User's Guide

Software Development Tool

User’s Guide
Software Development Tool

Published March 1997
©1997 Sony Computer Entertainment Inc. All Rights Reserved.

Written and produced by :
Sony Computer Entertainment Inc.
Akasaka Oji Building
8-1-22 Akasaka, Minato-ku, Tokyo, Japan 107
Inquiries to: Network Business Project
E-mail:ny-info@scei.co.jp
TEL:+81 (0) 3-3475-1711

Sony Computer Entertainment Europe
Waverley House
7-12 Noel Street
London W1V 4HH, England
Inquiries to: The Yaroze Team
E-mail: yaroze-info@scee.sony.co.uk
TEL: +44 (0)-171-447-1616

Sony Computer Entertainment America
919 E. Hillsdale Blvd., 2nd Floor
Foster City, CA 94404
USA
Inquiries to: The Yaroze Team
E-mail: yaroze@interactive.sony.com
TEL: +1-415-655-3600

Table of Contents

About Net Yaroze

In order to get started with Net Yaroze, you should have ***C programming*** competence and you should be familiar with ***2D graphic creation/editing tools***. You should also have a basic understanding of ***3D modelling packages*** and ***sound creation/editing tools.*** Together these will help you get the most out of your Net Yaroze System.

The Net Yaroze Manual Set

There are three books in the Net Yaroze manual set.

1. *Start-Up Guide*
   An introductory booklet explaining the contents and requirements of the Net Yaroze Starter Kit. The *Start-Up Guide* gives step-by-step instructions on setting up the Net Yaroze software on your PC. It also explains how to run software on the Net Yaroze system.
2. *User's Guide* (this document)
   A reference manual explaining how to create software for the Net Yaroze system.
3. *Library Reference*
   A manual listing and describing the functions and structures in the Net Yaroze libraries.

Additional Reading

See the *Additional Reading* section at the end of the Start-up Guide.

1

System Overview

This chapter contains an overview of the Net Yaroze system and an explanation and overview of PlayStation hardware.

Net Yaroze is a revolutionary product which enables anyone to create PlayStation applications using a set of Net Yaroze development tools, a personal computer and a special Net Yaroze PlayStation. Net Yaroze Members can post their applications to an exclusive Net Yaroze Web site where they can be shared and enjoyed by other Members.

The Net Yaroze System

The Net Yaroze system is designed to let you write, debug and test PlayStation applications on a personal computer (PC) linked to a special Net Yaroze PlayStation. The PC, which acts as a host machine, is linked via a dedicated serial cable to the PlayStation which runs the applications.

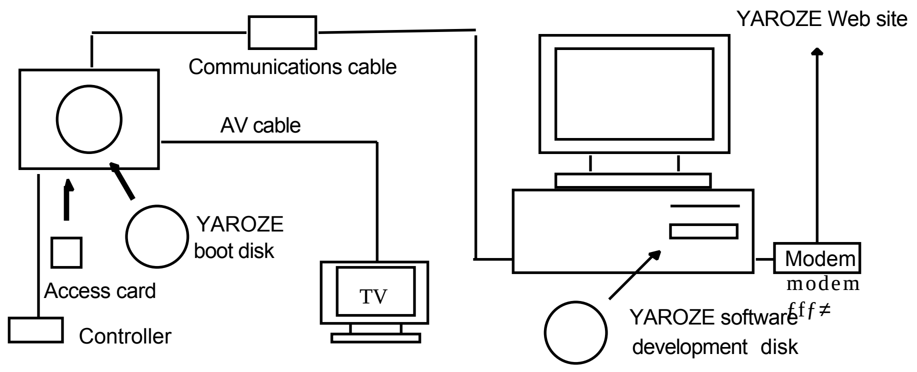


Figure: Net Yaroze System Set Up

Your first step in developing a Net Yaroze application is to create a program using a set of special Net Yaroze programming tools. After you've written, compiled and linked your program, you need to download it to the Net Yaroze PlayStation using SIOCONS, the console tool (discussed in Chapter 17). From there your application can be tested by running it directly on the Net Yaroze PlayStation.

You can also download and test data which is used by your application, such as a sound file. (You need to use the Controller connected to the PlayStation to control your application.)

If your Net Yaroze host computer also has an Internet connection, you can not only post your Net Yaroze programs to the Web site very easily, but also download and immediately execute those programs posted by other Members.

The Net Yaroze Web site

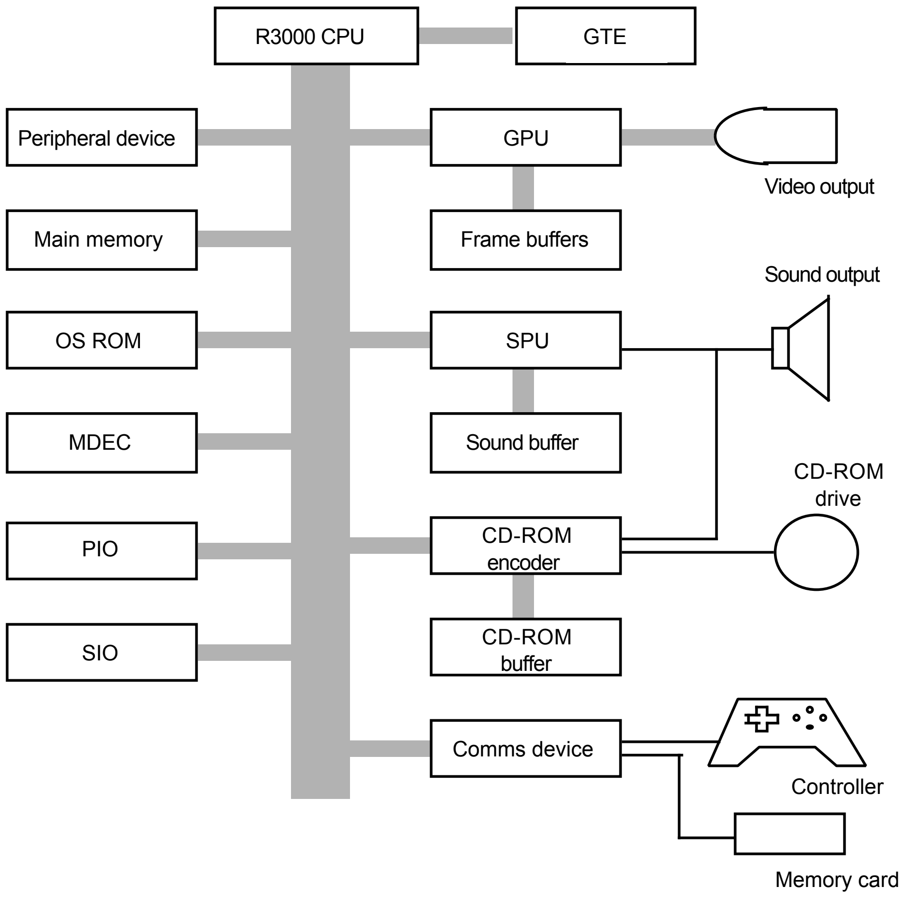
The Net Yaroze Web site is hosted on Sony’s World Wide Web server and is accessible via the Internet. The Net Yaroze Web site has features such as a mail forum, Member's home pages and access to documentation and program libraries. It enables the exchange of information between Members as well as the uploading and downloading of Net Yaroze applications. The site also provides the latest Net Yaroze-related information and technical support.

You'll need an Internet connection and a World Wide Web browser to access the Web site. Please refer to the *Start-Up Guide* for detailed instructions.

PlayStation Architecture

As is shown in the following diagram, the PlayStation system is based on a 32-bit RISC CPU, and comprises a number of processors and devices dedicated to specific functions such as graphics and sound.

Figure : PlayStation Block Diagram



Glossary

GTE	: Geometry Transfer Engine
GPU	: Graphics Processing Unit
SPU	: Sound Processing Unit
MDEC	: Data Decompression Engine
PIO	: Parallel Expansion port
SIO	: Serial Expansion Port

The CPU and Its Peripherals

The PlayStation uses a custom CPU based on the R3000 (33 MHz) 32-bit RISC CPU (little-endian architecture).

The I-Cache

The CPU reads instruction code from the I-cache at approximately 5 times the speed of main memory. Once instruction code has been read, it is stored in the I-cache within the CPU and can be re-executed without accessing main memory. The I-cache cannot be directly accessed from a program.

The CPU is equipped with a 4 KB I-cache. The PlayStation's logical memory space is divided into 4 KB units and these are multiply-mapped into the I-cache.

The D-Cache

The D-cache uses a special structure called a scratch pad which is mapped into 1 KB of logical memory space (addresses 0x1f800000~0x1f8003ff). The program developer can freely access this scratch pad area.

The General-Purpose Registers

There are thirty-two 32-bit general-purpose registers. The compiler assigns a specific use to each register as shown in the table below. Threaded databases and application development using the assembler require that registers be used according to the assignments shown below.

Register No.	Macro (1)	Macro (2)	Assembler Assignments	
0	R_ZERO	R_R0	zero	always 0
1	R_AT	R_R1	AT	reserved for the assembler
2~3	R_V0~1	R_R2~3	v0~1	values returned by functions
4~7	R_A0~3	R_R4~7	a0~3	function arguments
8~15	R_T0~7	R_R8~15	t0~7	destroyed within functions
16~23	R_S0~7	R_R16~23	s0~7	saved within functions
24~25	R_T8~9	R_R24~25	t8~9	destroyed within functions
26~27	R_K0~1	R_R26~27	k0~1	reserved for the kernel
28	R_GP	R_R28	gp	global pointer
29	R_SP	R_R29	sp	stack pointer
30	R_FP	R_R30	fp	frame pointer
31	R_RA	R_R31	ra	return address

Table: R3000 General-Purpose Registers

Return Address

The R3000 does not have any built-in support for subroutine calls. Instead, a subroutine call is performed by executing a jump instruction that saves the return address in a register. The register that contains the return address can be specified using the assembler, however, the C compiler will always use general-purpose register 31.

The Stack

The R3000 does not have any built-in support for a stack. Instead, the compiler implements a stack by storing a stack pointer in general-purpose register 29. In addition, efficient operation of function frames (memory areas used for automatic variables and as working areas) is achieved by saving the start address for the frame in general-purpose register 30. This address is known as the frame pointer. The value of the frame pointer is determined from the value of the stack pointer. When a module is activated, the frame pointer and stack pointer have identical values.

The Global Pointer

The R3000 can access memory with a register-indirect addressing mode that uses a signed 16-bit offset. For efficiency, the compiler keeps up to 64 KB of data in a block called the bss section. The midpoint address of this block is stored in general-purpose register 28. Using register-indirect addressing and a 16-bit offset from general-purpose register 28, the R3000 can access data in the bss section with a single instruction. The address in general-purpose register 28 is known as the global pointer and does not change within a module.

Main Memory

The PlayStation is equipped with 2 MB of main memory. Addresses are allocated in this memory starting from 0x0000 0000. This address space is referred to as 'physical memory space'.

The CPU's memory space consists of 32-bit addresses and is known as 'logical memory space'. Physical memory space is mapped to three areas in logical memory space. The CPU is not equipped with a virtual memory manager so this mapping between the two memory spaces is fixed as shown in the table below.

Physical Memory	Logical Memory	Segment Name	I-Cache
0x00000000~0x001fffff	0x00000000~x001fffff	ku	Available
	0x80000000~0x801fffff	k0	Not available
	0xa0000000~0xa01fffff	k1	Available

Table: Physical Memory and Logical Memory

The OS ROM

The PlayStation is equipped with 512 KB of ROM. The OS kernel and the boot loader are stored in this ROM. Access to the OS ROM is not permitted from an application program.

The DMA Controller

A DMA Controller is attached to the CPU. The DMA Controller is used for data transfers between memory and devices according to instructions from the CPU.

Graphics System

The PlayStation is equipped with a graphics processing unit (GPU). The GPU features CRTC functions for displaying frames on the screen, as well as for performing high-speed polygon drawing using a frame buffer.

