

# Net Yaroze File Formats

© 1997 Sony Computer Entertainment

Publication date: May 1997

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

Sony Computer Entertainment Europe  
Waverley House  
7-12 Noel Street  
London W1V 4HH, England

The *Net Yaroze File Formats* manual is supplied pursuant to and subject to the terms of the Sony Computer Entertainment PlayStation™ License and Development Tools Agreements or the Developer Agreement.

The *Net Yaroze File Formats* manual is intended for distribution to and use by only Sony Computer Entertainment licensed Developers and Publishers in accordance with the PlayStation™ License and Development Tools Agreements or the Developer Agreement.

Unauthorized reproduction, distribution, lending, rental or disclosure to any third party, in whole or in part, of this book is expressly prohibited by law and by the terms of the Sony Computer Entertainment PlayStation™ License and Development Tools Agreements or the Developer Agreement.

Ownership of the physical property of the book is retained by and reserved by Sony Computer Entertainment. Alteration to or deletion, in whole or in part, of the book, its presentation, or its contents is prohibited.

The information in the *Net Yaroze File Formats* manual is subject to change without notice. The content of this book is Confidential Information of Sony Computer Entertainment.

PlayStation and PlayStation logos are trademarks of Sony Computer Entertainment. All other trademarks are property of their respective owners and/or their licensors.

# Table of Contents

## About this Manual

About this Release	v
Relate Documents	v
Manual Structure	v
Typographic Conventions	v
Ordering Information	vi

## Chapter 1: 3D Graphics 1-1

RSD: Model Data	1-2
TMD: Modeling Data for OS Library	1-10

## Chapter 2: 2D Graphics 2-1

TIM: Screen Image Data	2-2
------------------------	-----

## Chapter 3: Sound 3-1

SEQ: PS Sequence Data	3-2
VAG: PS Single Waveform Data	3-3
VAB: PS Sound Source Data	3-3



---

## About this Manual

### About this Release

This is the first release of the File Formats manual for Net Yaroze. The purpose of this new book is to provide a single authoritative reference to the PlayStation file formats available for use in Net Yaroze games.

### Related Documentation

The following volumes in the Yaroze documentation set contain related information:

Net Yaroze User's Guide

Net Yaroze Library Reference

### 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.

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

The following conventions apply within structure and function descriptions only:

<i>Convention</i>	<i>Meaning</i>
<b>Medium Bold</b>	Denotes structure or function types and names.
<i>Italic</i>	Denotes function arguments and structure members.

{ } Denotes the start and end of the member list in a structure declaration.



---

# Chapter 1:

## 3D Graphics

---

# 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 1–1: RSD File Structure

ID
PLY file specification
MAT file specification
GRP file specification
Texture count specification
Texture file specification
:
:

