09
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
--	--	V3	U3
Vertex0		Normal0	
Vertex 2		Vertex1	
--		Vertex3	

Gouraud, Texture

0x3c	0x00	0x08	0x0c
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
--	--	V3	U3
Vertex0		Normal0	
Vertex1		Normal1	
Vertex 2		Normal2	
Vertex3		Normal3	

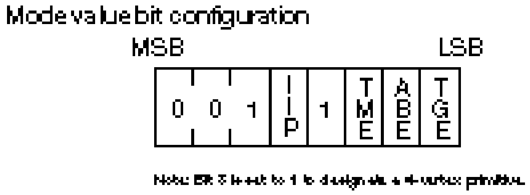
Note: same value as mode

Packet data configuration example-Polygon with 4 Vertices and No Light Source Calculation

Bit Configuration of Mode Value

The primitive section mode value bit configuration is shown below. For the value of each bit please refer to "3 angle polygon with light source calculation."

Figure 2–37: Mode Byte


Mode value bit configuration
MSB
LSB
0
0
1
IIP
1
TME
ABE
TGE
Note: Bit 3 is set to 1 to designate a 4-vertex primitive.

Packet Data Configuration

Figure 2–38: Packet Data

Flat, No Texture

0x29	0x01	0x03	0x05
Note	B	G	R
Vertex1		Vertex0	
Vertex3		Vertex2	

Gouraud, No Texture

0x39	0x01	0x06	0x08
Note	B0	G0	R0
--	B1	G1	R1
--	B2	G2	R2
--	B3	G3	R3
Vertex1		Vertex0	
Vertex3		Vertex2	

Flat, Texture

0x2d	0x01	0x07	0x09
CBA		V0	U0
TSB		V1	U1
--	—	V2	U2
--	—	V3	U3
--	B	G	R
Vertex1		Vertex0	
Vertex3		Vertex2	

Note: Has same value as mode.

Gouraud, Texture

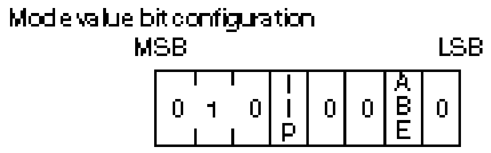
0x3d	0x01	0x0a	0x0c
CBA		V0	U0
TSB		V1	U1
--	—	V2	U2
--	—	V3	U3
--	B0	G0	R0
--	B1	G1	R1
--	B2	G2	R2
--	B3	G3	R3
Vertex1		Vertex0	
Vertex3		Vertex2	

Packet Data Configuration Example-Straight Line

Bit Configuration of Mode Value

The primitive section mode value bit configuration is as follows:

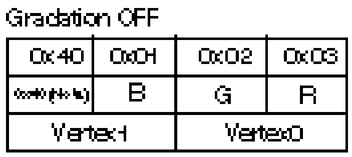
Figure 2–39: Mode



IIP:	With or without gradation 0: Gradation off (Monochrome) 1: Gradation on
ABE:	Translucency processing on/off 0: off 1: on

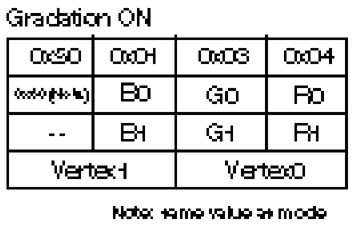
Packet Data Configuration

Figure 2–40: Packet Configuration for “Straight Line”



Gradation OFF

0x40	0x01	0x02	0x03
[illegible]	B	G	R
Vertex1		Vertex0	



Gradation ON

0x50	0x01	0x03	0x04
[illegible]	B0	G0	R0
--	B1	G1	R1
Vertex1		Vertex0	

Note: same value as mode

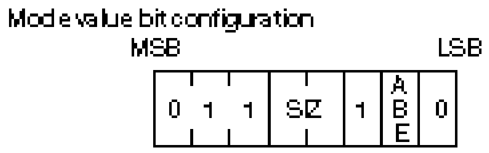
Packet Data Configuration Example - 3 Dimensional Sprite

A 3 dimensional sprite is a sprite with 3-D coordinates and the drawing content is the same as a normal sprite.

Bit Configuration of Mode Value

The primitive section mode value bit configuration is as follows:

Figure 2–41: Mode

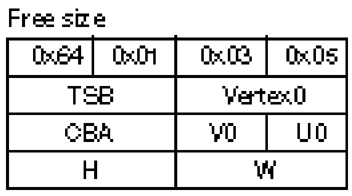


SIZ:	Sprite size 00: Free size (Specified by W, H) 01: 1 x 1 10: 8 x 8 11: 16 x 16
ABE:	Translucency processing 0: Off 1: On

Packet Data Configuration

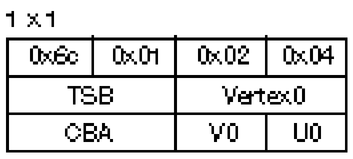
Packet data configuration is as follows:

Figure 2–42: Packet Data for Sprites



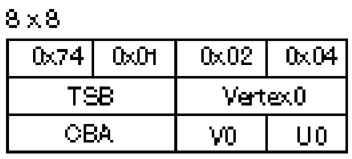
Free size

0x64	0x01	0x03	0x05
TSB		Vertex0	
CBA		V0	U0
H		W	



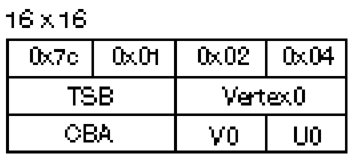
1 x 1

0x6c	0x01	0x02	0x04
TSB		Vertex0	
CBA		V0	U0



8 x 8

0x74	0x01	0x02	0x04
TSB		Vertex0	
CBA		V0	U0



16 x 16

0x7c	0x01	0x02	0x04
TSB		Vertex0	
CBA		V0	U0

PMD: High-Speed Modeling Data

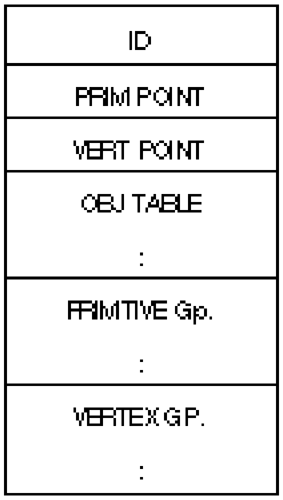
The PMD format is used for modeling data supported by the extended graphics library (libgs). The PMD format has a narrower range of functions than the TMD format, but this smaller scope enables faster processing.

PMD format handles the following kinds of objects.

- Triangular and rectangular polygons only
- Packet creation areas contained in the data
- Groups of polygons having the same attributes

The PMD file format consists of a table of 3D objects along with their PRIMITIVE and VERTEX descriptions.

Figure 2–43: Overall structure of PMD files



ID:	32-bit word containing the version of the PMD file.
	For the current version, this is 0x00000042.
PRIM POINT:	A 32-bit integer indicating the offset from the start of the PRIMITIVE Gp section of the file.
VERT POINT:	A 32-bit integer indicating the offset from the start of the VERTEX Gp section of the file.
	Enter "0" for an independent vertex.
OBJ TABLE:	The array of objects.
PRIMITIVE Gp:	A collection (primitive group) of polygons having the same attributes.
VERTEX Gp:	An array of vertex coordinates. VERTEX groups exist only in the case of shared vertices.

Coordinate Values

Coordinate values in the PMD file follow the 3D coordinate space handled by the 3D graphics library. The positive direction of the X axis represents the right, the Y axis the bottom, and the Z axis the depth. The spatial coordinate value of each object is a signed 16-bit integer value ranging from -32768 to +32767.

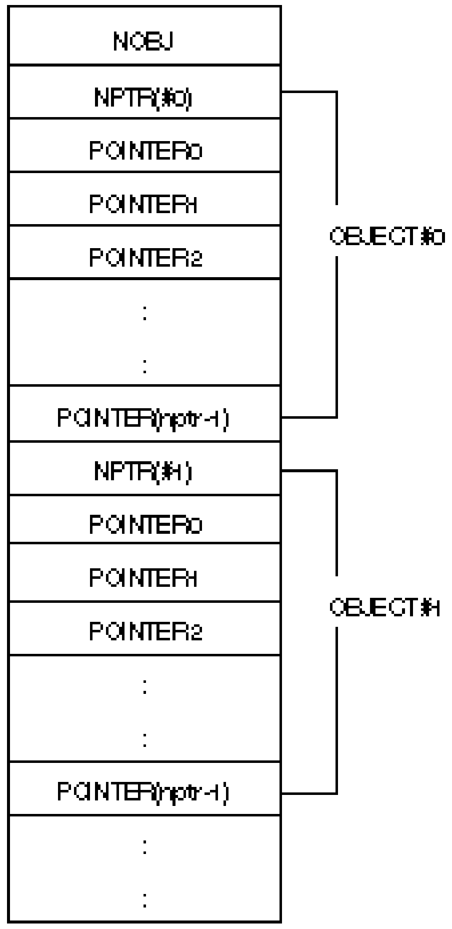
In the 3D object design phase and within the RSD format, the vertex information is stored as a floating point value. Conversion from RSD into PMD involves converting and scaling vertex values as needed. The scale used is reflected in the object structure, described later, as the reference value. This value can provide an index for mapping from object to world coordinates. The current version of LIBGS ignores the scale value.

OBJ TABLE

OBJ TABLE is a table that contains pointer information regarding the PRIMITIVE Gp for a particular object.

A single object is composed of primitive groups.

Figure 2–44: OBJECT Structure



NOBJ:	Number of objects in OBJ TABLE
NPTR:	Number of pointers in a single object
POINTER:	Pointer to a primitive group

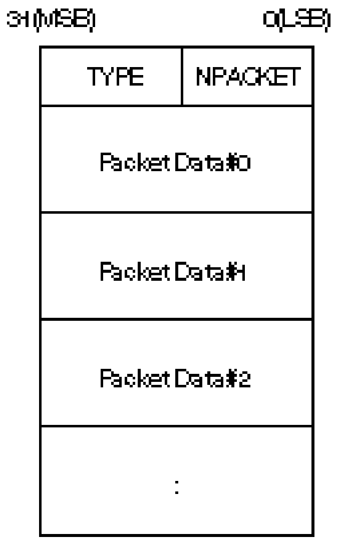
PRIMITIVE Gp

A PRIMITIVE Gp is a group of object structural element (primitive) graphics packets; a single packet contains one primitive.

Primitives defined by PMD are different from drawing primitives handled by libgpu. When PMD primitives undergo perspective transformation by libgs functions, they are converted to drawing primitives.

Each PRIMITIVE Gp has the following structure.

Figure 2–45: Packet Gp structure



TYPE : Packet type (see Table 2-46)

NPACKET : Number of packets

Figure 2–46: TYPE bit layout

Bit no.	When 0	When 1
16	Triangle	Quadrilateral
17	Flat	Gouraud
18	Texture-On	Texture-Off
19	Independent vertex	Shared vertex
20	Light source calculation Off	Light source calculation On
21	Back clip	No back clip
22-31	(Reserved for system)	

Packet Data structures change with the value of TYPE. Packet Data structure are broken down by type.