The Frame Buffer

The GPU has a 1 MB frame buffer. The frame buffer is described by a two-dimensional address space (1024 x 512) made up of 16-bit pixels. The frame buffer's memory space is managed by the GPU and cannot be directly accessed from the CPU.

The Display

The GPU displays the contents of an arbitrary rectangular area within the frame buffer directly on the CRT display. This area is called the 'display area'.

The table below shows the 10 supported screen modes of the GPU.

NTSC		PAL	
Interlaced	**Non-interlaced**	**Interlaced**	**Non-interlaced**
256(H) x 480(V)	256(H) x 240(V)	256(H) x 512(V)	256(H) x 256(V)
320 x 480	320 x 240	320 x 512	320 x 256
512 x 480	512 x 240	512 x 512	512 x 256
640 x 480	640 x 240	640 x 512	640 x 256
384 x 480	384 x 240	384 x 512	384 x 256

Table: Screen Modes (NTSC and PAL)

The GPU supports two color-display modes: 15-bit direct mode (32,768 colors) and 24-bit direct mode (full color).

In 15-bit direct mode 32,768 colors can be displayed simultaneously. This is fewer colors than can be displayed in 24-bit direct mode, however, color calculations within the GPU during drawing are always performed using 24 bits. Consequently, pseudo-full color can be achieved in 15-bit direct mode by dithering.

In 24-bit direct mode 16,777,216 colors can be displayed simultaneously. However, in 24-bit direct mode only static images (still pictures) can be displayed from image data that has been transferred to the frame buffer. The drawing functions of the GPU cannot be used in 24-bit direct mode.

In 24-bit direct mode a single pixel has a bit length of 24 bits, but the display position coordinates in the frame buffer must be specified as if the pixel size were 16 bits. Thus, a 640 x 240 image in 24-bit direct mode will have an actual size of 960 x 480 in the frame buffer.

Drawing Capabilities

The GPU supports the following drawing functions.

Name	Description
Polygon Drawing	4-bit CLUT (16 colors/polygon) 8-bit CLUT (256 colors/polygon) 16-bit direct (32768 colors/polygon) Flat shading, Gouraud shading Texture mapping
Line Drawing	Gradation is possible
Image Transfer	CPU ↔ frame buffer Frame buffer → frame buffer
Other Functions	α Blending (semi-transparency), dithering, clipping

Table: GPU Drawing Functions

Polygon Drawing

The GPU is capable of drawing polygons. The polygons that the GPU operates on can be either 3-sided or 4-sided figures. These polygons are drawn by specifying the screen coordinates for each of the vertices of the figure. The shape of the polygons, their color and/or texture can all be specified. For example, the GPU can perform 'texture mapping' (in which an image from an arbitrary area of the frame buffer is pasted onto the polygon surface), 'flat shading' (solid color paint out) and 'Gouraud shading' (gradation paint out, in which each pixel of the polygon is given a color calculated from the colors assigned to the vertices).

Images and texture patterns pasted onto polygons are transferred to the frame buffer before drawing the polygon. Data is stored in the frame buffer in 256 x 256 pixel pages, in which there can be as many pages as available memory will allow. These 256 x 256 areas are referred to as 'texture pages'. The size of a texture page in the frame buffer is determined by the color mode, as described below.

CLUT Processing

There are 3 color modes for texture patterns: a 4-bit CLUT mode, an 8-bit CLUT mode and a 15-bit direct mode.

CLUTs (Color Look-Up Tables) are used in 4-bit and 8-bit CLUT modes. A CLUT is a set of 16 or 256 RGB values stored in the frame buffer. These RGB values represent the colors that will ultimately be displayed on-screen. Each RGB value in the set is numbered in order from left to right. Each number corresponds to a pixel in a sprite pattern with the associated RGB value used to determine the pixel’s color. CLUTs can be selected at the sprite level which means each sprite can have its own independent CLUT.

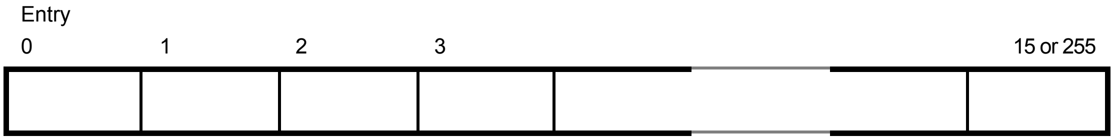


Figure: CLUT Structure

Each entry in the CLUT has the same structure as a single pixel in 15-bit direct mode. Therefore, one CLUT is equivalent to 1 x 16 or 1 x 256 pixels of image data.

Sound System

The PlayStation sound system is made up of the Sound Processing Unit (SPU) and the CD-ROM decoder. Audio output from the CD-ROM decoder enters the SPU, is mixed with the output from the SPU, then transformed into the final audio output.

The CD-ROM Decoder

Data read from the compact disc (CD) can be reproduced directly as audio output or transferred as sound data to main memory.

ADPCM data, defined by the CD-ROM XA standard, and CD-DA 16-bit PCM data are directly output as sound. Data read from the drive to the CD-ROM buffer is processed by the sound source within the CD-ROM decoder. The resulting audio signal is sent to the SPU via the mixer built into the decoder.

The SPU

The Sound Processing Unit (referred to as the SPU) is equipped with 24 voices and controls 512 KB of memory. This memory is known as the sound buffer. Compressed waveform data stored in the sound buffer is played back by the SPU at a sampling frequency of 44.1 KHz, in response to commands from the CPU. The sound buffer is composed of a one dimensional memory space made up of 2-byte units. The sound buffer cannot be directly accessed from the CPU.

The sound buffer can be used as a work area for reverb and other special effects. The sound buffer also serves as a temporary buffer during transfer of sound data from the CD or SPU to main memory. The sound buffer enables data to be transferred to main memory without interrupting sound production.

Each voice in the SPU is capable of operations such as pitch variation, envelope processing (attack, decay, sustain, release) and volume control.

Other Systems

The PlayStation is equipped with the following systems in addition to those described above.

The GTE

The GTE is a coprocessor that performs the vector and matrix arithmetic operations essential for 3D graphics. The GTE performs its processing using fixed-point arithmetic. The results of GTE calculations can be directly incorporated in drawing commands sent to the GPU.

The Data Decompression Engine

The data decompression engine performs high-speed reverse DCT conversion. The engine can decompress JPEG data and MPEG data (only within the frame). The data decompression engine is not available in the Net Yaroze system.

The Controller

The Controller serves as the interface by which players issue commands to an application. The main PlayStation unit has connectors for two Controllers. In professional PlayStation development, more Controllers can be connected using Multi tap ports. However, Multi tap ports cannot be used with the Net Yaroze system.

The Memory Card

The Memory card provides storage for PlayStation application data when the PlayStation is reset or turned off. The main PlayStation unit has slots for two Memory cards. In professional PlayStation development, a larger number of cards can be connected using Multi tap ports and Multi taps. Note, however, that Multi tap ports cannot be used with the Net Yaroze system.

Expansion Ports

Two expansion ports are provided: one serial, the other parallel.

In professional PlayStation development, two PlayStations can communicate with each other through a Link cable connected to the serial ports. However, the Net Yaroze system uses this serial port to connect the Net Yaroze PlayStation to your computer. Thus the PlayStation Link cable cannot be used with the Net Yaroze system.

The parallel port is reserved for future expansion and is currently not supported.

2

The PlayStation Development Environment

The PlayStation is equipped with the PlayStation operating system (OS). This OS was developed specifically for the PlayStation's R3000 CPU and incorporates extremely novel concepts used by both the Professional Development System and Net Yaroze. This chapter provides an overview of the Professional PlayStation Development System and describes the features of Net Yaroze.

The Professional PlayStation Development System

The PlayStation OS was developed specifically for the PlayStation's R3000 CPU and is based on new and innovative concepts.

The environment and services provided by a machine's operating system can greatly affect how efficiently a program can be developed. If the CPU and peripheral devices operate at sufficient speed, the programmer can rely on the services provided by the operating system to control the hardware. Application development can proceed smoothly and rapidly as there is no need to spend time pushing hardware performance to its limit. Thus, the programmer is free to concentrate on application development.

The design concept of the PlayStation OS is to provide the game developer with an optimal environment for developing interrupt-driven programs. Based on this concept, the kernel of the PlayStation OS provides a collection of services (subroutines) to control the R3000 and the PlayStation hardware.

Access to these services is provided via a set of library functions written in the C programming language. Using C means that the source code can be more readily understood and maintained. Furthermore, the structured nature of the language and the ease with which functions can be called allows program development to proceed easily.

The Professional PlayStation Development System is based on the concepts described above. It is composed of a small-scale OS kernel that provides efficient interrupt processing and multitasking support, and a hardware management library and middleware for high-level services such as a MIDI driver and 3D graphics system. The minimum memory required by the OS has been limited to 64 KB, and the OS has been designed so that the developer can have the maximum freedom possible under a CD-ROM system. SCE's R&D teams continue to expand the professional PlayStation library and middleware based on requests from professional application developers.

In offering a high degree of freedom, however, this environment requires developers to have a high level of expertise in design and development. As of this writing, there are more than 1600 C language functions available in the PlayStation library. With such a large number of functions it is possible to perform a single task in many different ways, each with its own set of advantages and disadvantages. Professional developers must choose the particular method that is best suited to the task at hand.

The Professional PlayStation Development System can be described as a collection of parts designed for use by specialists, where the emphasis is on variety and freedom, rather than uniformity, consistency, or ease of use.

The Net Yaroze PlayStation Development System

The Net Yaroze PlayStation Development System was designed with the goal of making the development environment simpler and easier to understand by offering a subset of the Professional PlayStation Development System.

The features of the Net Yaroze PlayStation Development System are described below.

Programming in C

All PlayStation OS services are provided as C functions so all programming can be done in C.

Ease of Using R3000 Functions

Interrupt processing with the R3000 is known to be complicated. The PlayStation OS simplifies this task by handling all interrupt processing in the kernel. In addition, the OS provides a 'callback' mechanism to notify the user that an interrupt has occurred.

Emphasis on Vertical Synchronization (VSync) Interrupts

VSync interrupts are the key to efficient operation of the PlayStation since it was designed expressly for playing video games. VSync interrupts are the focal point of the PlayStation’s callback system. This system provides a simple interface for coding interrupt processing routines directly in C.

Compactness of Programs

The PlayStation and the development host are connected through a simple serial interface in the Net Yaroze system. In order to overcome the slow data transfer speed of the serial interface, system programs such as graphics, sound, CD-ROM and the debugging monitor are all loaded into main memory from the boot disk as resident libraries. The addresses of these resident libraries are written directly into the application program by the linker so the libraries can be easily accessed by the program. Thus the size of the application program, and hence the download time, is kept to a minimum.

Parallel Processing

The PlayStation OS performs CPU program execution concurrently with dedicated hardware processing. In the Net Yaroze system, parallel processing functions are provided by making “polling” central to programming style.

3

Developing an Application

This chapter presents an overview of how a PlayStation application is created with the Net Yaroze system.

In the Net Yaroze system, application data such as sound and graphics files (e.g. .bmp files), can be easily converted from standard file formats into PlayStation file formats.

The application development process is based on that of standard C program development, so standard development procedures can be used. The GNU C compiler and associated utilities are provided as programming tools.

For detailed descriptions of each of the items given below, please refer to the chapter describing the hardware or the chapter describing the corresponding service.

Creating a Program

The Net Yaroze application development process is based on the standard process used in C language program development so it should be familiar to people who have had experience developing programs in C.

Creating Source Code

Source code should be saved in a standard MS-DOS text file using any commercial editor designed for writing programs.

Compiling and Linking

Source code files must be compiled and linked to create an executable program. The Net Yaroze system provides a special library, the GNU C compiler and various tools associated with the compiler for creating the executable program.

Test Runs

Executable programs can be run and tested on the Net Yaroze PlayStation. The console tool, SIOCONS, (described in Chapter 17) can be used to download the executable program from your PC to the Net Yaroze PlayStation.

Debugging

When your program fails to operate as it should, you can debug it at the source code level by using the GNU debugger (gdb) provided with your Net Yaroze kit. GDB allows you to step through your program and see what's happening as it executes.