## 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 1-2: PLY File Structure

ID
Data length record
Vertex group
Normal group
Polygon group

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 6 0 4 0 1 1 1
0 7 6 4 0 2 2 2
0 7 4 5 0 3 3 3
0 3 7 5 0 4 4 4
0 3 5 1 0 5 5 5
0 2 3 1 0 6 6 6
0 2 1 0 0 7 7 7
0 7 3 2 0 8 8 8
0 7 2 6 0 9 9 9
0 4 0 1 0 10 10 10
0 4 1 5 0 11 11 11
```

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 1-3: 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 1-4: 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 1-5: Normal Descriptor for PLY File

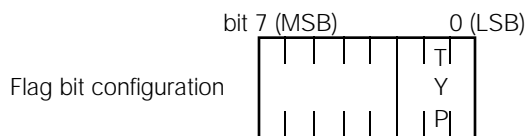
x component	y component	z component
-------------	-------------	-------------

### 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 1-6: 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 1-7: Vertex ordering for quad subdivision

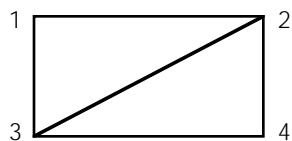


Figure 1–8: Polygon

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 1–9: 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 1–10: 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 1–11: 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
0-5    0 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 1-12: 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 1-1: 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 depending on the material type. There are several different material types. Each is designated by a special type code, as follows:

Table 1-2

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 1-13: 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 1-14: 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 1-15: Textured Polygon/Sprite

TYPE	TNO	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 1-16: 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 1-17: 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 1-18: 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 1-19: 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 1-3

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 1–20: TMD File Format

HEADER
OBJ TABLE SECTION :
PRIMITIVE SECTION :
VERTEX SECTION :
NORMAL SECTION :