The POLY_. . . primitive group structure comes in a set of two which corresponds to a double buffer The contents of both of these must be initialized in advance. Bits 20 and 21 have no effect on packet data structure.

The pkt in each structure indicates a corresponding drawing primitive packet, the vertex coordinate value of v1~v4, and the values of vp1~vp4 offset from the start of the shared vertex row.

TYPE=00 (Triangle/Flat/Texture-On/Independent vertex)

```
struct _poly_ft3 {
   POLY_FT3 pkt[2];
   SVECTOR v1, v2, v3;
}
```

TYPE=01 (Quadrangle/Flat/Texture-On/Independent vertex)

```
struct _poly_ft4 {
Å@POLY_FT4 pkt[2];
Å@SVECTOR v1, v2, v3, v4;
}
```

TYPE=02 (Triangle/Gouraud/Texture-On/Independent vertex)

```
struct _poly_gt3 {
   POLY_GT3 pkt[2];
```

```
        SVECTOR v1, v2, v3;
    }
```

TYPE=03 (Quadrangle/Gouraud/Texture-On/Independent vertex)

```
    struct _poly_gt4 {
        POLY_GT4 pkt[2];
        SVECTOR v1, v2, v3, v4;
    }
```

TYPE=04 (Triangle/Flat/Texture-Off/Independent vertex)

```
    struct _poly_f3 {
        POLY_F3 pkt[2];
        SVECTOR v1, v2, v3;
    }
```

TYPE=05 (Quadrangle/Flat/Texture-Off/Independent vertex)

```
    struct _poly_f4 {
        POLY_F4 pkt[2];
        SVECTOR v1, v2, v3, v4;
    }
```

TYPE=06 (Triangle/Gouraud/Texture-Off/Independent vertex)

```
    struct _poly_g3 {
        POLY_G3 pkt[2];
        SVECTOR v1, v2, v3;
    }
```

TYPE=07 (Quadrangle/Gouraud/Texture-Off/Independent vertex)

```
    struct _poly_g4 {
        POLY_G4 pkt[2];
        SVECTOR v1, v2, v3, v4;
    }
```

TYPE=08 (Triangle/Flat/Texture-On/Shared vertex)

```
    struct _poly_ft3c {
        POLY_FT3 pkt[2];
        long vp1, vp2, vp3;
    }
```

TYPE=09 (Quadrangle/Flat/Texture-On/Shared vertex)

```
    struct _poly_ft4c {
        POLY_FT4 pkt[2];
        long vp1, vp2, vp3, vp4;
    }
```

TYPE=0a (Triangle/Gouraud/Texture-On/Shared vertex)

```
    struct _poly_gt3c {
        POLY_GT3 pkt[2];
        long vp1, vp2, vp3;
    }
```

TYPE=0b (Quadrangle/Gouraud/Texture-On/Shared vertex)

```
    struct _poly_gt4c {
        POLY_GT4 pkt[2];
        long vp1, vp2, vp3, vp4;
    }
```

TYPE=0c (Triangle/Flat/Texture-Off/Shared vertex)

```
struct _poly_f3c {
   POLY_F3 pkt[2];
   long vp1, vp2, vp3;
}
```

TYPE=0d (Quadrangle/Flat/Texture-Off/Shared vertex)

```
struct _poly_f4c {
   POLY_F4 pkt[2];
   long vp1, vp2, vp3, vp4;
}
```

TYPE=0e (Triangle/Gouraud/Texture-Off/Shared vertex)

```
struct _poly_g3c {
   POLY_G3 pkt[2];
   long vp1, vp2, vp3;
}
```

TYPE=0f (Quadrangle/Gouraud/Texture-Off/Shared vertex)

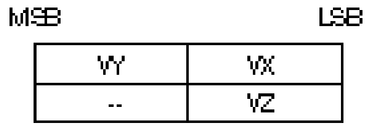
```
struct _poly_g4c {
   POLY_G4 pkt[2];
   long vp1, vp2, vp3, vp4;
}
```

pkt[ ] indicates the corresponding rendering primitive packet.
v1 to v4 indicates coordinate values of vertices.
vp1 to vp4 indicate offsets from the start of a row of shared vertices.

VERTEX

The VERTEX group is an SVECTOR structure array with shared vertices. The format of one of these structures is shown below.

Figure 2–47: VERTEX structure



VX, VY, VZ: The X, Y, and Z values of the vertex coordinates (16 bit integers)

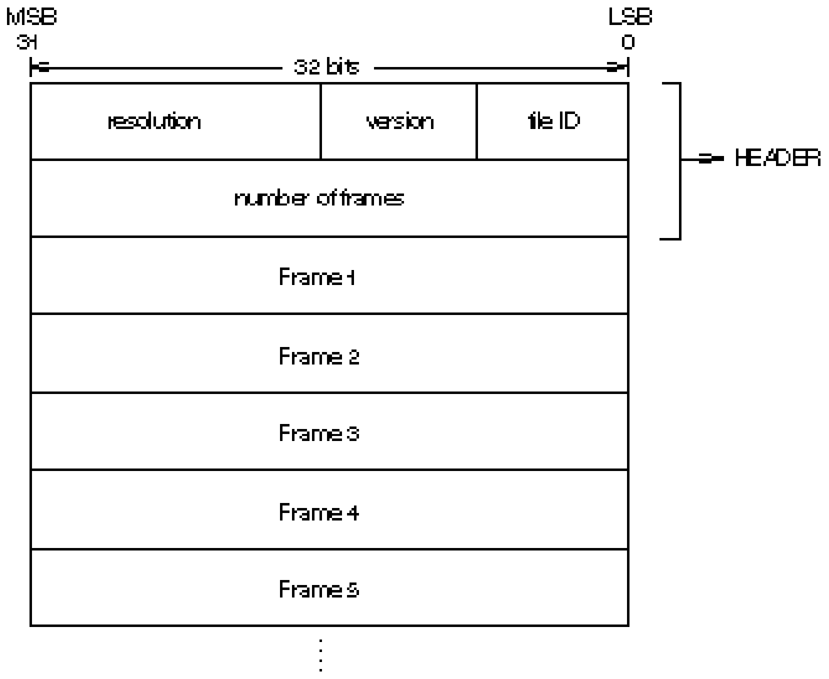
TOD: Animation Data

TOD format is used for specifying information along the flow of time, relative to a 3-dimensional object. It corresponds to the extended graphics library (libgs).

To be more precise, for each frame in a 3-dimensional animation (or frame sequence), the TOD file describes the required data relating to the 3-dimensional objects to be created, modified, or erased, and arranges the data for each frame along the flow of time.

A TOD file, as shown below, consists of a file header followed by frame data.

Figure 2–48: TOD file format



Header

At the top of the TOD file, there is a 2-word (64-bit) HEADER, in which the following four kinds of information are described.

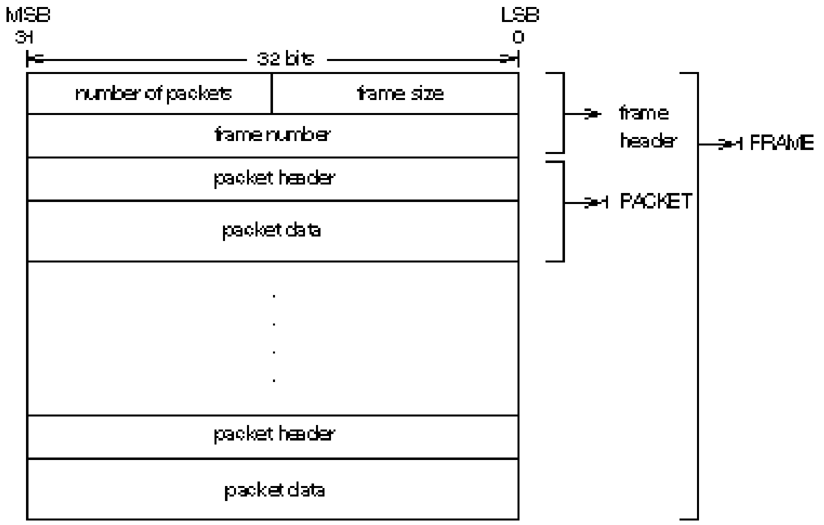
(a) File ID (8 bits)
This identifies the file as an animation file. Its value is 0x50.
(b) Version (8 bits)
Animation version. Its value starts at 0x00.
(c) Resolution (16 bits)
This is the time in which 1 frame is displayed (in units of ticks (1 tick = 1/60 seconds)).
(d) Number of frames (32 bits)
This is the number of frames described in the file.

Frame

Following the header the frame is described. Frames are arranged chronologically

Each FRAME consists of a frame header followed by a PACKET, as shown below.

Figure 2–49: Frame



Frame Header

There is a 2 word frame header at the beginning of each frame. The following information is described in a frame header.

- Frame size (16 bits)
  Frame length (including header) in words.
- Number of packets (16 bits)
  Number of packets.
- Frame numbers (32 bits)
  Frame number.

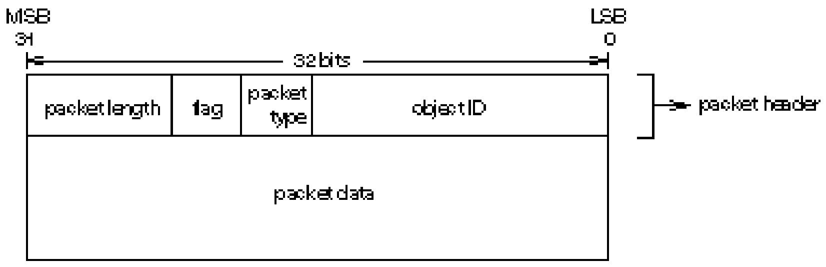
PACKET

After the frame header come the PACKETS. Each PACKET consists of a one-word packet header at the top, followed by the packet data (see Figure 2-50). There are several different kinds of PACKETS.

The size of the packet data in each PACKET will of course be different if the PACKETS are of different kinds; even if the PACKETS are of the same kind, the size of the packet data may be different.

A PACKET consists of a packet header and packet data, as shown below.

Figure 2–50: PACKET



Packet Header

The PACKET header contains the following information.

- Object ID (16 bits)
  The identification of the object to be handled.
- Packet type (4 bits)
  The type of packet data.

- Flag (4 bits)
  The meaning of the flag varies from packet to packet.
- Packet length (8 bits)
  This is the size of the packet (including the header) in units of words (4 bytes).

Object refers to a 3-dimensional object (a GsDOBJ2 structure) handled by libgs (the extended graphics library) which is to be made to reflect the packet data.

Packet type contains the classification of the data stored in the packet data. The significance of the flag varies according to the packet type.