Using Makefiles

Makefiles are a convenient way to simplify the series of operations needed for compiling and linking. The 'Make' command, which operates on makefiles, allows applications to be maintained at a high level. Make is provided with the Net Yaroze system.

Making a Library of Useful Routines

You can increase the efficiency of application development by putting frequently called routines and subroutines that are shared by different programs into a library. In the Net Yaroze system, several GNU utilities are provided to assist this process. Two of these are 'ar', the librarian, which builds a library from object files and 'nm', the object symbol management tool, which provides details (such as the start address) of symbols in object files. (See Chapter 16, *Programming Tools*, the documentation with the GNU compiler on your Net Yaroze PC disk and commercially available documentation, for more information on the GNU utilities.)

Creating Data

Data needed within a PlayStation application can be broadly divided into three categories: 2D graphic data, 3D graphic data and sound data. (Note that a fourth type, movie data, while available in the Professional PlayStation Development System cannot be used in the Net Yaroze system.)

Each of these data types is saved in a special PlayStation data format. The Net Yaroze system provides utilities to convert files from the standard formats in which they were created into PlayStation formats.

2D Graphic Data

2D graphic data used by the PlayStation consists of image data, which is used for sprite pattern data, and texture data. PlayStation-format 2D graphics data is referred to as TIM data.

There is a special utility in the Net Yaroze system to convert data created using any Windows or Macintosh painting tool to TIM data.

Data type:	TIM - image data (sprite pattern and texture)
Tools provided:	Utilities to convert from existing formats such as BMP, PICT, (among others) to TIM format
Operating environment:	Windows, MS-DOS

3D Graphic Data

3D graphic data used by the PlayStation is used by 3D applications for modeling data. PlayStation-format 3D graphics data is referred to as TMD data.

In the Net Yaroze system, there is a special utility to temporarily convert data from DXF format - the standard 3D modeling format - into the artist-oriented RSD format file, then into the binary TMD format data used by the library.

Data types:	RSD - modeling data (defines the shape of objects) TMD - modeling data (binary format)
Tools provided:	DXF → RSD format, RSD → TMD format converters
Operating environment:	Windows, MS-DOS

Sound Data

Sound data used by the PlayStation can be either score data or waveform data. Score data is known as SEQ data while waveform data is referred to as VAB data (Note: This data is also known as VAG data - VAG is single-sound data, whereas a VAB is a collection of VAGs). All of these data types can be directly processed using the sound services.

In the Net Yaroze system, a special utility converts AIFF data or standard MIDI data (SMF), which has been created using commercial sound tools, into SEQ and VAB data.

Data types:	VAG - waveform (single-sound) data VAB - waveform (bank - collection of VAGs) data SEQ - Score data
Tools provided:	SMF → SEQ format, AIFF → VAG format converters
Operating environment:	Windows, MS-DOS

Data Verification

The Net Yaroze system provides a viewer and player which, when called from the DOS prompt of your PC, will display sound and graphic data on your TV monitor via the Net Yaroze PlayStation. Thus you can verify how sound and graphic data will look when your application is actually run.

Using these tools it is possible to capture application run-time images.

The Link Between Data and Programming Tools

In order to provide graphic and sound data files for PlayStation applications, you need to create and/or edit standard-format data files in a commercial graphic or sound application, then use the Net Yaroze conversion utilities to convert them into the appropriate PlayStation file formats. You can then use these files as part of a Net Yaroze application (where they will be loaded into main memory and accessed by the program code).

Programming Style

The following is an explanation of programming methods characteristic of PlayStation applications together with an explanation of PlayStation terminology.

Basic Style

The basic flow of graphics-processing in the PlayStation can be described by the following 3-step sequence. These steps are performed simultaneously, enabling a continuous flow of new data to the screen.

Basic Style Steps:

(1) Data describing a 'world' is placed in memory and is used to create a structure known as a 'drawing command' list.

(2) The 'drawing command' list is sent to the GPU which draws the corresponding polygons in the frame buffer.

(3) The rendered image in the frame buffer is displayed on-screen.

List structures are created in main memory, whereas screen images are created in the frame buffer. Two sets of list structures and two sets of screen images are maintained. This allows the CPU to be creating the first list while the second is being transferred to the GPU. Similarly, the GPU can be drawing the first screen image while the second is being displayed in the frame buffer. GPU image drawing and display are shown in the figure below.

Figure: The GPU Draws One Image and Displays Another Simultaneously

The problem with drawing a single frame of 3D graphics is that it can be time-consuming. To overcome this limitation, image creation is divided into two processes which are performed in parallel: drawing command list creation and polygon drawing.

Of the Basic Style Steps listed on the previous page, step (1) is executed by the CPU according to the program stored in main memory. Step (2) is performed automatically by the DMA Controller, a hardware device dedicated to data transfer. Step (3) is carried out automatically by the image display part of the GPU. The CPU, and consequently the program, has no direct knowledge of the details of the execution of steps (2) and (3). The CPU is only involved in providing the dedicated hardware with very small amounts of data such as the display location and the start address for data transfer. The benefit of performing this processing in parallel is that the CPU can devote almost all of its time to creating drawing command lists.

The technique of saving a previous image for later display also serves to increase the time that is available for image creation. Japanese and US TV NTSC receivers display a frame every 1/30 or 1/60 of a second (30 or 60 frames are displayed in a second. In Europe, the display rate is 25 or 50 frames per second). While one frame is being displayed, the next frame is prepared for display. Even if the next frame is not ready in time to be displayed, the one currently being displayed can be held until the next one is ready. Thus a continuous display is maintained on-screen.

In the PlayStation, the CPU, DMA Controller and GPU work together to control this operation as described in the following.

When the CPU completes a drawing command list structure for one full screen and, when the DMA Controller has finished transferring the previous list to the GPU for drawing, then the CPU instructs the DMA Controller to transfer its newly created list to the GPU. At the same time, the image that has been drawn is set up for display in the display part of the GPU.

These steps are summarized as follows:

- CPU makes command list structure
- DMA Controller transfers list structure to GPU
- GPU draws one image while it displays another image

However, this process is governed by a time limit. It is not possible to freely switch the displayed image at any time. If image data that is in the process of being displayed is exchanged with other data, image discrepancies and display hardware activity will appear on the screen as noise. Image data switching must be performed at a time when the act of displaying to the screen is finished (the GPU has finished *making* the image on-screen and the whole image is being shown). The period of time during which this condition is met is called the vertical blanking interval. Vertical blanking occurs at a fixed rate of 60 times per second in NTSC and 50 times per second in PAL. The start of each vertical blanking interval is communicated to the CPU through a vertical synchronization interruption provided by the hardware.

Thus, the CPU completes list creation, waits for the DMA Controller to complete its data transfer, then waits for the vertical synchronization interruption. When these three conditions are met, the CPU switches the display image in the GPU and sets up the address of the new list in the DMA Controller. Once this process is complete, the CPU starts creating the next drawing command list. It reads the state of the Controller and uses this information as the basis for creating a new image. The basic execution style of PlayStation programs involves a repetition of this sequence.

Double Buffering

Double buffering is a technique whereby two equivalent data storage areas, or buffers, are used to simultaneously generate and transfer data or render and display an image. These buffers are swapped at appropriate times so that the two operations can be performed in parallel.

In standard PlayStation programs, two drawing command lists called 'ordering tables' (OT) (refer to *Z sorting,* below) are maintained in main memory. Double buffering speeds up image creation by enabling two operations to be performed in parallel. Similarly, two screen images are maintained in the frame buffer to guarantee time for image creation.

Non-blocking Functions

Most of the processes that are actually performed by dedicated hardware, such as graphics drawing and loading from CD-ROM, can be done in parallel with CPU program execution. The functions that activate this kind of parallel processing are called non-blocking functions. A non-blocking function transmits the relevant processing request to the hardware or OS, then terminates once the request has been recorded. The actual processing is performed after the non-blocking function has terminated.

In the Net Yaroze system, the completion of processing requested by means of non-blocking functions can be checked at any time by calling a special test function. This programming style, in which a program that makes a processing request and explicitly tests for the completion of that processing, is called polling.

Z-Sorting

When 3D objects are displayed on a 2D screen, a proper image is obtained only if surfaces that are hidden by other surfaces are eliminated. In the PlayStation this is achieved by means of a Z-sorting algorithm.

Z-sorting is a method of omitting the surfaces that should be concealed by drawing each surface in order, starting from surfaces that are furthest from the viewpoint to those that are nearest. In terms of rendering speed and hardware requirements, Z-sorting is more efficient than other methods (such as Z-buffering). However, the time required for sorting tends to increase dramatically as the number of elements to be sorted increases. The PlayStation performs reliable Z-sorting by using an OT (Ordering Table) and a drawing command list structure.

An OT is an empty drawing command array. An OT contains, for each drawing command, its own size and a pointer that specifies the next drawing command. In the initial state, as well as the cleared state, the pointers of each of the elements of an OT all point to the adjacent element. In other words, element N points to element N+1, and so on. In the initial state an OT is physically an array, and logically, a single list (see *Ordering Tables*, 6.5).

Each time a drawing command is created, the element number in the OT array is assumed to be the coordinate value along an axis perpendicular to the screen, (the z-axis, where x = vertical, y = horizontal and z = depth), and the new command is inserted between the corresponding OT array element and the element to which that element points by means of a list operation.

The creation of drawing commands is carried out independently of the coordinate value of the z-axis. However, the list operations ensure that the commands all form a single list. Moreover, all drawing commands are linked in between the OT element representing their coordinate value on the z-axis and the following OT element thus ensuring that the list will be in a sorted state. (Using the example above, the new command is inserted in the list between N and N+1, N now pointing to the new command and the new command pointing to N+1.)

Thus, even if the number of drawing commands increases, the amount of time required for sorting does not greatly increase, and processing is more likely to be reliable.

This sorted linked list of drawing commands is then sent to the GPU with the end that holds the largest z-axis value first (i.e. the drawing objects furthest from the viewpoint first). As this is a simple operation, it is performed by the dedicated hardware unit known as the DMA Controller. (The DMA Controller is known as the hardware unit which executes Z-sorting at high speed. It also performs initialization of OTs at high speed.) Thus, drawing can be performed according to the Z-sorting algorithm, starting from surfaces far away to those closest to the viewpoint.

Critical Section

The EnterCriticalSection() function can block all interrupts through a software operation. Sections of a program protected in this way are referred to as critical sections. Within a critical section, all interrupts processed within the operating system, including vertical sync interrupts, are blocked.

In this state most library functions will not operate properly. However, there are library functions, such as Exec(), that are guaranteed to operate normally *only* within a critical section.

The ExitCriticalSection() function is used to leave a critical section allowing interrupts to again be recognized.

Programs are always in a critical section at the start of execution.

Synchronizing with Vertical Synchronization Interrupts

In order to display smooth images on the screen from the PlayStation, the executing program must be synchronized with the timing of vertical synchronization interrupts (VSyncs). (When a VSync interrupt occurs, all lines of an image have been drawn on the screen. See the diagram below).

Figure: Displaying an Image On-Screen

The PlayStation OS is equipped with two methods for achieving VSync synchronization:

1. VSync()
   When this function is called with 0 specified as the first argument, VSync(0), it waits until a VSync interrupt occurs. Explicit synchronization of program execution with VSyncs can be achieved by calling this function in the main loop.

2. Callback
   Callbacks allow certain functions to be executed when a VSync is generated. For details, please refer to the following section, *Callback Functions*.

Callback Functions

The VSyncCallback() function is used to enter a pointer to a function into the callback system. This causes the function to be executed automatically when a VSync interrupt is generated.

A function specified in this way is called a callback function. Callback functions are executed with the same scope as the point where VSyncCallback() was called, so data can be shared with the main program through external variables. The stack resides in 4 KB of memory reserved within the operating system. If the space used by automatic variables exceeds this size, OS code will be destroyed.

Callback functions are executed as critical sections. In other words, the functions are executed with interrupts disabled. Caution should be observed as leaving the critical section from within a callback function can lead to unpredictable results.