**HEADER**

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

Figure 1–21: Structure of Header

ID
FLAGS
NOBJ

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 1–22: OBJ TABLE structure

OBJECT #1
OBJECT #2
:
:

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.

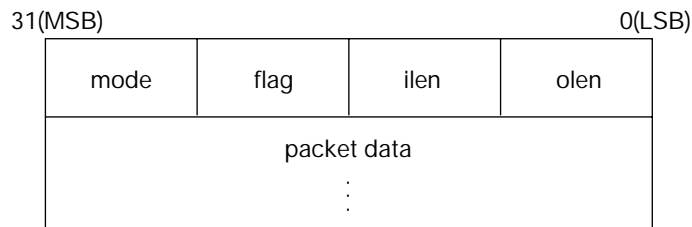
## PRIMITIVE

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 1-23: Drawing Packet General Structure

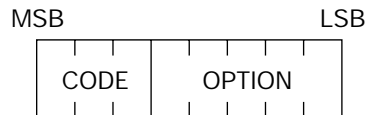


Each item in Figure 2-23 is as follows:

### Mode (8 bit)

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

Figure 1-24: Mode



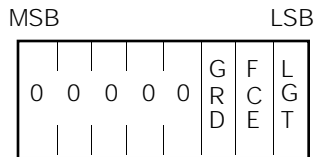
CODE: 3 bit code expressing entities  
 001 = Polygon (triangle, quadrilateral)  
 010 = Straight line  
 011 = Sprite

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 1-25: Flag



- 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

**Ilén (8 bit)**  
Indicates the length, in words, of the packet data section.

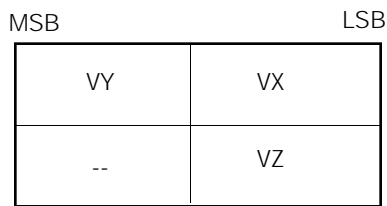
**Olen (8 bit)**  
Indicates the word length of the 2D drawing primitives that are generated by intermediate processing.

**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 1-26: Vertex Structure

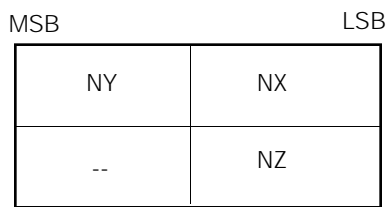


VX, VY, XZ: x, y and z values of vertex coordinates (16-bit integer)

NORMAL

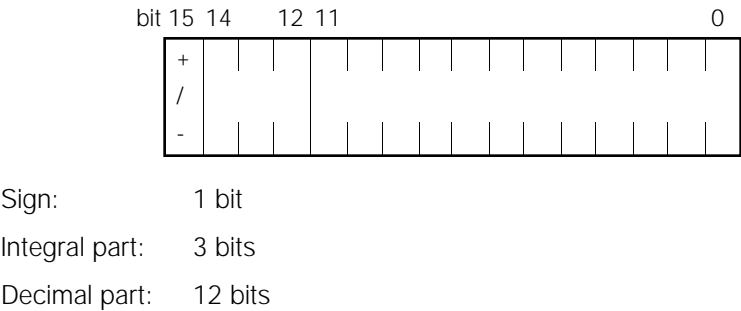
The normal section is composed of a set of structures representing normals. The following gives the format of one structure.

Figure 1-27: Normal Structure



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 1-28: Fixed-Point Format



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.

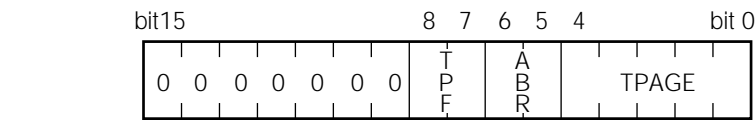
**Normal(n)**  
Index value of 16-bit length pointing to a normal. Same as Vertex.

**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.

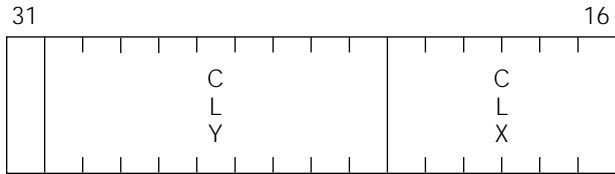
**TSB**  
Carries information on a texture/sprite pattern.

Figure 1-29: 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 1–30: 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

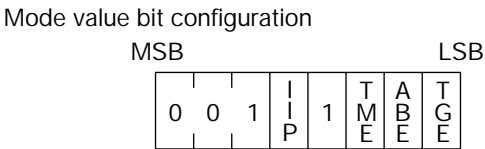