Packet length indicates the length of the packet in units of words (4 bytes).

Packet Data

Several kinds of data, such as the GsCOORDINATE2 structure RST value and the TMD data ID (the modeling data ID), are stored in the packet data.

The packet type slot in the header indicates which type the PACKET is. The relationship between the packet type value and the type of data is as follows:

Figure 2–51: packet type values and packet data contents

0	Attribute
1	Coordinate (RST)
10	TMD data ID
11	Parent object ID
100	Matrix value
101	TMD data
110	Light source
111	Camera
1000	Object control
1001 – 1101	User defined
1110	System reserved
1111	Special commands

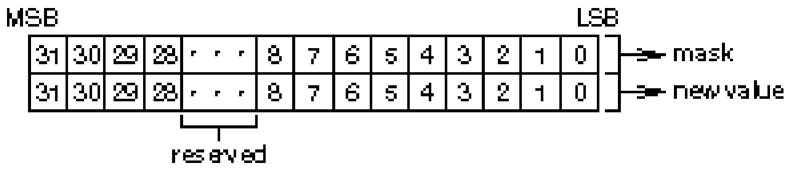
The different kinds of data are explained below.

Packet Data-Attribute

When packet type is 0000, the data that designates attribute of the GsDOBJ structure in the packet data is stored. In this case a flag is not used.

Packet data is composed of 2 words as follows:

Figure 2–52: Packet Data Configuration when Attribute



The first word is a mask which indicates the section which changes value and the section which does not change value. 0 is set in the bit which corresponds to the item which will change and 1 is set in the bit for the value which will not change.

In the second word, new data is input to the bits corresponding to items which are going to change, and the other bits are set to 0.

Note that the first and second words differ in the following respect: in the first word, the default value for the bits which are not going to be changed is 1, while in the second word, this default value is 0.

The breakdown of the bits of the second word packet data shown in Figure 2-52 is described below.

Table 2–4: Packet data bit-by-bit breakdown

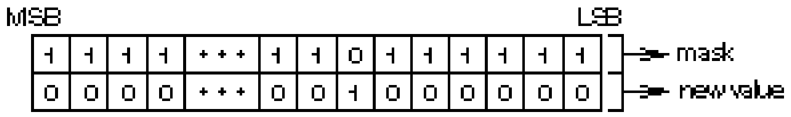
Bit (0) - bit (2)	Material damping 00 : material damping 0 01 : material damping 1 02 : material damping 2 03 : material damping 3
Bit (3)	Lighting mode, part 1 0 : fog off (no fog) 1 : fog on (fog)
Bit (4)	Lighting mode, part 2 0 : material on (material) 1 : material off (no material)
Bit (5)	Lighting mode, part 3 0 : use lighting mode 1 : use default lighting mode
Bit (6)	Light source 0 : light-source calculation off 1 : forced light-source calculation on
bit (7)	NearZ overflow handling 0 : z overflow clip 1 : z overflow not clip
Bit (8)	Back clipping status 0 : valid 1 : invalid
Bit (9) - bit (27)	Reserved (initialized at 0)
Bit (28) - bit (29)	Semi-transparency rate 00 : 50% back + 50% polygon 01 : 100% back + 100% polygon 10 : 100% back - 100% polygon 11 : 50% back + 25% polygon
Bit (30)	Semi-transparency rate 0 : OFF 1 : ON
Bit (31)	Display 0 : display 1 : no display

For example, to switch forced light-source calculation ON, the packet data bits should be set as shown in Figure 2-53.

Bit (6) of the first word is given the value 0, showing that the light source is to be changed. The other bits are given the value 1, showing that they are not going to be changed. Accordingly, the first word is 0xffbf.

Bit (6) of the second word is given the value 1 to indicate that forced light-source calculation is ON, and the other bits, which correspond to items which are not going to be changed, are given the default value 0. The second word is therefore 0x0040.

Figure 2–53: packet data when forced light-source calculation is switched ON



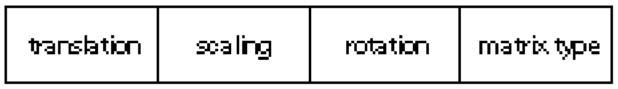
MSB													LSB	
1	1	1	1	...	1	1	0	1	1	1	1	1	1	mask
0	0	0	0	...	0	0	1	0	0	0	0	0	0	new value

Packet Data-Coordinate (RST)

When packet type is 0001, data that sets the coordinates of the GsDOBJ structure is stored in packet data.

In this case the flag will have the following meaning.

Figure 2–54: Flag when Coordinate (RST)



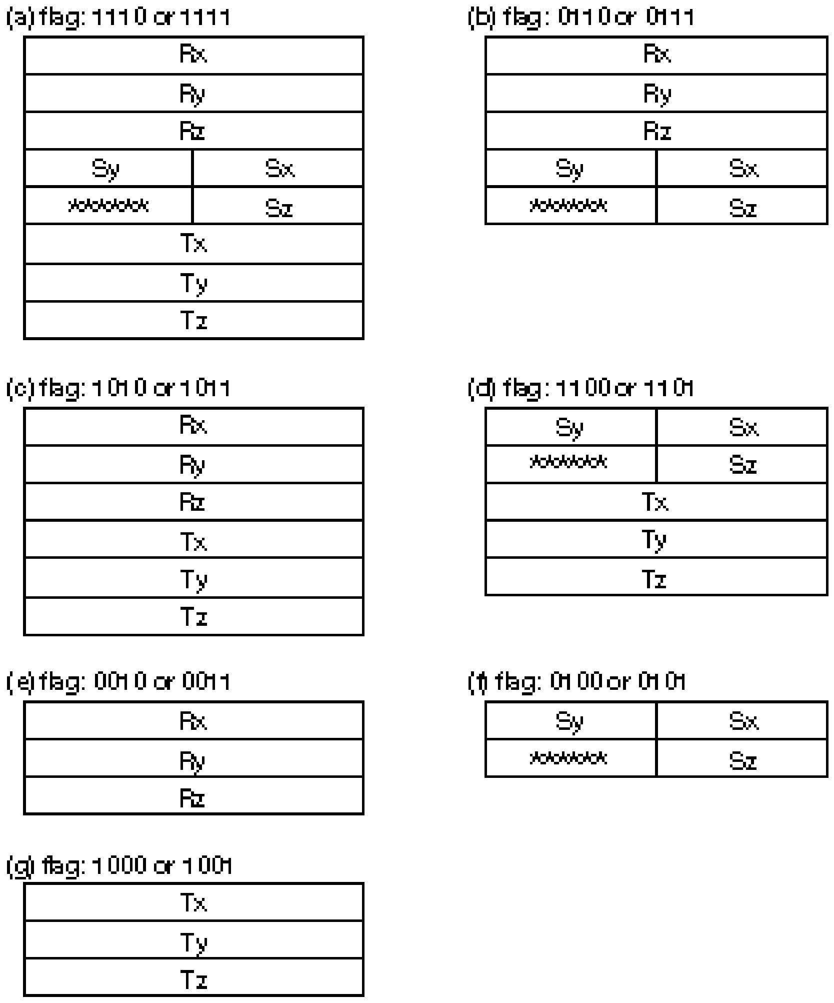
translation	scaling	rotation	matrix type

Matrix type:	RST matrix type 0: Absolute value 1: Differential matrix from preceding frame
Rotation:	Rotation (R) flag 0: None 1: Has
Scaling	Screening (S) flag 0: None 1: Has
Translation	Parallel movement (T) flag 0: None 1: Has

The configuration of packet data will differ according to the values of the flag rotation bit, the scaling bit, and the translation bit as per Figure 2-54.

In Figure 2-55, Rx, Ry and Rz indicate one degree as 4096, with a fixed point decimal value (1, 19, 12) that indicate the X axis component, the Y axis component, and the Z axis component of the angle of rotation. In the same way, Sx, Sy and Sz indicate the X axis component, the Y axis component, and the Z axis component of the scaling as a fixed point decimal (1, 3, 12), while Tx, Ty and Tz respectively indicate the X axis component, the Y axis component, and the Z axis component of the translation as an integer (1, 31, 0) that signals 32 bits.

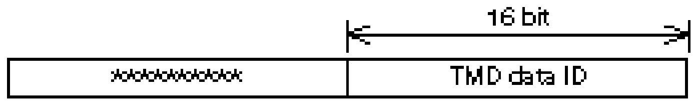
Figure 2–55: Packet Data Configuration when Coordinate (RST)



Packet Data-TMD Data ID

When packet type is 0010, the modeling data ID (TMD data) of the real object is stored in the packet data (See Figure 2-56). The TMD data ID is composed of 2 bytes. In this case no flag is used.

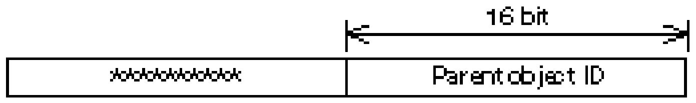
Figure 2–56: Packet Data Configuration when TMD Data ID



Packet Data- Parent Object ID

When packet type is 0011, the parent object ID of the object specified is stored in packet data (see Figure 2-57). The parent object ID is composed of 2 bytes. In this case no flag is used.

Figure 2–57: Packet Data Configuration when Parent Object



Packet Data - MATRIX Value

When the packet type is 0100, the data which designates coord members of the GsCOORDINATE2 structure to which GsDOBJ2 structure points is stored in packet data. In this case a flag is not used.

Figure 2–58: Packet Data Configuration when Mat rix Value

R01	R00
R10	R02
R11	R11
R21	R20
************	R22
Tx	
Ty	
Tz	

Packet Data-TMD Data Body

When packet type is 0101, TMD data is stored. This is not presently supported.

Packet Data-Light Source

When packet type is 0110, the data that designates light source is stored in packet data. When this is the case, the object ID is separate from the normal object ID and becomes the light source ID. Flags have the following meanings:

Figure 2–59: Flag when Light Source Packet

**********	Color	Direction	Data type

Data type:	Data type 0: Absolute value 1: Difference from preceding frame
Direction:	Direction flag 0: None 1: Has
Color:	Color flag 0: None 1: Has

The configuration of packet data will differ according to the value of the flag direction bit and the color bit.

Figure 2–60: Packet Data when Light Source Packet

(a) flag: 0110 or 0111

X			
Y			
Z			
**	B	G	R

(b) flag: 0100 or 0101

X
Y
Z

(c) flag: 0010 or 0011

**	B	G	R

Packet Data-Camera

When packet type is 0111, data which designates viewpoint location information is stored in the packet. When this is the case, the object ID is separate from the normal object ID and becomes the camera ID. Flags have the meaning indicated in Figure 2-61. Please be careful to note that the meaning of other bits will change depending on the type bit.