While a callback function is being executed, the main flow of the program (whatever was being executed when the VSync occurred) is forcibly suspended temporarily. On return from the callback function, information related to the main flow, which had been saved, is restored by the CPU and execution of the main flow resumes.

Processing of Vertical Synchronization Interrupts within the OS

Immediately after a VSync interrupt occurs, the OS will communicate with the Controllers and the Memory card before any callback functions are called. (Communication with the Memory card can only occur within the read() or write() functions.)

When the main flow calls VSync() and waits to synchronize with a VSync interrupt, communication with the Controllers will take place within VSync() before any callback functions are called. When control returns from the callback function, VSync() will exit and the main flow will resume.

Multiple Vertical Synchronization Interrupts

If a callback function takes a long time to complete, the next VSync may occur while the callback function is still executing. Interrupts are disabled during execution of a callback function, so subsequent VSyncs are saved and held pending. When control returns from the callback function, a VSync interrupt that was held pending becomes valid and the callback function will be called again.

Allowing a VSync interrupt to occur while a callback function is executing is undesirable. This behavior can cause delays in updating the screen or playing back music and should be avoided. Callback functions should return quickly and require as little time as possible to execute. Within the callback function, counters can be used to determine where the callback function is taking too much time. For example, the VSync(1) function, which returns the number of horizontal lines drawn (HSyncs) since it was last called, can be used as a rough counter and is useful in finding sections of code that could be slowing down the callback function. (See below for timings of VSyncs and HSyncs.) Note that VSync(1) can be used within VSyncCallback() despite VSyncCallback() being a critical section.

Note that only a single interrupt can be stored and held pending at a given time. Thus, if two or more VSync interrupts occur during execution of a callback, only one VSync interrupt will be stored.

Timings of VSyncs and HSyncs:

VSyncs: 50 (PAL) or 60 (NTSC) vertical synchronization interrupts per second
HSyncs: 311 per VSync

Video Mode

The library function SetVideoMode() sets the current video signal mode to either PAL or NTSC. The default video signal mode for the PlayStation is NTSC, but the mode can be changed by calling SetVideoMode() prior to calling any other function.

The Application Environment

In this section, the memory map, stack pointer initialization and standard start-up routines that pertain to running applications are described.

Memory Map

The memory map as seen by a Net Yaroze application is shown below.

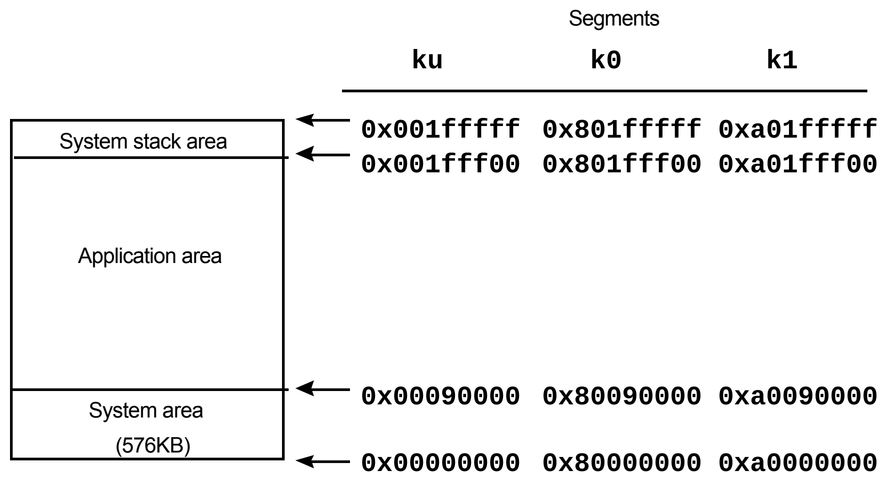

Segments
ku
k0
k1
System stack area
Application area
System area
(576KB)
0x001fffff 0x801fffff 0xa01fffff
0x001fff00 0x801fff00 0xa01fff00
0x00090000 0x80090000 0xa0090000
0x00000000 0x80000000 0xa0000000

Figure: Memory map

The OS itself ('System area' in the above diagram) is located at the low-address part of memory, and the system stack, which is the OS operating space, is located at the high-address part of memory. All other memory is available to the application.

Start-Up Routine

When an application is activated, a program called the start-up routine is executed prior to main(). The start-up routine performs a number of functions including initialization of global pointers and zeroing of external variables which do not have initial values.

The Net Yaroze standard start-up routine is provided as part of the Net Yaroze library.

Stack Pointer

The value of the application's stack pointer is inherited from that of the parent program. Where there is no explicit parent program, the application stack pointer value is inherited from the system stack.

4

The Net Yaroze Library

The Net Yaroze library provides services that make use of PlayStation features such as graphics, sound, Memory card and CD-ROM management. The library also provides standard C and math functions. This chapter gives a summary of each of these services.

Graphics Services

These library services make use of the graphics features of the PlayStation.

· **Frame Buffer Access**

Basic services for accessing the frame buffer and setting the drawing and display environments.

· **Integrated Graphics**
Various services relating to 2D and 3D graphics. Objects created with external tools can be controlled through these services.

Sound Services

These library services provide background playback of sound sequences which have previously been recorded as score data (MIDI-type data).

Standard Services

These library services provide support for standard C language functions.

· **Standard C Functions**
These are a subset of the standard C language library and include character functions, memory operation functions, character class tests, non-local jumps, and other utility functions.

· **Math Functions**
These functions conform to the ANSI/IEEE754 standard. They include a package which provides floating point arithmetic emulation in software.

Other Services

These library services provide support for PlayStation OS functions such as the API, CD-ROM management, and peripheral device management services

- **Kernel Management**
  An interface (API) between applications and the PlayStation OS.
- **CD-ROM Management**
  Services for reading image data, sound data and programs from the CD-ROM drive as well as playback of CD-DA (digital audio) sound.
- **Peripheral Device Management**
  Services for managing the peripheral devices such as the Controller and Memory card, and for managing callbacks for interrupt processing.

File Organization

The following is a description of files related to the Net Yaroze library.

- **Header Files**
  To use a service provided by the Net Yaroze library, you need to include a header file in the source code that defines the variables and functions used by that service.

  The following is a description of how Net Yaroze library header files are organized. All PlayStation functions are defined in the file 'libps.h'. Always include 'libps.h' when creating programs.

File	Contents	Notes
abs.h	abs()	included in stdlib.h
asm.h	R3000 register definitions	used with the assembler(*)
assert.h	assert()	
convert.h	atoi(), atol(), etc. (type conversion)	included in stdlib.h
ctype.h	isupper(), toupper(), etc. (type evaluation)	
fs.h	macro definitions	for internal use
libps.h	all services related to PlayStation functions	always include this
limits.h	C type limit macro definitions	
malloc.h	malloc(), etc.	included in stdlib.h
memory.h	memcpy(), etc. (memory operations)	included in strings.h
qsort.h	qsort()	included in stdlib.h
r3000.h	R3000 memory definitions	used with the assembler(*)
rand.h	rand(), srand(), etc. (random number generation)	included in stdlib.h
romio.h	-	for internal use
setjmp.h	setjmp(), longjmp(), etc. (jump over large areas)	
stdarg.h	va_start(), va_end(), etc. (variable number of arguments)	
stddef.h	type definitions	
stdio.h	standard I/O	
stdlib.h	standard functions	
string.h	strcpy(), etc. (character string operations)	identical to strings.h
strings.h	strcpy(), etc. (character string operations)	
sys/errno.h	errno and error definitions	
sys/fcntl.h	macro definitions	used by sys/file.h
sys/file.h	file I/O macro definitions	used by open(),close(), etc.
sys/ioctl.h	macro definitions	for internal use
sys/types.h	type definitions	

Table: Net Yaroze Header Files

* Refer to the Net Yaroze Web site for information related to assembler programming.

- **Library Files**
  The Net Yaroze library file, libps.a, is automatically linked when compiling so it does not need to be explicitly referred to.

Processing of Data

The Net Yaroze library can directly process graphics and sound data which are in the PlayStation format without modification. Data that is in PlayStation format can be created through conversion from standard graphics and sound file formats. (See Chapter 13, *Creating PlayStation Applications*, or the Net Yaroze Web site for more information.)

5

Frame Buffer Access

The frame buffer access services provide basic functions such as setting of the drawing and display environments, and transfer of image data to the frame buffer.

These services can be broadly divided into the following three groups.

(1) Environment-setting functions
(2) Frame-buffer-access functions
(3) Drawing-control functions

Description of the Frame Buffer

Pixels

From a software point of view, a frame buffer is a 1024 x 512 two-dimensional address space composed of 16-bit pixels. The top left pixel has the coordinates (0, 0) and the bottom right pixel has the coordinates (1023, 511). Each pixel is made up of three 5-bit data items and a semi-transparency flag. The data items each range from 0-31 and represent RGB brightness values.

The semi-transparency flag is only valid when the pixel is used as a texture.

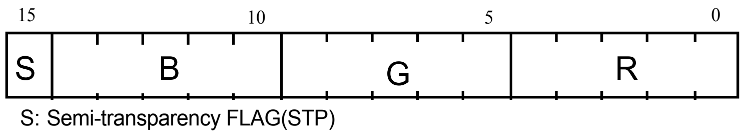


Figure: Pixel structure

Display Area

The part of the frame buffer that is displayed on-screen is a rectangular area known as the 'display area'. The size of the display area is determined by a GPU display function. The size can be selected as a pair of values ranging from 256 x 240 to 640 x 480 for NTSC and 256 x 256 to 640 x 512 for PAL. Interlace mode is on enabled when the display height is 480 for NTSC or 512 for PAL.

Drawing Area

Drawing is limited to a rectangular area, referred to as the 'drawing area' within the frame buffer. The drawing area may be any size that can be accommodated in the frame buffer.

Environment-Setting Functions

Information related to drawing as a whole, such as where drawing is to be carried out within the frame buffer and the starting point (offset) for drawing is referred to as the 'drawing environment'. In the same way, information related to display of the frame buffer, such as which part of the frame buffer to display, is referred to as the 'display environment'. The drawing environment is set up using the GsInitGraph() function, and the display environment is set up using the GsDefDispBuff() function.

GsInitGraph() takes the following arguments:

x_res Horizontal resolution (256/320/384/512/640)

Specifies the horizontal screen resolution as well as the width of the display and drawing areas. Drawing is clipped to the specified width.

y_res Vertical resolution (240/480) for NTSC and (256/512) for PAL.

Specifies the vertical screen resolution, and the height of the display and drawing areas. Drawing is clipped to the specified height.

intl Attributes

Interlaced display flag (bit 0)

Non-interlaced display is set when bit 0 is 0, and interlaced display when bit 0 is 1.

Set bit 0 to 1 when using a display height of 480 in NTSC or 512 in PAL.

Offset specification flag (bit 2)

When bit 2 is 0, the GTE offset feature is enabled. When bit 2 is 1, the GPU offset feature is enabled. The GPU offset feature is normally enabled.

dither	Dither processing flag (dtd) If this is set to 1, the drawing engine will perform dithering when drawing pixels.
vram	VRAM mode When VRAM mode is 0, 16 bits are displayed as 1 pixel. When VRAM mode is 1, 24 bits are displayed as 1 pixel. When VRAM mode is 1, only frame buffer access functions such as the LoadImage() function, are available. Other drawing functions, such as the GsDrawOt() function are only available when VRAM mode is 0.

GsDefDispBuff() takes the following arguments:

x0,y0	The top left coordinates of image buffer 0.
x1,y0	The top left coordinates of image buffer 1.

Image buffers '0' and '1' are rectangular areas in the frame buffer. The top left corners of the image buffers are (x0, y0) and (x1, y1), respectively. The width and height of the image buffers are specified with the x_res and y_res arguments of the GsInitGraph() function. One image buffer is used as the drawing area and the other as the display area. The buffers are swapped each time GsSwapDispBuff() is called which provides for the implementation of double buffering.

Frame-Buffer-Access Functions

The functions below allow data to be transferred within the frame buffer and between the frame buffer and the main memory. These functions are the only way to directly manipulate data in the frame buffer.