**Packet Data Configuration Example-3 Vertex Polygon with Light Source Calculation**

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 1–31: 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

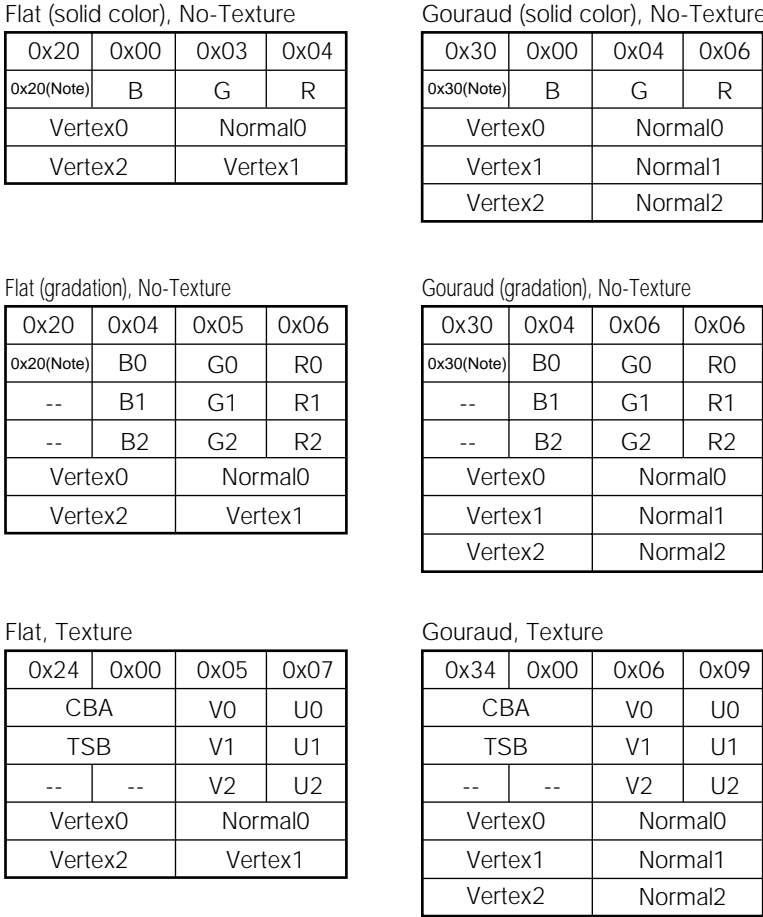
0: On

1: Off (Draws texture as is)

**Packet Data Configuration**

Packet data configuration is as follows:

Figure 1–32: Packet Data for Polygons



Note: same value as mode

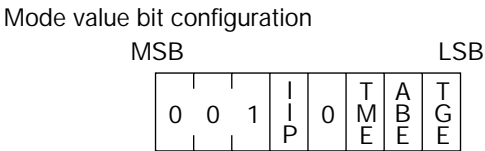
In the above example, the values of mode and flag indicate a single-faced polygon and semitransparency processing not carried out.

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 1–33: Mode Byte



Packet Data Configuration

Packet data configuration will be as follows:

Figure 1-34

Flat, No Texture				Gouraud, No Texture			
0x21	0x01	0x03	0x04	0x31	0x01	0x05	0x06
Note	B	G	R	Note	B0	G0	R0
Vertex1		Vertex0		--	B1	G1	R1
--		Vertex2		--	B2	G2	R2
				Vertex1		Vertex0	
				--		Vertex2	

Flat, Texture				Gouraud, Texture			
0x25	0x01	0x06	0x07	0x35	0x01	0x08	0x09
CBA		V0	U0	CBA		V0	U0
TSB		V1	U1	TSB		V1	U1
--	--	V2	U2	--	--	V2	U2
--	B	G	R	--	B0	G0	R0
Vertex1		Vertex0		--	B1	G1	G1
--		Vertex2		--	B2	G2	G2
				Vertex1		Vertex0	
				--		Vertex2	

Note: Has same value as mode.

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 1-35: Mode Byte

Mode value bit configuration							
MSB				LSB			
0	0	1	1	1	T	A	T
			P		M	B	G
					E	E	E

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

Packet Data Configuration

Packet data configuration is as follows:

Figure 1–36: Mode

Flat (solid color), No-Texture			
0x28	0x00	0x04	0x05
0x28(Note)	B	G	R
Vertex0		Normal0	
Vertex2		Vertex1	
--		Vertex3	

Gouraud (solid color), No-Texture			
0x38	0x00	0x05	0x08
0x38(Note)	B	G	R
Vertex0		Normal0	
Vertex1		Normal1	
Vertex1		Normal2	
Vertex2		Normal3	

Flat, (gradation), No-Texture

0x28	0x04	0x07	0x08
0x28(Note)	B0	G0	R0
--	B1	G1	R1
--	B2	G2	R2
--	B3	G3	R3
Vertex0		Normal0	
Vertex2		Vertex1	
--		Vertex3	

Gouraud (gradation), No-Texture

0x38	0x04	0x08	0x08
0x38(Note)	B0	G0	R0
--	B1	G1	R1
--	B2	G2	R2
--	B3	G3	R3
Vertex0		Normal0	
Vertex1		Normal1	
Vertex2		Normal2	
Vertex3		Normal3	

Flat, Texture

0x2c	0x00	0x07	0x09
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
--	--	V3	U3
Vertex0		Normal0	