Figure 2–61: Flag for Camera

(a) camera type: 0

z angle	position & reference	data type	camera type = 0

(a) camera type: 1

translation	rotation	data type	camera type = 1

When camera type bit is 0 other bits are:

Data type:	Data type 0: Absolute value 1: Difference from preceding frame
Position & reference	Position and reference position flag 0: None 1: Has
z angle	Reference angle flag from level 0: None 1: Has

When camera type bit is 1 other bits are:

Data type:	Data type 0: Absolute value 1: Difference from preceding frame
Rotation:	Rotation (R) flag 0: None 1: Has
Translation:	Horizontal movement (T) flag 0: None 1: Has

The structure of packet data differs according to the flag content, as shown in Figs.2-62 and 2-63.

Figure 2–62: Composition of packet data with camera (part 1)

(a) flag: 1100 or 1110

Tx
Ty
Tz
TRx
TRy
TRz
Z

(b) flag: 0100 or 0110

Tx
Ty
Tz
TRx
TRy
TRz

(c) flag: 1000 or 1010

Z

Tx, Ty, Tz:	camera position
TRx, TRy, TRz:	camera close-up position

Figure 2–63: Composition of packet data with camera (part 2)

(a) flag: 0111

Rx
Ry
Rz
Tx
Ty
Tz

(b) flag: 0011

Rx
Ry
Rz

(c) flag: 0101

Tx
Ty
Tz

Rx, Ry, Rz:	Rotation
Tx, Ty, Tz:	Translation

Packet Data-Object Control

If the packet type is 1000, object control is not set. In this case, there is no packet data. The flag has the meanings shown below.

Figure 2–64: The meanings and values of the flag when object control is set

0	create
1	kill
0010-1111	system reserved

Packet Data-Extended Commands

If the packet type is 1110, it shows the extended commands.

Packet Data-Special Commands

If the packet type is 1111, animation control is performed. Details of these special commands have not yet been finalized.

Chapter 3: 2D Graphics

TIM: Screen Image Data

The TIM file covers standard images handled by the PlayStation unit, and can be transferred directly to its VRAM. It can be used commonly as sprite patterns and 3D texture mapping materials.

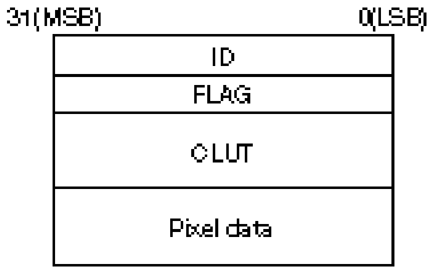
The following are the image data modes (color counts) handled by the PlayStation unit.

- 4-bit CLUT
- 8-bit CLUT
- 16-bit Direct color
- 24-bit Direct color

The VRAM supported by the PlayStation unit is based on 16 bits. Thus, only 16- and 24-bit data can be transferred directly to the frame buffer for display. Use as sprite pattern or polygon texture mapping data allows the selection of any of 4-bit, 8-bit and 16-bit modes.

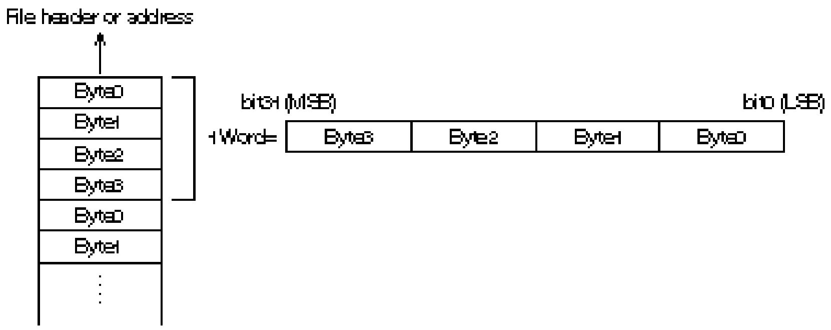
TIM files have a file header (ID) at the top and consist of several different blocks.

Figure 3–1: TIM File Format



Each data item is a string of 32-bit binary data. The data is Little Endian, so in an item of data containing several bytes, the bottom byte comes first (holds the lowest address), as shown in Figure 3-2.

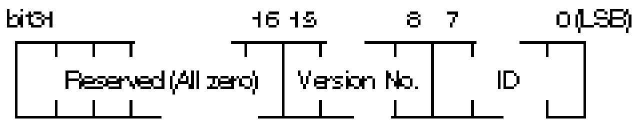
Figure 3–2: The order of bytes in a file



ID

The file ID is composed of one word, having the following bit configuration.

Figure 3–3: Structure of TIM File Header



Bits 0 – 7: ID value is 0x10

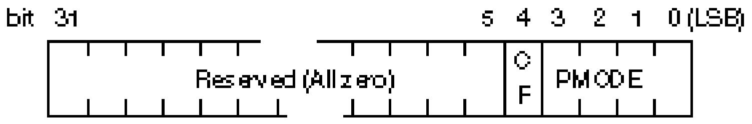
Bits 8 – 15:	Version number. Value is 0x00

Flag

Flags are 32-bit data containing information concerning the file structure. The bit configuration is as in Figure 3-4.

When a single TIM data file contains numerous sprites and texture data, the value of PMODE is 4 (mixed), since data of multiple types is intermingled.

Figure 3–4: Flag Word



Bits 0 -3 (PMODE):	Pixel mode (Bit length) 0: 4-bit CLUT 1: 8-bit CLUT 2: 15-bit direct 3: 24-bit direct 4: Mixed
Bit 4 (CF):	Whether there is a CLUT or not 0: No CLUT section 1: Has CLUT section
Other:	Reserved

CLUT

The CF flag in the FLAG block specifies whether or not the TIM file has a CLUT block. A CLUT is a color palette, and is used by image data in 4-bit and 8-bit mode.

As shown in Figure 3-5, the number of bytes in the CLUT (bnum) is at the top of the CLUT block. This is followed by information on its location in the frame buffer, image size, and the substance of the data.

Figure 3–5: CLUT

bit 31(MSB)	bit 0(LSB)
bnum	
DY	DX
H	W
CLUT 1	CLUT 0
:	:
CLUT n	CLUT n-1

bnum	Data length of CLUT block. Units: bytes. Includes the 4 bytes of bnum.
DX	x coordinate in frame buffer.
DY	y coordinate in frame buffer.
H	Size of data in vertical direction.
W	Size of data in horizontal direction.
CLUT 1~n	CLUT entry (16 bits per entry).

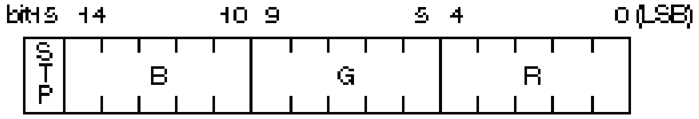
In 4-bit mode, one CLUT consists of 16 CLUT entries. In 8-bit mode, one CLUT consists of 256 CLUT entries.

In the PlayStation system, CLUTs are located in the frame buffer, so the CLUT block of a TIM file is handled as a rectangular frame buffer image. In other words, one CLUT entry is equivalent to one pixel in the frame buffer. In 4-bit mode, one CLUT is handled as an item of rectangular image data with a height of 1 and a width of 16; in 8-bit mode, it is handled as an item of rectangular image data with a height of 1 and a width of 256.

One TIM file can hold several CLUTs. In this case, the area in which several CLUTs are combined is placed in the CLUT block as a single item of image data.

The structure of a CLUT entry (= one color) is as follows:

Figure 3–6: A CLUT entry



STP	Transparency control bit
R	Red component (5 bits)
G	Green component (5 bits)
B	Blue component (5 bits)

The transparency control bit (STP) is valid when data is used as Sprite data or texture data. It controls whether or not the relevant pixel, in the Sprite or polygon to be drawn, is transparent. If STP is 1, the pixel is a semitransparent color, and if STP is other than 1, the pixel is a non-transparent color.

R, G and B bits control the color components. If they all have the value 0, and STP is also 0, the pixel will be a transparent color. If not, it will be a normal color (non-transparent).

These relationships can be represented in a table as follows:

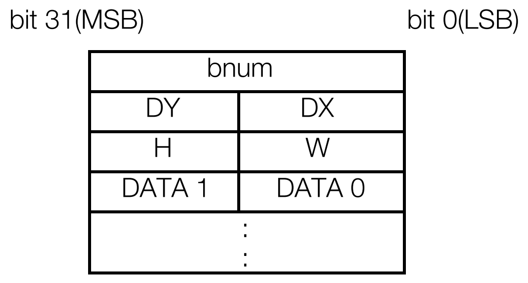
Table 3–1: STP Bit Function in Combination with R, G, B Data

STP/R,G,B	Transucent processing on	Translucent processing off
0,0,0,0	Transparent	Transparent
0,X,X,X	Not transparent	Not transparent
1,X,X,X	Semi-transparent	Not transparent
1,0,0,0	Non-transparent black	Non-transparent black

Pixel Data

Pixel data is the substance of the image data. The frame buffer of the PlayStation system has a 16-bit structure, so image data is broken up into 16-bit units. The structure of the pixel data block is as shown below.

Figure 3–7: Pixel data



DATA n	DATA n-1

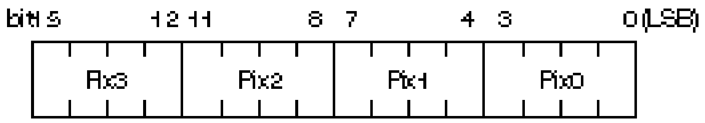
bnum	Data length of pixel data. Units: bytes.Includes the 4 bytes of bnum.
DX	Frame buffer x coordinate
DY	Frame buffer y coordinate
H	Size of data in vertical direction
W	Size of data in horizontal direction (in 16-bit units)
DATA 1~n	Frame buffer data (16 bits)

The structure of one item of frame buffer data (16 bits) varies according to the image data mode. The structure for each mode is shown in Figure 3-8.

Care is needed when handling the size of the pixel data within the TIM data. The W value (horizontal width) in Figure 3-7 is in 16-pixel units, so in 4-bit or 8-bit mode it will be, respectively, 1/4 or 1/2 of the actual image size. Accordingly, the horizontal width of an image size in 4-bit mode has to be a multiple of 4, and an image size in 8-bit mode has to be an even number.

Figure 3–8: Frame buffer data (pixel data)

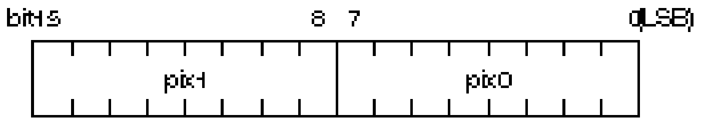
(a) In 4-bit mode



pix 0-3 pixel value (CLUT No.)

The order on the screen is pix0, 1, 2, 3, starting from the left.

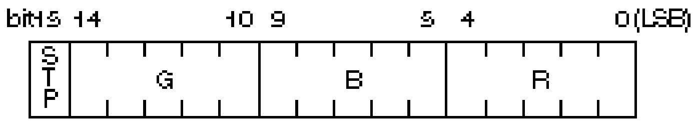
(b) In 8-bit mode



pix 0-1 pixel value (CLUT No.)