Function Name	Direction of Transmission
LoadImage()	main memory → frame buffer
StoreImage()	frame buffer → main memory
MoveImage()	frame buffer → frame buffer

Table: Frame-Buffer-Access Functions

These functions are non-blocking functions, that is to say, they terminate as soon as the access requests have been registered by the system. Up to 64 access requests may be registered (this number is the total for all three functions). These functions will block, i.e. will not terminate, until the access request has been registered.

Drawing-Control Functions

The functions below control drawing (specifically, the transfer of drawing commands to the GPU by the DMA Controller) and frame-buffer-access request processing.

Function Name	Action
GsDrawOT()	starts drawing
DrawSync()	waits for the termination of both drawing and the processing of frame-buffer-access requests, or returns request status.
ResetGraph()	stops drawing

Table: Drawing-Control Functions

Drawing Control in Non-Interlaced Mode

A simple rule controls drawing in non-interlaced mode: image buffers are not swapped until drawing completes.

After DrawSync(0) detects that drawing is complete, VSync(0) is executed. This delays the program until the next vertical synchronization interrupt. After the interrupt occurs, the image buffers are swapped using GsSwapDispBuff().

```
while (1) {
  ....
  DrawSync(0);
  VSync(0);
  GsSwapDispBuff();
  ....
}
```

Interlaced Mode

In interlaced mode, pixels with odd and even numbered vertical coordinates are displayed alternately every 1/60th of a second in NTSC, or every 1/50th of a second in PAL. This action forces automatic double buffering of the display.

This behavior, which applies solely to interlaced mode, can provide significant savings in frame buffer space since the display area and the drawing area are perfectly stacked on top of each other. However, the display area is always swapped every 1/60th of a second in NTSC or 1/50th of a second in PAL, regardless of how the program is executing. Therefore, the program must swap the image buffers at exactly the same time as the display area.

Drawing Control in Interlaced Mode

In interlaced mode, calculation and drawing must always be completed within 1/60th of a second (in NTSC) or 1/50th of a second (in PAL). Therefore, it is essential that buffer swapping after a vertical synchronization interrupt be performed regardless of whether drawing has completed. Thus, ResetGraph(1) is called following VSync(0) rather than using DrawSync().

```
while (1) {
  ....
  VSync(0);
  ResetGraph(1);
  GsSwapDispBuff();
  ....;
}
```

Kanji [Japanese Character] Fonts

The PlayStation is equipped with 16 dot x 16 dot kanji fonts stored as binary bit maps. You can use these to create messages that are easy to read.

Data Format	16-dot x 16-dot binary bit map / 15-dot x 15-dot kanji size
Contents	JIS Level 1 standard kanji/non-kanji/foreign language, gothic Non-kanji includes top space (0x2121)
Encoding	SHIFT-JIS

Table: Kanji Fonts

Font data is stored beginning with the upper left corner of the pattern and followed by the upper right byte, as shown in the figure below. Bits are ordered with the most significant bit (MSB) to the left.

#0	#1
#2	#3
#30	#31

Table: Font Data Format

The frame buffer access services include the Krom2Tim() function, which converts a SHIFT-JIS string to 4-bit TIM data, as well as character display functions that use font streams for debugging. For details refer to the *Library Reference Manual* and the SHIFT-JIS code and font data on the Net Yaroze PC CD under \psx\sample\sjiscode.

6

Integrated Graphics

The integrated graphics service operates on 3D and 2D graphics such as polygons, sprites and background surfaces.

The following functions are supported by the integrated graphics service.

1. Hierarchical coordinate systems
2. Light source calculation (3 parallel light sources, depth cueing, and ambient light)
3. Automatic object partitioning (polygon subdivision) and semi-transparency processing
4. Viewpoint management
5. Z-sorting
6. OT (ordering table) initialization, hierarchical organization and compression
7. Frame double buffering
8. Automatic aspect ratio adjustment
9. 2D clipping, offset processing
10. Sprites / backgrounds / lines

In addition, data created with all of the graphics tools can be handled without modification by the integrated graphics service.

The 3D model format used by the integrated graphics service is called 'TMD'. TMD format efficiently stores the information required to define a 3D model such as polygon vertices, surface normals and texture coordinates.

The 2D graphics data format used by the integrated graphics service is called 'TIM'. TIM format stores image resolution, number of colors and color lookup table (CLUT) information, as well as pixel data.

For details of the TMD and TIM formats refer to Chapter 15*, Graphics Tools*.

Drawing Process

The flow chart below shows the steps in the drawing process for the integrated graphics service.

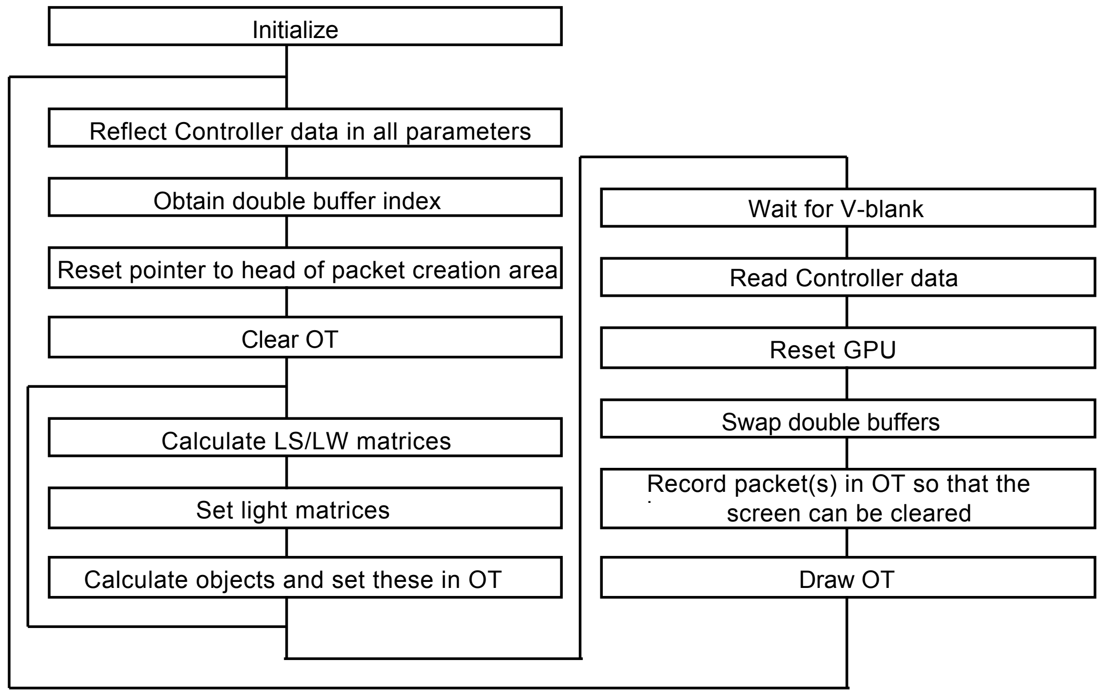

Initialize
Reflect Controller data in all parameters
Obtain double buffer index
Reset pointer to head of packet creation area
Clear OT
Calculate LS/LW matrices
Set light matrices
Calculate objects and set these in OT
Wait for V-blank
Read Controller data
Reset GPU
Swap double buffers
Record packet(s) in OT so that the screen can be cleared
Draw OT

Figure: Integrated Graphics Service Drawing Process

The drawing process sequence is described below.

1. Initialize the drawing and display environments and variables to be used.(GsInitGraph, GsDefDispBuf)
2. Set parameters to reflect Controller data.
3. Set up the viewpoint. (GsSetRefView2, GsSetView2)
4. Set up the packet creation working area. (GsSetWorkBase)
5. Clear the ordering table. (GsClearOt)
6. Calculate the LS/LW (Local Screen/Local World) matrices. (GsGetLs, GsGetLw)
7. Set up the LS/LW (Local Screen/Local World) matrices. (GsSetLightMatrix, GsSetLsMatrix)
8. After performing coordinate calculation, perspective transformation and light source calculation, enter the drawing packets in the ordering table. (GsSortObject4)
9. Wait for drawing to complete, then wait for V-blank. (DrawSync, VSync)
10. Swap the double buffers. (GsSwapDispBuff)
11. Draw the ordering table. (GsDrawOt)
12. Return to step 2.

* LS/LW refer to local/screen transformations and local/world transformations, respectively. These transformation matrices are used in perspective transformation and light source calculation. Please refer to *Coordinate Transformation & Light Source Calculation*, below.

Graphics System Initialization

The graphics system is initialized using the GsInitGraph() function. GsInitGraph() also sets the screen resolution and interlace mode. GsInitGraph() must be called before the integrated graphics service is used as it initializes various internal system variables.

The GsDefDispBuff() function is used to define the two rectangular areas of the frame buffer used for double buffering. In addition, GsDefDispBuff() initializes the clipping area and GPU offset coordinates to prevent drawing outside of the drawing area. The clipping area and GPU offset coordinates can be changed using the GsSetClip2D() and GsSetOrign() functions, respectively.

The GsSwapDispBuff() function is used to swap double buffers, while the GsGetActiveBuff() function can be used to determine which buffer contains the current drawing area.

The Viewpoint

A flat 2D TV screen cannot display 3D graphics directly. Instead, a 3D graphic must be drawn on a 2D screen so that it appears three dimensional. This is done by projecting 3D space onto a theoretical screen positioned in front of a particular viewpoint.

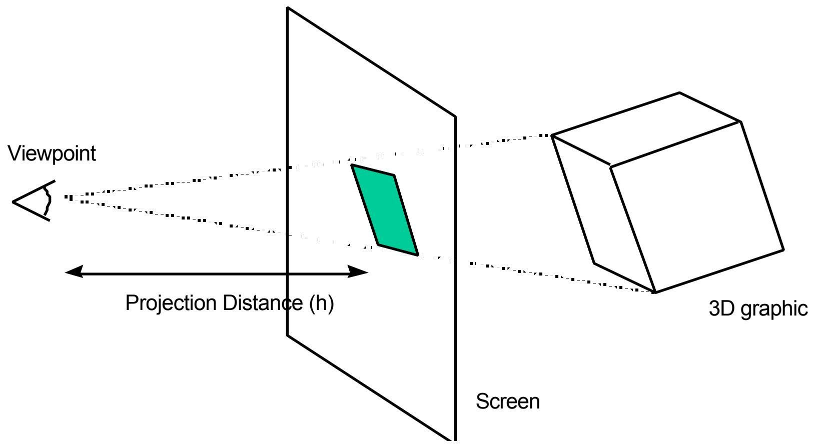


Figure: Projecting a 3D Image Onto a 2D Screen

Therefore, in order to project an image onto the physical screen, a viewpoint and a theoretical screen need to be specified.

Setting the viewpoint

The viewpoint is set by initializing the members of either a GsRVIEW2 or GsVIEW2 structure, and calling either the GsSetRefView2() or GsSetView2() function, respectively.

GsRVIEW2 and GsVIEW2 set the viewpoint using different methods. GsRVIEW2 sets the coordinates of the viewpoint and a reference point, while GsVIEW2 directly sets up a matrix for conversion to the viewpoint coordinate system.

You can set up a hierarchical coordinate system using GsVIEW2 or GsRVIEW2. For example, if the reference coordinate system is the world coordinate system, the result is an ordinary camera that captures an objective view. Alternatively if the reference coordinate system is the local coordinate system of an object, then the result is a camera that captures the subjective view of that object.

Setting the Theoretical Screen

The theoretical screen is positioned in front of the viewpoint. The GsSetProjection() function is used to set h, the projection distance, which is the distance from the viewpoint to the theoretical screen.

The Theoretical Screen

The height and width of the theoretical screen should match the resolution of the physical screen. For example, if the resolution of the physical screen is 640 x 480, the width of the theoretical screen should be set to 640 and the height should be set to 480.