Vertex2		Vertex1	
--		Vertex3	

Gouraud, Texture

0x3c	0x00	0x08	0x0c
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
--	--	V3	U3
Vertex0		Normal0	
Vertex1		Normal1	
Vertex2		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 1–37: Mode Byte

Mode value bit configuration							
MSB				LSB			
0	0	1	1	1	T	A	T
			P	E	B	E	G

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



**Packet Data Configuration**

Figure 1-38: 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	

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	

Note: Has same value as mode.

**Packet Data Configuration Example-Straight Line****Bit Configuration of Mode Value**

The primitive section mode value bit configuration is as follows:

Figure 1-39: Mode

Mode value bit configuration

MSB			LSB			
0	1	0	I	0	0	A
			P			B
						E
						0

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 1-40: Packet Configuration for "Straight Line"

Gradation OFF

0x40	0x01	0x02	0x03
0x40(Note)	B	G	R
Vertex1		Vertex0	

Gradation ON

0x50	0x01	0x03	0x04
0x50(Note)	B0	G0	R0
--	B1	G1	R1
Vertex1		Vertex0	

Note: same value as mode

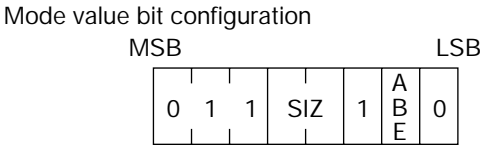
Packet Data Configuration Example - 3 Dimensional Sprite

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 1-41: 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 1-42: 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

---

## Chapter 2:

# 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.

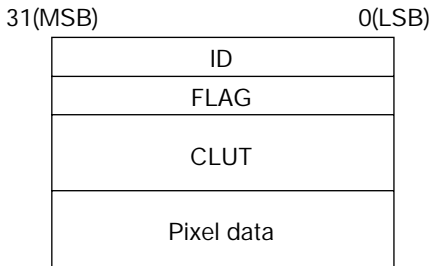
The following are the image data modes (color counts) handled by the PlayStation unit.

- 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.

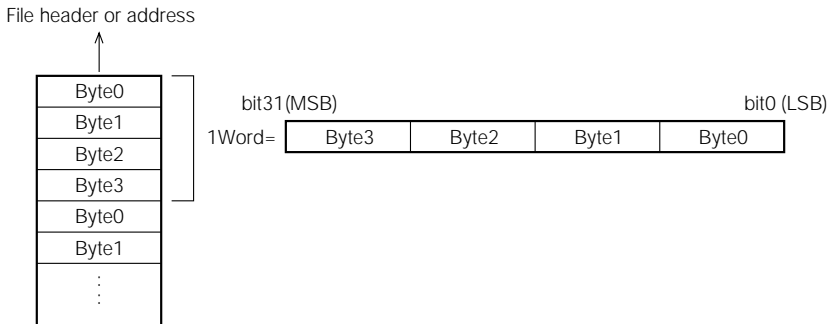
TIM files have a file header (ID) at the top and consist of several different blocks.

Figure 2-1: 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.

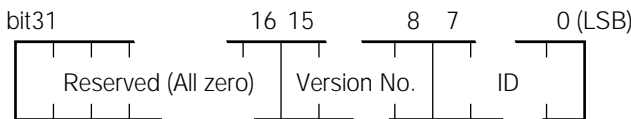
Figure 2-2: The order of bytes in a file



## ID

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

Figure 2-3: Structure of TIM File Header



Bits 0 – 7: ID value is 0x10

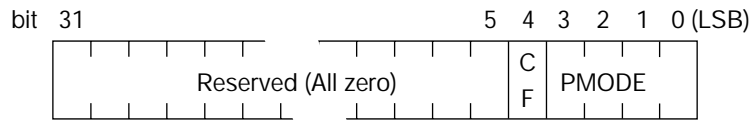
Bits 8 – 15: Version number. Value is 0x00