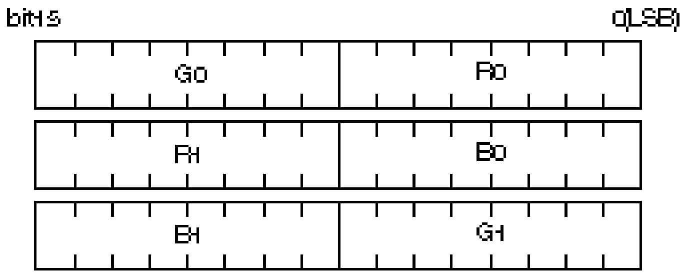
The order on the screen is pix0, 1, starting from the left.

(c) In 16-bit mode



STP	transparency control bit (see CLUT)
R	Red component (5 bits)
G	Green component (5 bits)
B	Blue component (5 bits)

(d) In 24-bit mode:



R0, R1	Red component (8 bits)
G0, G1	Green component (8 bits)
B0, B1	Blue component (8 bits)

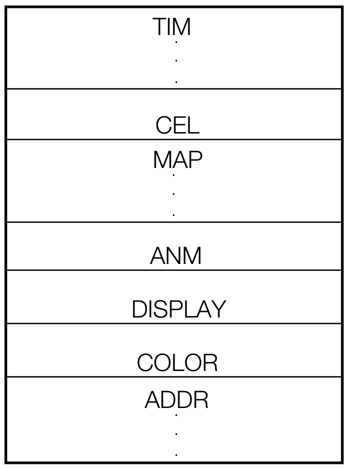
In 24-bit mode, 3 items of 16-bit data correspond to 2 pixels' worth of data. (R0, G0, B0) indicate the pixels on the left, and (R1, R2, B1) indicate the pixels on the right.

SDF: Sprite Editor Project File

The SDF file stores settings and file groups created and edited by the PlayStation sprite editor and enables all linked files to be loaded together.

The SDF file is an ASCII text file composed of seven blocks of information, as shown in figure 3-9. Each block is designated by a unique keyword that begins each line within the block. For some blocks, a bank number, ranging from 0-3, is appended to the block keyword. Some blocks use only one bank of data while others use four. Following the keyword and bank number is a list of parameters. For those blocks with four banks of data, each bank must be specified, even if no parameters are given.

Figure 3–9: SDF File Structure



The block is composed of lines assigned key word values.

Sample SDF File Contents

```
TIM0 file0.tim
TIM1 file1.pxl file1.clt
TIM2
TIM3
CEL0 file2.cel
MAP0 file3.bgd
MAP1 file4.bgd
MAP2
MAP3
```

```
ANM0 file5.anm
DISPLAY 1
COLOR0
ADDR0 768 0 0 480 16
ADDR1 768 256 0 496 16
ADDR2 512 0 256 480 16
ADDR3 512 256 256 496 16
```

TIM

The keyword of the TIM block is “TIM?” where “?” is a bank number from 0 to 3. All four banks must be specified. Following the keyword is the name of a TIM file, or the name of separate PXL and CLT files. If no data is required for a bank, the remainder of the line is left blank. For example:

```
TIM0 file0.tim
TIM1 file1.pxl file1.clt
TIM2
TIM3
```

Note

The key word of a bank not used must not be omitted, but assigned an item having no value.

CEL

The keyword of the CEL block is “CEL0”. There is only one bank of data. Following the keyword is the name of a CEL file. If no data is required for the CEL block, the remainder of the line following is left blank. For example:

```
CEL0  file0.tim
```

if no data is required:

```
CEL0
```

MAP

The keyword of the MAP block is “MAP?” where “?” is a bank number from 0 to 3. All four banks must be specified. Following the keyword is a filename. If no data is required for a bank, the remainder of the line is left blank. For example:

```
MAP0 file3.bgd
MAP1 file4.bgd
MAP2
MAP3
```

Note

The key word of a bank not used must not be omitted, but assigned an item having no value.

ANM

The keyword of the ANM block is “ANM0”. There is only one bank of data. Following the keyword is the name of a ANM file. If no data is required for the ANM block, the remainder of the line following is left blank. For example:

```
ANM0 file5.anm
```

if no data is required:

```
ANM0
```

DISPLAY

A DISPLAY block specifies and Artist Boards screen mode. The keyword for this block is DISPLAY. The argument value depends on the desired artist board screen mode as shown in Table 3-2.

Table 3–2: Display

Image mode	Value
256x240	0
320x240	1
512x240	2
640x240	3
256x480	4
320x480	5
512x480	6
640x480	7

For example:

```
DISPLAY 1
```

This would set the artist board to the 320x240 resolution mode.

COLOR

The COLOR block specifies the color mode. The keyword is COLOR. The value depends on the desired color mode, as shown in Table 3-3.

Table 3–3: Color

Color Mode	Value
16	0
256	1

For example:

```
COLOR 1
```

This would set the artist board to the 256 color mode.

ADDR

An ADDR block specifies the coordinates of images specified in the corresponding TIM bank, as well as the palette coordinates, and the number of color sets. The keyword of the ADDR block is “ADDR?” where “?” is a bank number from 0 to 3. All four banks must be specified. Following the keyword is a parameter list. The parameters for the ADDR block are as follows:

ADDR? X Y CX CY N

X:	X coordinate of TIM image
Y:	Y coordinate of TIM image
CX:	X coordinate of palette
CY:	Y coordinate of palette
N:	Number of color sets

All parameters must be specified. For example:

```
ADDR0 768 0 0 480 16
ADDR1 768 256 0 496 16
```

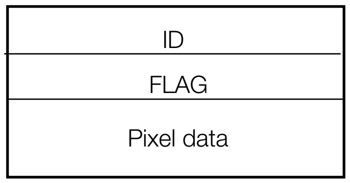
```
ADDR2 512 0 256 480 16
ADDR3 512 256 256 496 16
```

PXL: Pixel Image Data

The PXL format stores 4-bit or 8-bit indexed-color graphics images created and edited by the PlayStation sprite editor. Palette information is not included (this is contained within a CLT file that is used together with the PXL file).

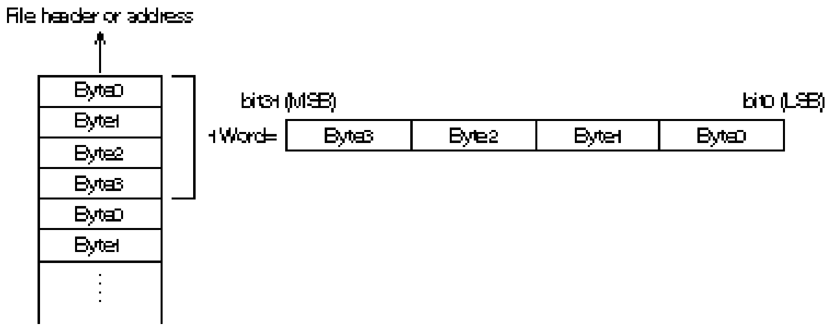
The PXL format has a simple header containing an ID field and a FLAG field. After that comes raw pixel data. All values are stored in little-endian format. Bytes for 16-bit or 32-bit values are stored in ascending order (i.e. a 32-bit value would be stored byte 0, byte 1, byte 2, then byte 3).

Figure 3–10: PXL File Structure



Data is of the 32-bit binary format. Because of LittleEndian, bytes are arranged in ascending order. (see Figure 3-11).

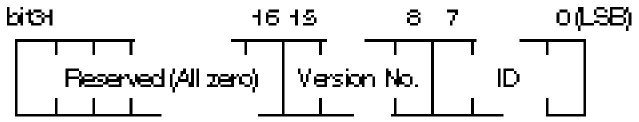
Figure 3–11: Byte Order in File



ID

The ID field is a 32-bit value with the following bit definition:

Figure 3–12: Structure of PXL File Header

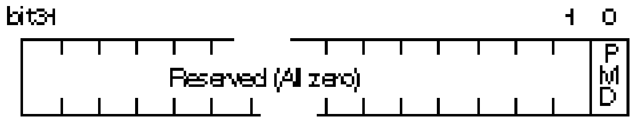


Bit 0 – 7:	ID value is 0x11
Bit 8 – 15:	Version number. Value is 0x00

FLAG

The FLAG field is a 32-bit value containing information about the pixel data format. It has the following bit definition:

Figure 3–13: FLAG Bit Configuration

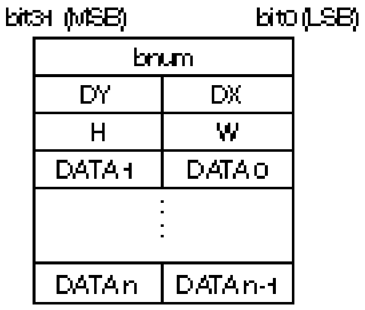


Bit 0: (PMODE):	Pixel mode (Bit length) 0: 4-bit CLUT 1: 8-bit CLUT

Pixel Data

The pixel data section contains the actual image information. It includes a short header as shown below, followed by the raw image data that is ready to be copied to the PlayStation's VRAM.

Figure 3–14: Configuration of Pixel Data Section

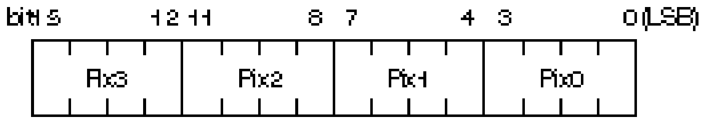


bnum	Length of pixel data in bytes, including the 48 bytes of *bnum*
DX	Frame buffer x coordinate
DY	Frame buffer y coordinate
H	Size of data in vertical direction
W	Size of data in horizontal direction
DATA	VRAM (16 bits)

The configuration of a piece of VRAM data (16 bits) depends on the mode. The following gives the configuration in each mode.

Figure 3–15: VRAM Data (Pixel Data)

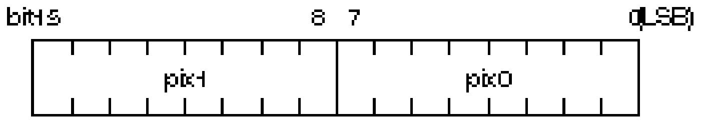
a) In 4-bit mode



pix 0-3 pixel value (CLUT No.)

The order on the screen is pix0, 1, 2, 3, starting from the left.

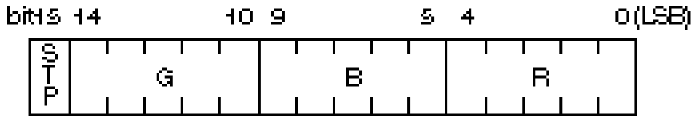
b) In 8-bit mode



pix 0-1 pixel value (CLUT No.)

The order on the screen is pix0, 1, starting from the left.