When the ratio of the screen's horizontal resolution to its vertical resolution is 4 to 3, the dots on the screen will have a regular shape. If the actual screen resolution is such that the dots are irregular (that is, the ratio of horizontal to vertical resolution is not 4 to 3), the vertical resolution must be adjusted accordingly. For example, when the screen resolution is 640 x 240, objects should be displayed with one-half of their original vertical values so that the objects will appear as if their dots were regular in shape.

The Projection Distance

The projection distance is used to adjust the field of view. The larger the projection distance, the narrower is the field of view (this tends towards a parallel projection). On the other hand, the smaller the projection distance, the wider is the field of view. A wide field of view emphasizes perspective, creating the impression of close and distant objects from the viewpoint.

Packets

A 'packet' (or 'primitive') is the smallest unit of drawing commands handled by the GPU. Packets can be generated using the GsSortObject4(). A large work area must be allocated before using this function as packets can grow depending on the number and type of polygons. This work area is allocated using the

GsSetWorkBase() function, while the GsGetWorkBase() function can be used to determine how much of the work area has been used for packet generation.

Packets in the packet generation work area are used during drawing. Consequently, GsSortObject4() should not be used to access the work area during drawing. Instead, double buffering should be used to manage packet access by allocating two separate packet generation work areas.

Ordering Tables

Polygons will either be hidden or visible when 3D graphics are displayed. Polygons are hidden when they are obscured by other polygons that are in front of them along the Z axis.

An ordering table is a mechanism for controlling the proper display of polygons through a simple algorithm known as a Z-sort. Z-sorting ensures that polygons will be drawn in the correct order so that those in back will be obscured by those in front.

An ordering table (OT) can be thought of as a Z axis 'ruler'. Each scale mark on this ruler can hold as many polygons as needed.

Sorting is based on the average Z value of polygons, and is performed by placing polygons at the scale mark on the ruler corresponding to their average Z value. Polygons are transferred to the rendering chip in the order that they were placed along the ruler. Thus hidden surfaces are removed and polygons appear in the correct order as drawing progresses.

GsOT

A 'GsOT' structure is used to manage an ordering table (OT). The GsOT structure contains a pointer to the OT (the ‘org’ member) and a number of parameters indicating the attributes of the OT.

The resolution of the Z axis (i.e. the ruler markings) can be specified with the member 'length'. The length can be set to any one of 14 levels from 2^1 to 2^{14}.

GsOT_TAG

A GsOT_TAG represents a scale mark along the Z axis ruler. An OT is defined as an array of GsOT_TAGs with the number of elements of the array equal to 2^{length}. For example, if the member 'length' has a value of 4, the actual OT will be an array of 16 GsOT_TAGs ($16 = 2^4$).

OT Initialization

The GsClearOt() function is used to initialize an OT. GsClearOt() takes three arguments: 'offset', 'point' and 'otp'. 'otp' is a pointer to the OT handler. ('offset' and 'point' are described later.)

When an OT is initialized it is placed in an empty state with no polygons linked to it. An OT must be initialized each frame, prior to resorting.

Multiple OTs

The graphics service supports the use of multiple OTs. The GsSortOt() function can be used to merge and sort multiple OTs within one global OT. The typical Z value for the entire local OT is stored in the GsOT member 'point'. ‘Point’ is used when inserting and sorting the local OT into the global OT. Using multiple OTs allows the order of sorting to be controlled.

For example, you can sort in object units by preparing a local OT for each object. These local OTs are then sorted and merged together into a single global OT. This technique is extremely useful if the depth relationship between objects (i.e. which objects are in front and behind) is already known. (For instance: when you are looking down from a helicopter on cars that are driving along a road you already know that the helicopter is in front of all the cars.)

OT Compression

Using OTs increases the speed of sorting, but OTs consume a considerable amount of memory.

You can reduce an OT's memory consumption by making 'length' smaller. Doing this, though, reduces the sorting resolution, and can result in flickering polygons on screen.

Decreasing the amount of memory consumed by an OT without reducing the resolution is possible using an offset. Offsetting can be used when you know that the Z values of the polygons to be sorted are all higher than a certain value ('x').

Offsetting is implemented by setting the 'offset' parameter of GsClearOt() to the lowest value ('x'). With offsetting, the OT does not use up any memory for the part of the table below the offset value ('x'), thus reducing the amount of memory consumed.

Coordinate Transformation & Light Source Calculation

Coordinate Transformation

The PlayStation stores all the vertices of the objects to be displayed as 3D coordinates so the image of an object, as seen from different viewpoints, can be generated solely through calculation. This allows the game's viewpoint to be freely selected as either the player's viewpoint, or the viewpoint of one of the characters in a scene.

In order to display objects expressed in terms of 3D coordinates on a screen, 3D images need to be expressed as 2D images via a projective transformation. Projective transformation is performed by multiplying the 3D coordinate matrix by a 3 x 3 rotation matrix and adding a 3D translation vector. The result is then divided by the distance form the viewpoint in order to give perspective (so objects further away appear smaller).

For example, if the object to be displayed is described by the coordinate system (Xw, Yw, Zw), and (Xs, Ys, Zs) represents the coordinate system of the theoretical screen, the following equation (Equation 1) expresses the projective transformation calculation.

Equation 1

$$\begin{bmatrix} Xs \\ Ys \\ Zs \end{bmatrix} = \begin{bmatrix} SW11, SW12, SW13 \\ SW21, SW22, SW23 \\ SW31, SW32, SW33 \end{bmatrix} \begin{bmatrix} Xw \\ Yw \\ Zw \end{bmatrix} + \begin{bmatrix} SWx \\ SWy \\ SWz \end{bmatrix}$$

* Swij is the world/theoretical screen transformation matrix.

If you want to display more than one object, each of which is independently movable, you need to assign a coordinate system (Xl, Yl, Zl) to each independent object.

In the PlayStation the three types of coordinate systems are referred to as follows:

(Xl, Yl, Zl)	Local coordinate system (a coordinate system for a particular object and centered on that object)
(Xw, Yw, Zw)	World coordinate system (a coordinate system fixed to the world in which objects are placed)
(Xs, Ys, Zs)	Theoretical screen coordinate system (a coordinate system fixed to a theoretical screen and centered on a viewpoint)

Vertex coordinates are usually expressed in terms of the local coordinate system, so the following transformations are necessary in order to display an object on a theoretical screen.

local → world → screen

Equation 1 (above) and Equation 2 (below) together perform the required projective transformation calculation. Equation 3 shows the combined calculation together, while Equation 4 presents it in a reduced and simplified form.

Equation 2

$$\begin{bmatrix} Xw \\ Yw \\ Zw \end{bmatrix} = \begin{bmatrix} WL11, WL12, WL13 \\ WL21, WL22, WL23 \\ WL31, WL32, WL33 \end{bmatrix} \begin{bmatrix} XL \\ YL \\ ZL \end{bmatrix} + \begin{bmatrix} WLx \\ WLy \\ WLz \end{bmatrix}$$

Equation 3

$$\begin{bmatrix} Xs \\ Ys \\ Zs \end{bmatrix} = \begin{bmatrix} SW11, SW12, SW13 \\ SW21, SW22, SW23 \\ SW31, SW32, SW33 \end{bmatrix} \begin{bmatrix} WL11, WL12, WL13 \\ WL21, WL22, WL23 \\ WL31, WL32, WL33 \end{bmatrix} \begin{bmatrix} XL \\ YL \\ ZL \end{bmatrix}$$

$$+ \begin{bmatrix} SW11, SW12, SW13 \\ SW21, SW22, SW23 \\ SW31, SW32, Sw33 \end{bmatrix} \begin{bmatrix} WLx \\ WLy \\ WLz \end{bmatrix} + \begin{bmatrix} SWx \\ SWy \\ SWz \end{bmatrix}$$

Simplifying this equation gives:

Equation 4

$$\begin{bmatrix} Xs \\ Ys \\ Zs \end{bmatrix} = \begin{bmatrix} SL11, SL12, SL13 \\ SL21, SL22, SL23 \\ SL31, SL32, SL33 \end{bmatrix} \begin{bmatrix} XL \\ YL \\ ZL \end{bmatrix} + \begin{bmatrix} Trx \\ Try \\ Trz \end{bmatrix}$$

SLij and Tr can be determined using the GsGetLs() function. Also, as shown in Equation 5 (below), multiplying the theoretical screen coordinate values by h/Zs will apply a perspective transformation in which objects appear to be projected in parallel onto the theoretical screen (i.e. the image size will be scaled according to the distance from the viewpoint).

Equation 5

$$\text{Xss} = \text{Xs}\,\frac{h}{Zs}$$

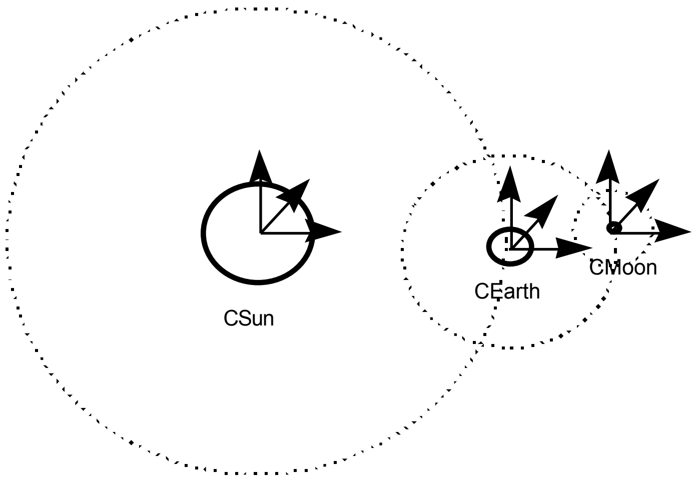
$$\text{Yss} = \text{Ys}\,\frac{h}{Zs}$$

* 'h' is the distance (projection) from the viewpoint to the theoretical screen.

The GsSortObject4() function takes TMD data (which only contains the local coordinate system) and creates the polygon drawing packet for a polygon projected onto a theoretical screen. This is done by specifying the matrix in Equation 2 in the 'coord' member of the GsCOORDINATE2 structure, the matrix in Equation 4 using the GsSetLsMatrix() function, and 'h' in Equation 5 using the GsSetProjection() function.

Hierarchical Coordinate Systems

Hierarchical coordinate systems involve the introduction of a layered tree structure into the coordinates of an object. Consider, for example, the case in which the earth is traveling in an orbit around the sun, and the moon is orbiting around the earth. If the sun is at the origin of the world coordinate system, it is far easier to describe the motion of the moon using an earth coordinate system which has the earth as its origin, rather than using the world coordinate system centered on the sun. In this case, the best approach is to set up a pointer to the earth's coordinate system in the 'super' member of the GsCOORDINATE2 structure which the moon's object handler would refer to. This would indicate that the moon's coordinate system is set based on the earth's coordinate system (i.e. the moon's coordinate system is under the earth's coordinate system). See diagram below.



```
CSun {
       coord  ...
       super  WORLD;
}
CEarth {
       coord  ... /* describes orbital motion around the sun */
       super  CSun;
}
CMoon {
       coord  ... /* describes orbital motion around the earth */
       super  CEarth;
}
```

Figure: A Hierarchical Coordinate System

Light Source Calculation

The light source calculation is used to determine the contribution of light from each light source within the world. The PlayStation supports the following three methods for light source calculation:

(Note that the normal line vector or 'normal' is an imaginary line perpendicular to either the surface of a polygon or the polygon vertices (the points that define the position of the polygon). Similarly, the light source vector is an imaginary line that defines where the light in an on-screen scene is coming from.)

- Flat shading
  In this method each polygon has a single normal line vector from its surface (the 'flat' or 'face' normal). The color and brightness is determined by the inner-product (measure of the angle) between the face normal and light-source vector.

- Gouraud shading
  In this method each polygon vertex (each point that is a 'corner' of the polygon) has its own normal line vector. The color and brightness at each vertex is determined by the inner product of the vertex normal and the light source. The color and brightness at each vertex is then interpolated to approximate the light source contribution across the polygon. This produces a smooth graduation of color and brightness across the polygon.