## 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 2-4: 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

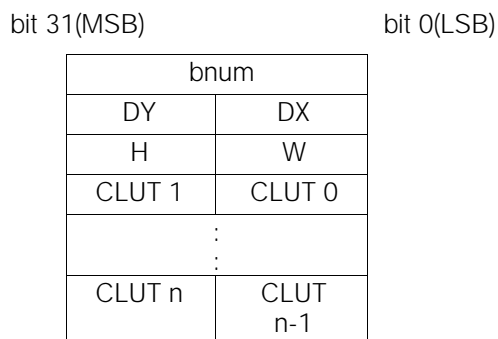
Other: Reserved

## 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 2-5: CLUT**



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).

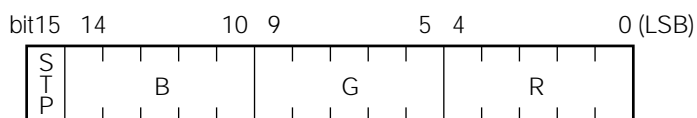
In 4-bit mode, one CLUT consists of 16 CLUT entries. In 8-bit mode, one CLUT consists of 256 CLUT entries.

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 2-6: 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 2-1: STP Bit Function in Combination with R, G, B Data

STP/R,G,B	Translucent 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 2-7: Pixel data



bnum	
DY	DX
H	W
DATA 1	DATA 0
	:
	:
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)

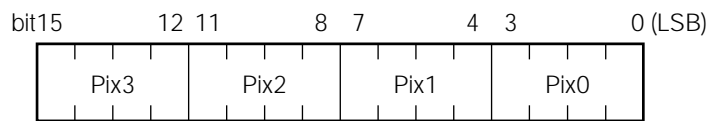
DATA 1~n    Frame buffer data (16 bits)

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 2-8: Frame buffer data (pixel data)**

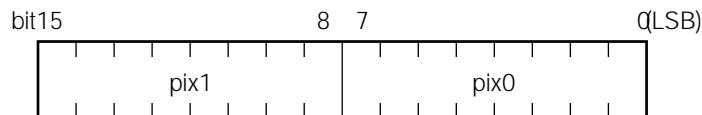
**(a) In 4-bit mode**



pix 0-3 pixel value (CLUT No.)

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

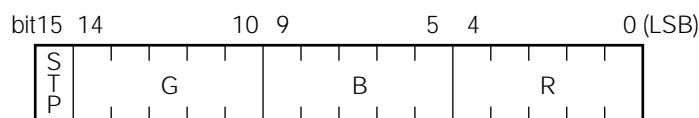
**(b) In 8-bit mode**



pix 0-1 pixel value (CLUT No.)

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

**(c) In 16-bit mode**



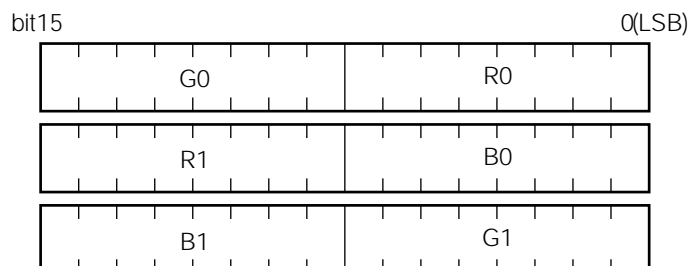
STP    transparency control bit (see CLUT)

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.



---

# Chapter 3:

## Sound

---

# SEQ: PS Sequence Data

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

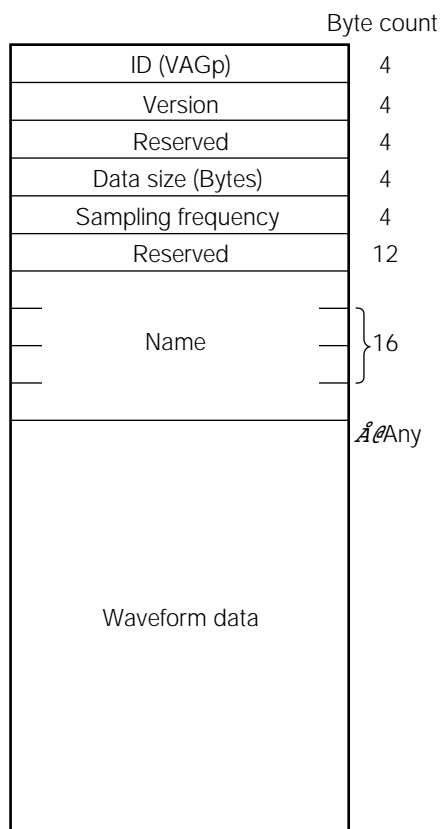
Figure 3-1: SEQ Format

	Byte count
ID (SEQp)	4
Version	4
Resolution of quarter note	2
Tempo	3
Rhythm	2
Score data	Any
End of SEQ	3

## 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 3-2: 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.

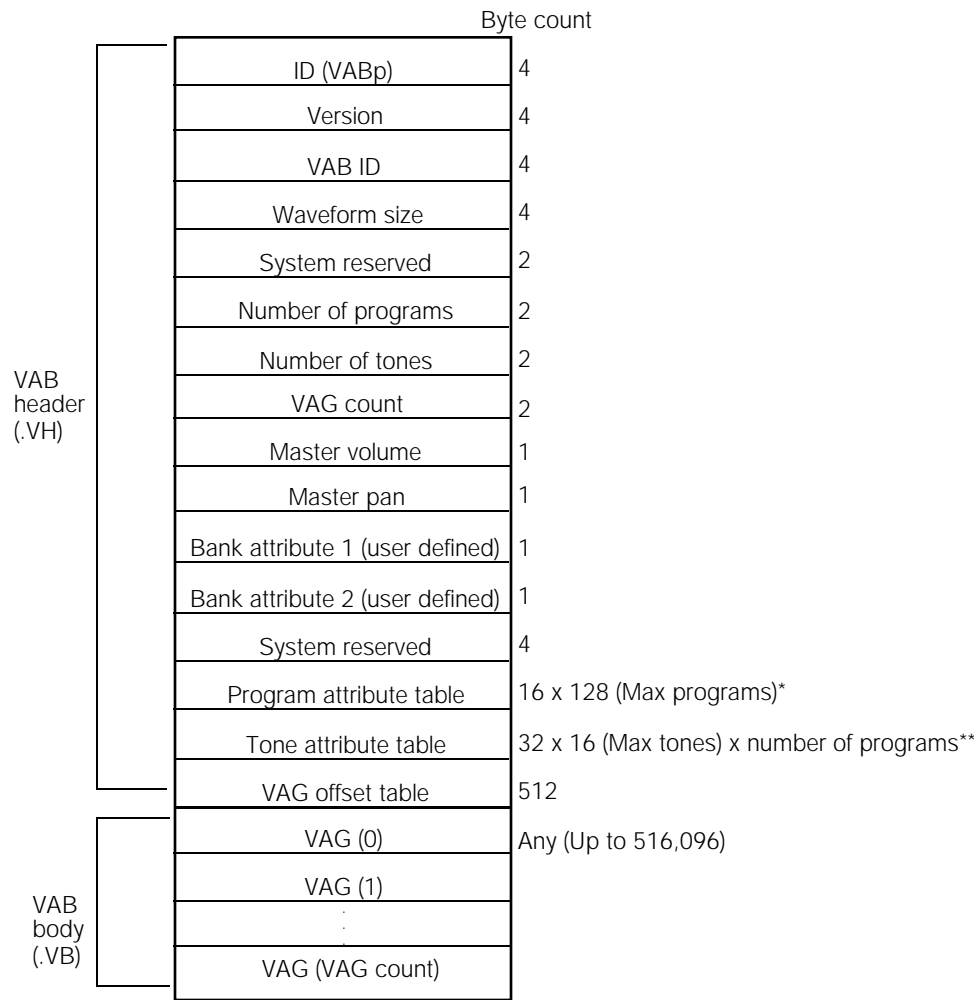
Each VAB file may contain up to 128 programs. Each of these programs can contain up to 16 tone lists. Also, each VAB file can contain up to 254 VAG files.

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 3–3: 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 3-1

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