(c) In 16-bit mode



STP	Transparency control bit (see CLUT)
R	Red component (5 bits)
G	Green component (5 bits)
B	Blue component (5 bits)

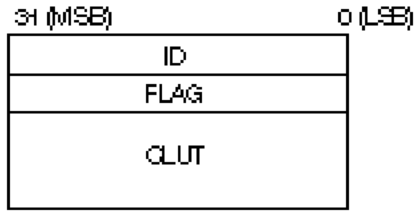
The coordinate system for the VRAM is based on 16 bits per pixel. Thus, note coordinate/size values in TIM data. In the X axis direction, a value of 2/4 is H in the 4-bit mode, and a value of 1/2 is H in the 8-bit mode. This means that, in the 4-bit mode, the image size must be a multiple of 4 and, in the 8-bit mode, the image size must be even.

CLT: Palette Data

The CLT file saves 8-bit or 4-bit palette data edited by the PlayStation sprite editor.

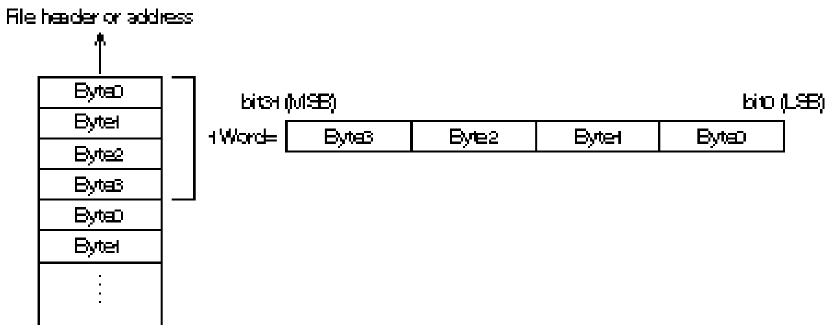
The CLT format has a simple header containing an ID field and a FLAG field. After that comes raw pixel data. All values are stored in little-endian format. Bytes for 16-bit or 32-bit values are stored in ascending order (i.e. a 32-bit value would be stored byte 0, byte 1, byte 2, then byte 3).

Figure 3–16: CLT File Structure



Data is of the 32-bit binary format. Because of LittleEndian, bytes are arranged in ascending order. (see Figure 3-17).

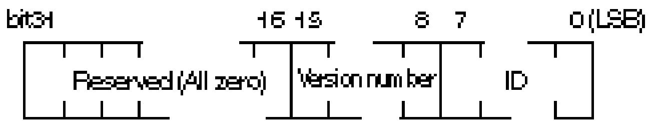
Figure 3–17: Byte Order in File



ID

The ID field is a 32-bit value with the following definition:

Figure 3–18: Structure of CLT File Header

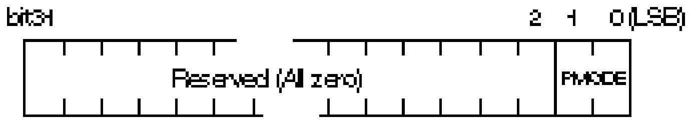


Bit 0–7:	ID value is 0x12
Bit 8–15:	Version number. Value is 0x00

FLAG

The FLAG field is a 32-bit value containing information about the pixel data format. It has the following bit definition:

Figure 3–19: FLAG Bit Configuration



Bit 0-1:	PMODE 0x2

CLUT Section

The CLUT section begins with data on its byte count (bnum), followed by inner-VRAM positional information, index size and the main data body.

Figure 3–20: Structure of CLUT Section

bit 31(MSB)	bit 0(LSB)
bnum	
DY	DX
H	W
CLUT 1	CLUT 0
:	
CLUT n	CLUT n-1

bnum	Data length of CLUT block.
DX	x coordinate in frame buffer.
DY	y coordinate in frame buffer.
H	Size of data in vertical direction.
W	Size of data in horizontal direction.
CLUT	VRAM data (16 bits per entry).

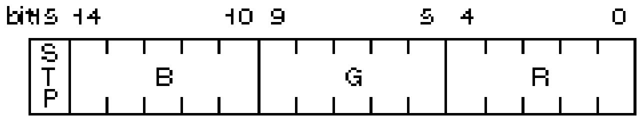
One CLUT set is composed of 16 CLUT entries in the 4-bit mode and of 256 CLUT entries in the 8-bit mode. (However, one file is composed of 16 sets of CLT data output from the sprite editor in the 4-bit mode.)

As CLUT is located on the VRAM, the PlayStation system handles the CLUT section in a TIM file as a rectangular VRAM image. This means that one CLUT entry is equivalent to one pixel in the VRAM. Thus, one CLUT set is handled as rectangular image data having a height of 1 and a width of 16 in the 4-bit mode and a height of 1 and a width of 256 in the 8-bit mode. (CLUT data output from the sprite editor is a rectangular image with a height of 16 and a width of 16 in the 4-bit mode.)

One TIM file can contain two or more CLUT sets. The area composed of two or more CLUT sets is considered to be a piece of image data and written in the CLUT section.

A CLUT entry, which expresses one color, has the following configuration.

Figure 3–21: CLUT Entry



STP	Transparency control bit (see CLUT)
R	Red component (5 bits)
G	Green component (5 bits)
B	Blue component (5 bits)

The transparency control bit is applicable to sprite and texture data. If all of R, G, B and STP are zero, the color is regarded as being transparent. If not, the color is considered to be opaque.

For semitransparency processing, if the STP value is 1, the color is considered to be semitransparent. If not, the color is regarded as being opaque. (Only in all zeros, the color is considered to be transparent.)

Table 3-4-1: Role of STP Bit

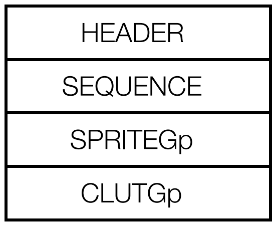
STP/R,G,B	Semi-transparency processing on	Semi-transparency processing off
0,0,0,0	Transparent	Transparent
0,X,X,X	Not transparent	Not transparent
1,X,X,X	Semi-transparent	Not transparent

ANM: Animation Information

The ANM format contains information that specifies image data animation. An ANM file is typically used in conjunction with a TIM file, which contains the actual screen information.

The ANM file has a header at the top, and is divided into four blocks.

Figure 3–22: ANM file format



HEADER

This is the file header. Its structure is shown below.

Figure 3–23: HEADER

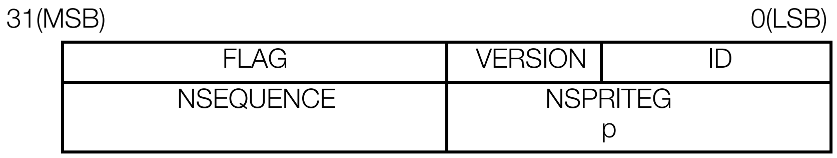
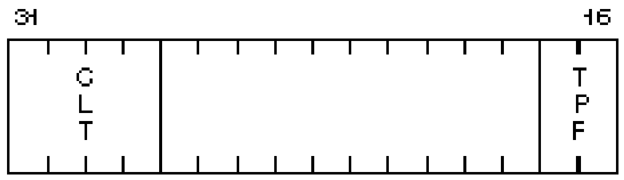


Table 3–4

First LONG word		
Bits 32-16	FLAG	See Figure 3-23
Bits 15-8	VERSION	0x01
Bits 7-0	ID	0x21

Second LONG word		
Bits 32-16	NSEQUENCE	Number of sequences
Bits 15-0	NSPRITEGp	Number of sprite groups

Figure 3–24: FLAG

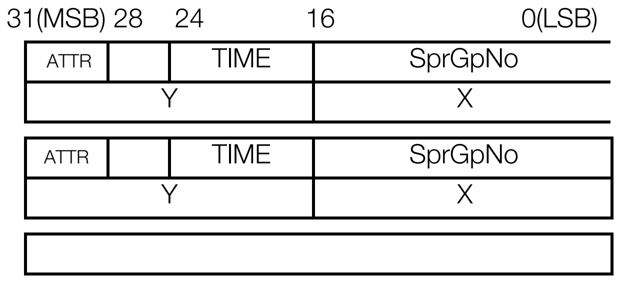


CLT	Number of CLUTs for color animation
TPF	Texture pattern pixel depth 00:4-bit 01:8-bit

SEQUENCE

Sequence data provides a set of coordinates of the hot spots in the frames, display time and sprite group numbers.

Figure 3–25: Sequence

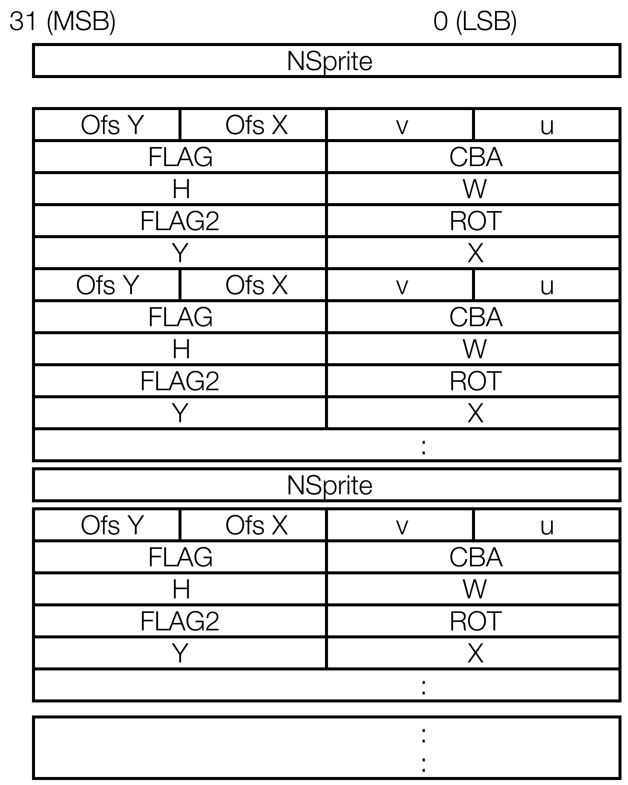


TIME	Display time (number of repetitions)
SprGpNo	Number of Sprite group to be displayed, from among the sprit group data
X	X coordinate of hot spot
Y	Y coordinate of hot spot
ATTR	Attribute

SPRITEGp

Sprite group data is a set of sprite groups, describing where a sprite is to be displayed in a frame.

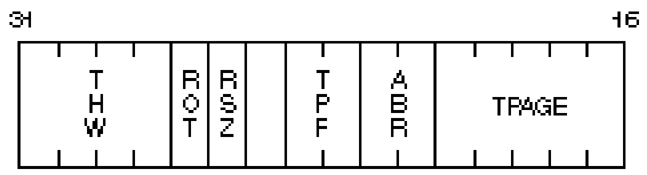
Figure 3–26: SPRITEGp



NSprite	Number of Sprites in one Sprite frame
FLAG	See Figure 3-27
v	Vertical offset from base address of texture page
u	Horizontal offset from base address of texture page
Ofs Y	Vertical offset from hotspot within frame
Ofs X	Horizontal offset from hotspot within frame
CBA	See Figure 3-28
H	Width of texture of optional size
W	Height of texture of optional size
ROT	Angle of rotation
FLAG2	See Figure 3-29
Y, X	Scaling factor (specified as a fixed-point number)

FLAG has the following bit configuration.