- Depth cueing
  This method varies the color and brightness of the polygon depending on the distance from the viewpoint. This achieves a fogging effect by making the color of a polygon closer to the background color the further away it is from the viewpoint.

The following figure describes the model on which light source calculations are based.

Figure: Light Source Calculation

In this figure, point P represents a point on the surface of an object. To calculate the light source contribution at point P the following terms are used:

(Nx, Ny, Nz)	= The normal line vector at point P.
(R, G, B)	= The inherent color of the object.
(Lx, Ly, Lz)	= The light source vector.
(LR, LG, LB)	= The light source color.
(RBK, GBK, BBK)	= The background (ambient) light.

The following is an example of the light source calculation sequence in the PlayStation if three light sources are used. (RR, GG, BB) represents the final color values used when point P is displayed.

(1) Convert coordinates of the normal line vector into the world coordinate system.

$$\begin{bmatrix} NWx \\ NWy \\ NWz \end{bmatrix} = \begin{bmatrix} WL11, WL12, WL13 \\ WL21, WL22, WL23 \\ WL31, WL32, WL33 \end{bmatrix} \begin{bmatrix} Nx \\ Ny \\ Nz \end{bmatrix}$$

(2) Obtain the inner product of the light source vector and the normal line vector.

Normal line vector (world) · Light source vector → Light source influence (L)

$$\begin{bmatrix} L1 \\ L2 \\ L3 \end{bmatrix} = \begin{bmatrix} Lx1, Ly1, Lz1 \\ Lx2, Ly2, Lz2 \\ Lx3, Ly3, Lz3 \end{bmatrix} \begin{bmatrix} NWx \\ NWy \\ NWz \end{bmatrix}$$

(3) Obtain the light source influence color by multiplying the inner product and the light source color separately for each item.

Light source influence (L) x Light source color (Lr, Lg, Lb) → Light source influence color (LI)

$$\begin{bmatrix} LIr \\ LIg \\ LIb \end{bmatrix} = \begin{bmatrix} Lr1, Lr2, Lr3 \\ Lg1, Lg2, Lg3 \\ Lb1, Lb2, Lb3 \end{bmatrix} \begin{bmatrix} L1 \\ L2 \\ L3 \end{bmatrix}$$

(4) Obtain the total influence color from the environment by adding together the light source influence color and the ambient color.

Light source influence color (LI) + Ambient color (BK) → influence color (LT)

$$\begin{bmatrix} LTr \\ LTg \\ LTb \end{bmatrix} = \begin{bmatrix} LIr \\ LIg \\ LIb \end{bmatrix} + \begin{bmatrix} BKr \\ BKg \\ BKb \end{bmatrix}$$

(5) Obtain the theoretical screen color by multiplying the inherent color with the influence color for each item separately.

$$\begin{bmatrix} RR \\ GG \\ BB \end{bmatrix} = \begin{bmatrix} R & LTr \\ G & LTg \\ B & LTb \end{bmatrix}$$

Use GsGetLw() to calculate the local world matrix. In order for this to be applied (as in the Wlij matrix in Equation 1, above) set it internally using the GsSetLightMatrix() function.

Set the light source vector and the light source using the GsSetFlatLight() function, and the ambient color using the GsSetAmbient() function.

When all the above are set, use the GsSortObject4() function to perform the light source calculation. GsSortObject4() will perform the calculation in accordance with the equations given above.

Creating Packets

The GsSortObject4() function performs coordinate calculation, perspective transformation and light source calculation on polygons defined within TMD data. The results are converted into drawing packets which are then added to an ordering table.

The GsSortObject4() function uses the GsDOBJ2 structure to handle object data (TMD data). In order for the GsDOBJ2 structure to handle TMD data, the TMD data must be mapped to a real address using the GsMapModelingData() function then linked to GsDOBJ2 using the GsLinkObject4() function.

The GsSortObject4() function performs coordinate transformation using the local screen matrix set with the GsSetLsMatrix() function. It then performs perspective transformation using the projection distance set with the GsSetProjection() function. Next, the GsSortObject4() function performs light source calculation using the local world light matrix set with the GsSetLightMatrix() function. Finally, the GsSortObject4() function creates a drawing packet and using the Z values determined during coordinate transformation, links the drawing packet to an ordering table.

The GsDrawOt() function can be used to draw a drawing packet that has been linked to an ordering table. GsDrawOt() returns immediately after execution and drawing is performed in the background. Subsequently, the DrawSync() function can be used to detect when drawing is complete.

Objects

The graphics service provides the GsDOBJ2 structure to handle objects (TMD data). The ‘coord2’ and ‘attribute’ members of GsDOBJ2 are used when operating on objects.

'coord2' is a pointer to a coordinate system GsCOORDINATE2. The position and orientation of the object are controlled by setting the parameters for GsCOORDINATE2. 'attribute' is used to set the object's attributes. The following lists the attributes that can be set.

Light Source Mode

This attribute determines the method of light source calculation.

Light Source Calculation OFF

This attribute disables light source calculation.

Inhibit Display

This attribute disables packet creation for the entire object.

Automatic Partitioning (polygon subdivision)

When enabled, this attribute causes all the polygons contained within the object to subdivide. The number of subdivisions can be 2 x 2, 4 x 4, 8 x 8, 16 x 16, 32 x 32 or 64 x 64. Subdividing polygons in this way reduces texture distortion and polygon fragmentation.

7

Sound

The sound service consists of functions that automatically playback sound data such as background music and special effects. Sound service functions are classified into one of the following four groups.

1. SEQ access functions. Functions that handle score data (i.e. SEQ data).
2. Individual voice-setting functions. Functions that handle individual sounds, including single-shot sounds and sound effects.
3. Shared-attribute setting functions. Functions that perform settings necessary to activate the sound service. These functions also set attributes that are shared by all voices of the SPU.
4. Sound utility functions. Functions that change attribute tables within VAB data at runtime. These functions also apply sound effects to an allocated voice after a 'KeyOn' is issued.

Score Data

Within the sound service, score data is represented in the SEQ data format.

The SEQ Data Format

The SEQ data format is a Format-1 SMF (Standard Midi File) converted for PlayStation use. SEQ is a format in which all track and chunk data of the MIDI data structure is merged in time order

In the SEQ format, sixteen songs can be played simultaneously. Notes are expressed in the form of status, data and delta time. Status is 1 byte in length, whereas the data length is determined by the status byte, and delta time length is variable up to a maximum of 4 bytes.

The SEQ format uses running status, where Note-Offs are converted to Note-Ons with zero velocity. The SEQ format supports the following types of status data as defined within the MIDI standard.

- note on
- note off
- program change
- pitch bend

The following items are included from 'control change'.

- data entry(6)
- main volume(7)
- panpot(10)
- expression(11)
- nrpn data(98,99)

Note: The numbers inside parentheses are control numbers.

MIDI Support

Setting VAB Attribute Data Using Control Change

NRPN data is defined so that VAB attribute data can be set from the MIDI standard Control Change NRPN.

When using a sequencer to create an SMF file, VAB attribute data can be set by sending the following sequence.

bnH	99	data1	(NRPN MSB)
bnH	98	data2	(NRPN LSB)
bnH	06	data3	(Data Entry)

Using Control Change to Set Repeat Loops Within Music

NRPN data can be used to implement a loop, or repeat function, for sections within music. Any number of repetitions between 0 and 126 can be specified (specifying 127 will cause continuous playback).

The repetition symbol '||:' identifies Loop1 (the start of the loop) and the symbol ':||' identifies Loop2 (the end of the loop). Although the repeat function can be used any number of times within one piece of music, it is not possible to embed a loop within a loop, such as (Loop1...(Loop1'...Loop2')...Loop2).

Attribute	data1(CC 99)	data2(CC 06)
Loop1(start)	20	0 127
Loop2(end)	30	

Table: Repetition using Control Change

Caution

No MIDI event that needs to take place before another MIDI event should be placed at the same “tick” as that event. Depending on the sequencer, this situation could produce invalid output when the data is converted to SMF.

If identical delta times are used, the order of the data may switch incorrectly during conversion and invalid data may be generated, even if the data was originally entered in the correct order.

For example, if a patch change, a controller 99, a controller 6 and the first key on of the sequence all appear at 1/1/000 in the event list, there is no guarantee that the order will be preserved on conversion.

Note

When setting VAB attribute data, values become valid after Key On following Data Entry read.

Caution

The setting of a repeat loop should be done only in one place within the SEQ data for a given sequence. It is not necessary to set the repeat loop for each channel.

Waveform Data

Waveform data is described by the VAB data format.

VAB Format

VAB is a data format used for waveform management. Sampled sounds such as that from a piano or an explosion are compressed and encoded in a format known as VAG. The VAB format is a collection of these individual VAG-formatted sounds. VAB-formatted data is organized in files for use at runtime.

A VAB file contains all of the sampled data and sound file attributes used in a particular scene. Multi-timbral sound is supported with a voice priority protocol.

A single VAB file can contain up to 254 VAG sounds and 128 programs with each program having up to 16 tone lists.

Individual waveforms can be referenced in more than one tone list so a single waveform can be played back in many different ways.

The PlayStation allows up to 16 VABs to be used simultaneously.

For further information on data formats, please refer to the Net Yaroze Web site.

Function Execution Order

The following illustrates the order of operations when using sound service functions.

1. Open SEQ data

   Execute the SsSeqOpen() function.

2. Perform desired operations

   After setting the main volume, perform the desired operations.

3. Close SEQ data

. Execute the SsSeqClose() function.

8

Standard C Functions

Net Yaroze provides a set of headers and functions that are broadly similar to standard C (as defined in *The C Programming Language* by Kernighan & Ritchie).

Include Headers

File	Contents	Notes
abs.h	abs()	included in stdlib.h
assert.h	assert()	
convert.h	atoi(), atol(), etc. (type conversion)	included in stdlib.h
ctype.h	isupper(), toupper(), etc. (type evaluation)	
fs.h	macro definitions	for internal use only
limits.h	C type limit macro definitions	
malloc.h	malloc(), etc.	included in stdlib.h
memory.h	memcpy(), etc. (memory operations)	included in strings.h
qsort.h	qsort()	included in stdlib.h
rand.h	rand(), srand(), etc. (random number generation)	included in stdlib.h
setjmp.h	setjmp(), longjmp(), etc. (jump over large areas)	
stdarg.h	va_start(), va_end(), etc. (variable arguments)	
stddef.h	type definitions	
stdio.h	standard I/O	
stdlib.h	standard functions	
string.h	strcpy(), etc. (character string operations)	identical to strings.h
strings.h	strcpy(), etc. (character string operations)	
sys/errno.h	errno and error definitions	
sys/fcntl.h	macro definitions	used by sys/file.h
sys/file.h	file I/O macro definitions	used by open(),close(), etc.
sys/ioctl.h	macro definitions	for internal use only
sys/types.h	type definitions	

Supported Functions

The following standard C functions are supported.

Function	Description
abs	calculate absolute value
atoi	convert character string to integer
atol	convert character string to long integer
bcmp	compare memory blocks
bcopy	copy memory blocks
bsearch	perform binary search
bzero	write zeros to memory block
calloc	allocate main memory
exit	perform normal termination
free	free allocated memory blocks
getc	read one character from a stream
getchar	read one character from the standard input stream
gets	read a character string from the standard input stream
isXXX	perform test on character
labs	calculate absolute value of long
malloc	allocate memory
memchr	find character in memory block
memcmp	compare memory blocks
memcpy	copy memory blocks
memmove	copy memory blocks
memset	writes specified characters to memory blocks
printf	send formatted output to standard output ('*stdout*')
putc	outputs a single character to a specified stream
putchar	outputs a single character to *stdout*
puts	outputs a character string to *stdout*
qsort	perform quicksort
rand	generate random numbers

realloc	reallocate heap memory
srand	initialize random number generator
strcat	concatenate one character string to another
strchr	search for the position at which a specified character appears in a character string
strcmp	compare two character strings
strcpy	copy a character string
strcspn	return the length of the first part of a character string that is composed entirely of characters not included in a specified collection of characters
strlen	find the number of characters in a character string
strncat	concatenate a character string of a specified length to another character string
strncmp	compare two character strings with each other
strncpy	copy character strings of a specified length
strpbrk	find the position at which a specified character first appears in a character string
strrchr	find the position at which a substring first appears in a character string
strspn	find the first part of a character string that contains only characters that occur within a specified subset
strstr	find the position where a substring occurs in a string
strtok	find a character string delimited by characters from a specified character set
strtol	convert a character string into a long
strtoul	convert a character string into an unsigned long
toascii	mask bit 7 from an input value
tolower	convert characters to lower case
toupper	convert characters to upper case

9

Math Functions

Floating-point Numbers

The Net Yaroze system includes a standard set of C math functions that support IEEE 754 standard single-precision floating-point numbers ('float') and double-precision floating-point numbers ('double').