Figure 3–27: FLAG

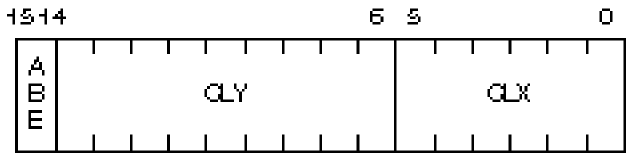


THW	The size of the rectangular area of the Sprite, divided by 8 (if it cannot be divided by 8, this bit is set to 0x0 and the actual size is specified using H and W, described earlier.)

ROT	Rotation status 0: Not rotated 1: Rotated
RSZ	Scaling status 0: Not scaled 1: Scaled
TPF	Pixel depth of texture pattern 00: 4-bit CLUT 01: 8-bit CLUT 10: 16-bit
ABR	Semi-transparency rate. 00: 50%Back + 50%Sprite 01: 100%Back + 100%Sprite 10: 100%Back – 100%Sprite 11: 100%Back + 25%Sprite
TPAGE	Texture page number (0-31)

CBA has the following bit configuration.

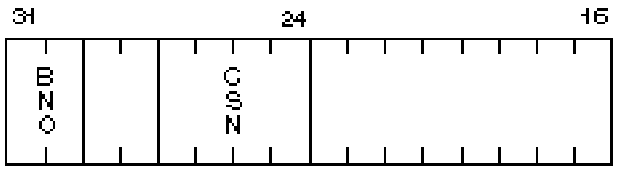
Figure 3–28: CBA



ABE	0 : Semi-transparency processing OFF 1 : Semi-transparency processing ON
CLY	Y coordinate of beginning of CLUT (9 bits)
CLX	X coordinate of beginning of CLUT (6 bits)

FLAG2 has the following bit configuration.

Figure 3–29: FLAG2



BNO	TIM bank number (Sprite editor)
CSN	Color set number (Sprite editor)

CLUTGp

CLUT Gp is a group of CLUTs used for color animations. The number of CLUTs is specified by the CLT parameter of the FLAG in the HEADER block.

Figure 3–30: CLUTGp

31 (MSB)	0 (LSB)
bnum	
DY	DX
H	W
CLUT 1	CLUT 0
:	
CLUT n	CLUT n-1

bnum	
DY	DX
H	W
CLUT1	CLUT 0
:	
CLUT n	CLUT n-1

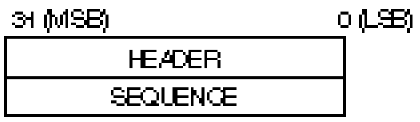
:

bnum	Data length of CLUT (in bytes)
DX	x coordinate in frame buffer
DY	y coordinate in frame buffer
W	Horizontal size of data
H	Vertical size of data
CLUT 0~n	CLUT entries (16 bits per entry)

TSQ: Animation Time Sequence

TSQ is a binary file that stores time sequence data for sprite animations created and edited by the PlayStation sprite editor. It has a short header composed of two blocks.

Figure 3–31: SEQ Data Structure

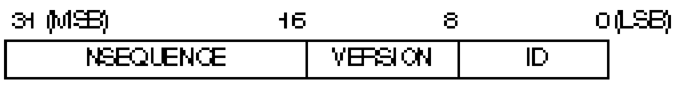


31 (MSB) 0 (LSB)
HEADER
SEQUENCE

HEADER

The file header has the following configuration.

Figure 3–32: HEADER



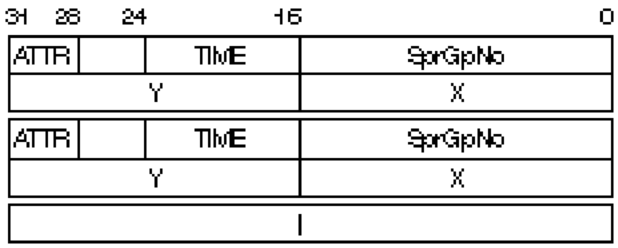
31 (MSB) 16	8	0 (LSB)
NSEQUENCE	VERSION	ID

ID:	0x24
VERSION:	0x01
NSEQUENCE	Sequence data count

SEQUENCE

Sequence data is a set of coordinates of the hot spots in the frames, display time, and sprite group numbers.

Figure 3–33: SEQUENCE



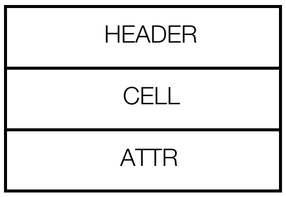
TIME	Display time
SprGpNo	Number of Sprite group to be displayed
X	X coordinate of hot spot
Y	Y coordinate of hot spot
ATTR	Attribute

CEL: Cell Data

CEL format stipulates the pointer table, in the VRAM, of the CELLs forming constituents of the BG surface.

A CEL file has a header at the top, and is divided into three blocks.

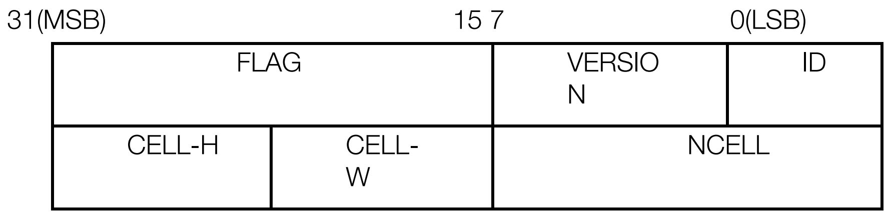
Figure –3–34: CEL file format



HEADER

The file header has the following configuration.

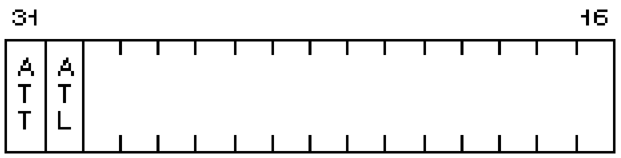
Figure 3–35: HEADER



FLAG	Described later
ID	0x22
VERSION	0x04
NCELL	Number of cell data items (units: cells)
CELL-H	Size of cell display window height (units: cells) (used locally in sprite editor)
CELL-W	Size of cell display window width (units: cells) (used locally in sprite editor)

FLAG has the following bit configuration.

Figure 3–36: FLAG



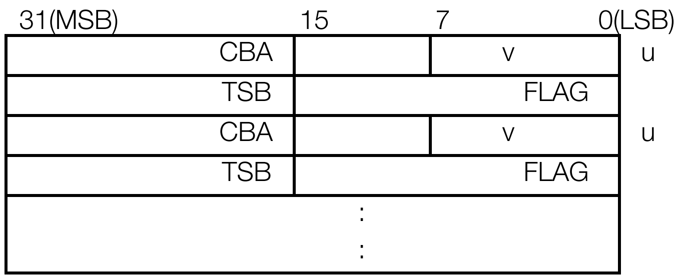
ATT Indicates whether an ATTR block is included in this file
0: ATTR is not included
1: ATTR is included

ATL Length of ATTR data
0: 8-bit
1: 16-bit

CELL

Cell data provides a table of VRAM pointers to cells constituting BG. Four bytes form one cell.

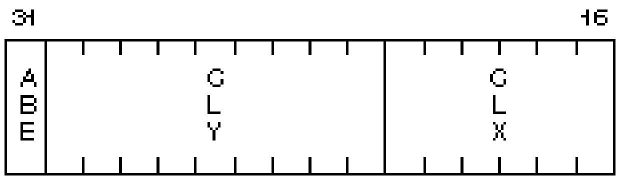
Figure 3–37: CELL Data Section



v	Offset in vertical direction from base address of texture page (8 bits)
u	Offset in horizontal direction from base address of texture page (8 bits)
CBA	Described later
TSB	Described later
FLAG	Described later

CBA has the following bit configuration.

Figure 3–38: CBA



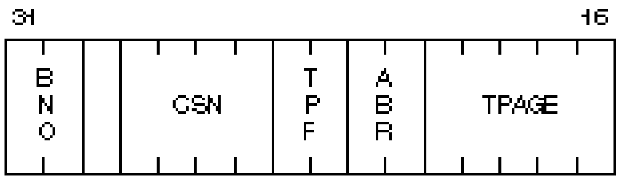
ABE 0 : Semi-transparency processing OFF
1 : Semi-transparency processing ON

CLY Y coordinate of beginning of CLUT (9 bits)

CLX X coordinate of beginning of CLUT (6 bits)

TSB has the following bit configuration.

Figure 3–39: TSB



BNO	TIM bank no.
CSN	color set no.

TPF Pixel depth of texture pattern

00 :	4-bit CLUT
01 :	8-bit CLUT
10 :	16-bit Direct

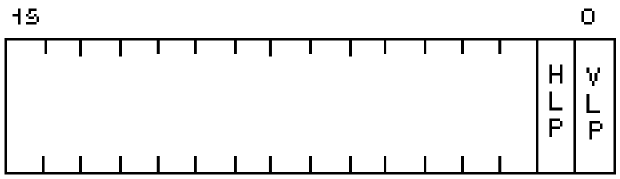
ABR Translucence rate (F=foreground, B=background)

00 :	0.50xF + 0.50xB
01 :	1.00xF + 1.00xB
10 :	-1.00xF + 1.00xB
11 :	0.25xF + 1.00xB

TPAGE Texture Page number (0-31)

FLAG has the following bit configuration.

Figure 3–40: FLAG



HLP	Horizontal reversal information
VLP	Vertical reversal information

ATTR

Expresses attribute data. Attribute data is additional information concerning the cell and is arranged in the same order as CEL.

There are two types of attribute data, 8 bit length data and 16 bit length data and each is shown below. Data length is indicated in the Header section, in the ATL bit in the FLAG half word.

Figure 3–41: ATTR Format (8 Bit)

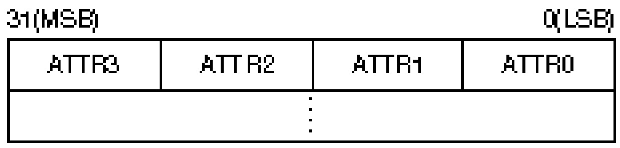


Figure 3–42: ATTR Format (16 Bit)

31 (MSB)	0 (LSB)
ATTR1	ATTR0
ATTR3	ATTR2
⋮	

BGD: BG Map Data

The BGD file provides data constituting the BG plane used in the 2D system. BG refers to any row of rectangular pixel data. The BGD file is used along with the TIM and CEL files having the same name. Actual pixel images are carried by the TIM file.

The BGD file has a header at the top, and is divided into three blocks. The ATTR block may be omitted.

Figure 3–43: BG file format

HEADER
MAP
ATTR

HEADER

This is the file header. Its structure is as follows:

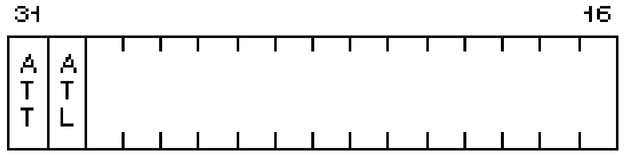
Figure 3–44: HEADER

31 (MSB)			0 (LSB)
FLAG		VERSION	ID
CELLH	CELLW	MAPH	MAPW

ID	0x23
VERSION	0X00
FLAG	See Figure 3-45
MAPH	Vertical size of BG map data (in cell units)
MAPW	Horizontal size of cell BG map data (in cell units)
CELLH	Vertical size of cell data (in pixel units)
CELLW	Horizontal size of cell data (in pixel units)

The structure of the FLAG in Figure 3-44 is as follows:

Figure 3–45: FLAG



ATT	Indicates whether an ATTR block is included in this file 0: ATTR is not included 1: ATTR is included
ATL	Length of ATTR data 0: 8-bit 1: 16-bit

MAP Section

A map is considered as a set of the cells of MAPH x MAPW (a matrix of the vertical and horizontal size) which describes the order of arrangement of these cells. For example the arrangement of the cells of an 8 x 8 map would be as follows:

Figure 3–46: Cell Arrangement in MAP (when 8 x 8)

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

The Map section is an aggregate of cell numbers arranged in numerical order in a form like that in Figure 3-46. Cell number is a number which indicates the number of the cell in the CEL file.

Figure 3–47: MAP

31(MSB)	0(LSB)
CELL No.(1)	CELL No.(2)
⋮	

ATTR Section

Indicates attribute data. Attribute data is additional information concerning the MAP and is arranged in the same order as MAP.

There are two types of attribute data, 8-bit data and 16-bit data and each is shown below. Data length is indicated in the Header section, in the ATL bit in the FLAG.

Figure 3–48: ATTR (8 bit)

31(MSB)			0(LSB)
ATTR3	ATTR2	ATTR1	ATTR0
⋮			

Figure 3–49: ATTR (16 bit)

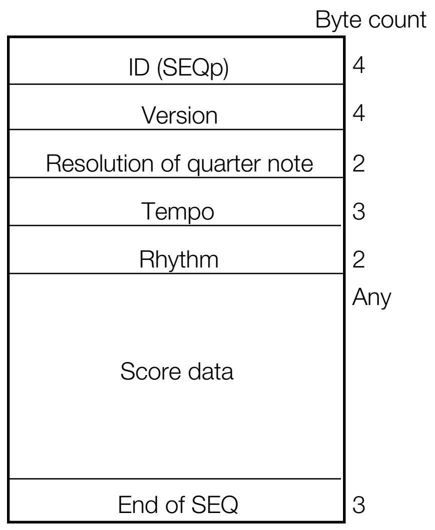
31(MSB)	0(LSB)
ATTR1	ATTR0
ATTR3	ATTR2
⋮	

Chapter 4:
Sound

SEQ: PS Sequence Data

SEQ is the PlayStation sequence data format. The typical extension in DOS is ".SEQ".

Figure 4–1: SEQ Format



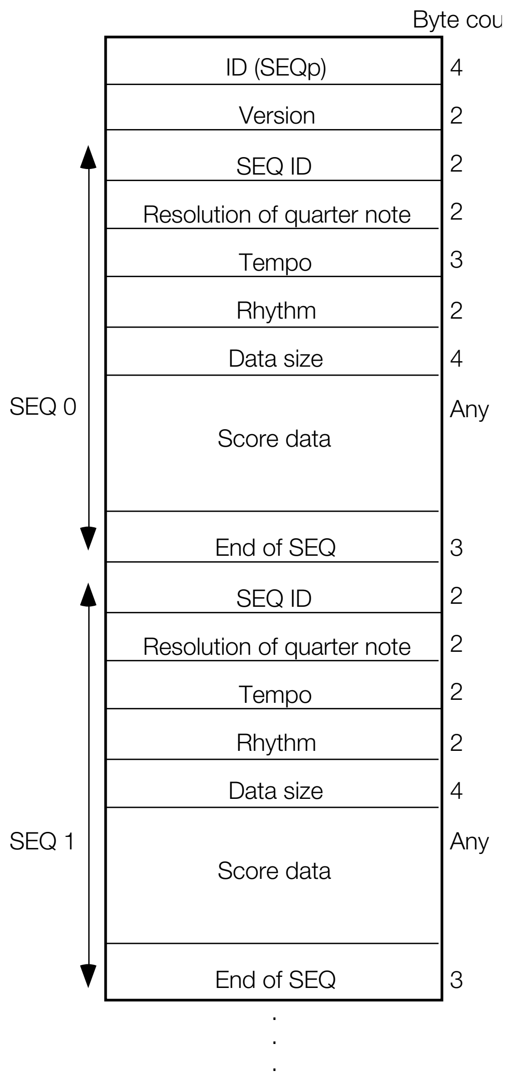
SEP: PS Multi-Track Sequence Data

A SEP is a package containing multiple SEQ data files. SEPs enable multiple SEQ data files to be managed as one file.

SEPs can be accessed by specifying the ID number returned when the SEP is opened, along with the SEQ number of the SEQ data to be accessed. See the Library Reference for details of access-related functions.

The SEP data format is illustrated below.

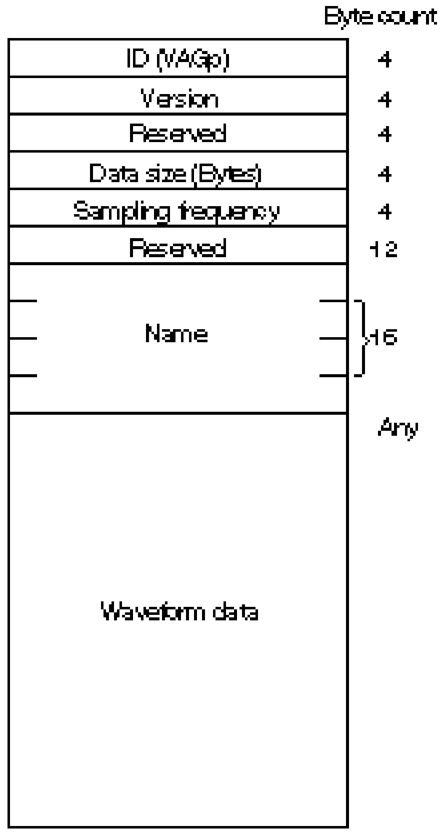
Figure 4–2: SEP Format



VAG: PS Single Waveform Data

VAG is the PlayStation single waveform data format for ADPCM-encoded data of sampled sounds, such as piano sounds, explosions, and music. The typical extension in DOS is ".VAG".

Figure 4–3: VAG Format



VAB: PS Sound Source Data

The VAB file format is designed to manage multiple VAG files as a single group. It is a sound processing format that is handled as a single file at runtime.

A VAB file contains all of the sounds, sound effects, and other sound-related data actually used in a scene. Hierarchical management is used to support multitimbral (multisampling) functions.

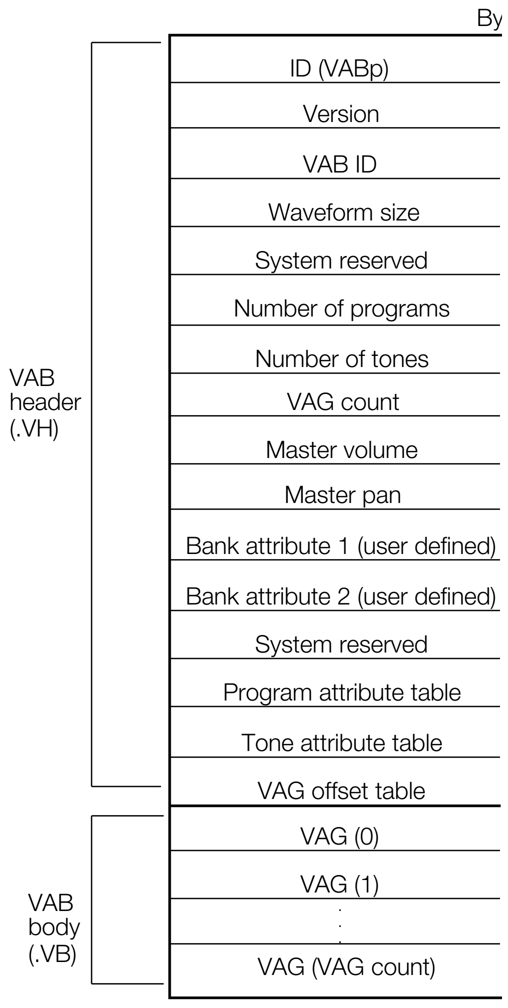
Each VAB file may contain up to 128 programs. Each of these programs can contain up to 16 tone lists. Also, each VAB file can contain up to 254 VAG files.

Since it is possible for multiple tone lists to reference the same waveform, users are able to set different playback parameters for the same waveform, thus giving the same waveform different sounds.

Organization

A VAB format file is organized as follows:

Figure 4–4: VAB Format



* See (b) in Structure

** See (c) in Structure

Structure

The structure of a VAB header is as follows. It is possible to set each attribute dynamically using this structure at the time of execution.

(a) VabHdr structure is contained within the first 32 bytes (See libsnd in the Library Reference for details.).

(b) ProgAtr structure for 128 programs is contained in the program attribute table (See libsnd in the Library Reference for details.).

(c) VagAtr structure for each tone is contained in the tone attribute table (See libsnd in the Library Reference for details.).

(d) VAG offset table contains 3-bit right-shifted VAG data size stored in short (16 bit). For example:

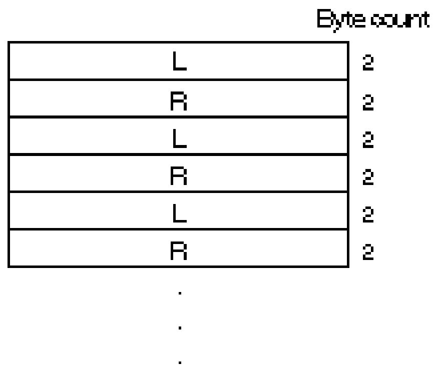
Table 4–1

VAG#	0	1	2	. . .
VAG offset table	0x1000	0x0800	0x0200	. . .
Actual size	0x8000	0x4000	0x1000	. . .
Offset	0x8000	0xc000	0xd000	. . .

DA: CD-DA Data

DA is the PlayStation CD-DA data format.
The typical extension in DOS is ".DA".

Figure 4–5: DA Format


Byte count
L 2
R 2
L 2
R 2
L 2
R 2