Because the PlayStation version of the R3000 CPU does not possess a floating-point arithmetic coprocessor, the floating-point arithmetic package is implemented entirely in software.

Attribute	Specification
Size	4 bytes
No. of digits available	6 digits (decimal number conversion)
Overflow limit value (Largest number)	2.0^{128} = 3.4e38 *
Underflow limit value (Smallest Number)	0.5^{126} = 2.2e-38 *

Table: 'Float' DataType

Attribute	Specification
Size	8 bytes
No. of digits available	15 digits (decimal number conversion)
Overflow limit value (Largest Number)	2.0^{1024} = 1.8e308 *
Underflow limit value (Smallest Number)	0.5^{1022} = 2.2e-308 *

Table: 'Double' Data Type

* *Note:* In scientific notation, 'e' means '10 to the power of' so, 2.2e-308 is 2.2×10^{-308}

Error Handling

Errors related to floating-point arithmetic are reported via 'events' in addition to the normal C method of setting external variables (such as 'math_errno').

Error Types

There are two types of errors that can be reported during floating point operations: ‘domain errors’ and ‘range errors’.

A function tests the range of its arguments to ensure they fit in the appropriate range. A 'domain error' is reported if the argument value of a function is out of range, when the allowed range is stated explicitly in the function definition

A ‘range error’ is reported when a function performs a calculation, or an application uses an arithmetic operator, that produces a value that exceeds the range of the data type.

Internal Processing when Errors Occur

When a function reports a 'domain′ or 'range' error, an external variable is set with the appropriate error code. If the error occurred during an arithmetic calculation, the result of the operation is set to a signed infinite number so that calculation can continue as long as possible (see table, below).

A function returns a NaN (Not a Number) when a division by zero error occurs.

	Float Data Type	Double Data Type
Positive infinity	0x7F800000	0x7FF0000000000000
Negative infinity	0xFF800000	0xFFF0000000000000
Positive NaN	0x7FFFFFFF	0x7FFFFFFFFFFFFFFF
Negative NaN	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF

(Note that NaN is a bit pattern reserved so that the arithmetic subroutine can report an error to the system. The value cannot be assigned to variables.)

Error Variables

The math functions use an external variable, *math_errno* to indicate an error code. An 'extern' declaration can be found in the *libps.h* file for this variable.

The variable *math_errno* is initially set to zero. When an error occurs, *math_errno* is set to the value indicated by the macros EDOM or ERANGE (both of which are defined in the sys/errno.h file), depending on the type of error. Note that *math_errno* does not automatically reset itself, so you should explicitly set it to zero once your error handling is complete.

Supported Functions

Function	Description
pow	x^y - *X* to the power of *Y*
exp	exponential
log	$\ln(x)$ - natural logarithm of *X*
log10	$\log_{10}(x)$ - base 10 logarithm
floor	largest integer not greater than *X*
ceil	smallest integer not smaller than *X*
fmod	floating-point remainder of ***x/y*,** with the same sign as *X*
modf	splits *X* into integral and fractional parts
sin	sine
cos	cosine
tan	tangent
asin	$\sin^{-1}(x)$ - arcsine
acos	$\cos^{-1}(x)$ - arccosine
atan	$\tan^{-1}(x)$ - arctangent
atan2	$\tan^{-1}(y/x)$ - arctangent
sinh	hyperbolic sine
cosh	hyperbolic cosine
tanh	hyperbolic tangent
sqrt	square root
hypot	absolute value of a complex number
ldexp	$X * 2^n$
frexp	splits **X** into a normalized fraction in the interval [_,1] (separation into mantissa and exponent)
fabs	absolute value (macro)
printf2	formatted output to the console (supports 'float' and 'double' type arguments)
sprintf2	formatted output to an array (supports 'float' and 'double' type arguments)

10

Kernel Management

Kernel management services provide basic functions for controlling the PlayStation hardware and the CPU. Kernel management services are implemented as a set of C functions which execute jump instructions that branch directly into the kernel.

Kernel management services are classified as follows:

- Root counter management
- I/O management
- Module management
- Additional services

Root Counter Management

Root counter management services are critical for game programs as they provide important functions like time period management and timing adjustment. The root counter is a single 16-bit counter that increments every 8 ticks of the system clock (about 0.24 microseconds). The value of the root counter can be obtained with the GetRCnt() function.

Up-Counting

The root counter is incremented by hardware. The root counter will continue incrementing even if interrupts are disabled by software.

Target Values

A target value can be set for the root counter. When the value of the root counter reaches the target value, the root counter is cleared and continues incrementing.

Macros

RCntCNT2 is the name of the root counter that is available in the Net Yaroze system. The RCntCNT2 macro is used within root counter functions such as SetRCnt() to specify this root counter. The operating mode of the root counter is specified as an argument to SetRCnt() and must be set to (RCntMdNOINTR | RCntMdSP). No other operating mode is valid for Net Yaroze.

Start-Up State

The state of the root counter is unknown at start-up. The root counter must be initialized with StartRCnt() before it is used.

I/O Management

The I/O management service provides support for file I/O. The data structures and macros used in the I/O management service are defined in the file 'sys/file.h'.

The I/O management service is responsible for all PlayStation file management. General file access functions such as open() and close() are provided as well as other functions such as file search (firstfile(), nextfile()).

Block Size

Each device has its own ‘block size’ which serves as its unit of data access.

All data access operations are performed in multiples of the block size. Data which exceeds an integral number of blocks will be ignored.

The Standard I/O Stream

File descriptors 0 and 1 are used for the standard input and standard output streams, respectively.

On start-up, Net Yaroze assigns the standard input/output streams to the ‘tty’ serial device. The serial device performs input and output of characters via the serial interface.

The Memory Card Driver (bu)

Memory cards are shared across multiple applications. Therefore, access to Memory cards is permitted only through the file system. The bu device is the Memory card driver and supports functions such as file search.

A description of the 'bu' device is given in the table below.

Item	Description
Device Name	bu
Physical devices	bu00: Memory card in slot 1 bu10: Memory card in slot 2
Functions supported	open, close, read, write, firstfile, nextfile, delete, rename, format
Block size	128 bytes

Table: The Memory Card Driver

Item	Description
Filename	buXY:filename 'filename' is a 21 character ASCII character string A colon (':') is placed between 'buXY' and 'filename' 'XY' specifies the insertion slot Valid characters are the upper case English letters, numbers and the character '_'.
Directory	not used
Number of files	Between 1 and 15 for each card

File size	Multiples of 8 KB
	Specified at file creation. No automatic expansion using write().
	Example: (this creates a 24K file)
	`long size = 3;`
	`long fd = open(fname, O_CREAT | (size<<16));`
	`close(fd); /* always close immediately after creation */`

Table: The Memory Card File System

The Serial Driver (tty)

The serial driver provides the basic link between the host PC and the Net Yaroze PlayStation. The transfer speed of the serial driver can be set with the ioctl() function.

A description of the serial driver is given in the table below.

Item	Description
Device name	tty
Physical device	serial interface
Functions supported	open, close, read, write, ioctl
Block size	1 byte

Table: The Serial Driver

Module Management Service

The module management service provides basic functions for loading and executing application modules.

Executable Files

The PlayStation executable file format is known as 'PS-X EXE'. All applications that run on the PlayStation must conform to this format. Executable files contain the information listed below.

Information within Executable Files

(a) Code and data linked to fixed addresses

(b) The execution start address

(c) gp register initial value

(d) The starting address and size of the data area for uninitialized data

Layout within Executable Files

Offset from Start of File	Block name	Contents	Size
0 bytes	header	(b), (c) & (d) from the list above	2048 bytes
2048	body of the program	(a) from the list above text section + data section for initialized data	multiple of 2048 bytes

Table: Layout within Executable Files

Executable File Information Data Structure

The functions that operate on executable files (Load() and Exec()) use the internal information described above. The EXEC structure can be used to access this information directly.

EXEC has the following structure.

```
struct EXEC {                                 /* executable file information */
        unsigned long pc0;        /* execution start address */
        unsigned long gp0;        /* gp register initial value */
        unsigned long t_addr;     /* starting address of text section
                                   and initialized data section */
        unsigned long t_size;     /* size of text section
                                  + size of initialized data section */
        unsigned long d_addr;     /* reserved for system use */
        unsigned long d_size;     /* reserved for system use */
        unsigned long b_addr;     /* starting address of the data section for
                                  uninitialized data */
        unsigned long b_size;     /* size of the data section for
                                  uninitialized data */
        unsigned long s_addr;     /* stack area starting address
                                  (user-specified) */
        unsigned long s_size;     /* stack area size (user-specified)*/
        unsigned long sp,fp,gp,ret,base;  /* register save area */
};
```

Loading Executable Files

Executable files are loaded from the CD-ROM using CdReadExec(). Loaded files are executed with Exec().

Additional Services

The following additional services are provided for kernel management.

· InitHeap()
  InitHeap() initializes the heap area. The heap area should be initialized before using the memory allocation function, malloc(). However, InitHeap() should be called explicitly only when the heap area is changed as it is usually called automatically prior to main().

- FlushCache()

  FlushCache() flushes the R3000 I-cache. FlushCache() should always be called between loading an executable file (using CdReadExec(), for example) and executing the file with Exec().

The Heap

The heap is an area in memory used for storage allocation . It is dynamically controlled using malloc() and free().

The heap should be allocated a fixed amount of memory from inside the start-up routine according to the size of the application program.

11

CD-ROM Management

The CD-ROM management service includes functions that support reading files from the CD-ROM and playing CD-DA data.

CD-ROM Overview

Sectors

Digital data is recorded in a spiral pattern on the surface of a CD-ROM, just as on an audio CD. The digital data on a CD-ROM is processed in units known as ‘sectors.’ It takes 75 sectors to equal 1 second’s worth of digital data.

Each sector can be classified as an audio sector, a data sector, or an ADPCM sector, depending on how it is used.

Audio Sectors

Data stored in audio sectors is recorded as digital stereo audio data at a sampling frequency (f_s) of 44.1 KHz,. This is the normal sampling frequency for CD audio data. Audio sectors can be played back with the CdPlay() function, but audio sectors cannot be read as user data.

Data Sectors

User data is recorded in data sectors. The amount of user data which can be recorded in a data sector differs slightly depending on the mode, but generally, 2048 bytes are available (mode-1 format).

ADPCM Sectors

ADPCM sectors are more properly called 'real time sectors' or 'mode-2 form-2' sectors. ADPCM sectors store audio data compressed using ADPCM compression. ADPCM sectors are typically used to play back audio in the same way as audio sectors. However, ADPCM sectors are not supported by Net Yaroze.

Transfer Rate

The PlayStation CD-ROM can rotate at either standard speed or double speed.

Standard speed provides the same rotation rate as a standard CD player. The transfer rate for standard speed is 150 KB/sec, while the transfer rate for double speed is 300 KB/sec.

This means that in one second, 75 sectors of data can be read at standard speed and 150 sectors can be read at double speed.

The File System

The PlayStation CD-ROM uses a file system based on ISO-9660 level 1. The details of the file system are shown in the table below.

Item	Description
File format	basename.ext;version 'basename' can be up to 8 characters, 'ext' can be up to 3 characters. A '.' (period) is placed between 'basename' and 'ext'. A ';' (semicolon) is placed between 'ext' and 'version'. Valid characters are the upper case English letters, numbers and the character '_'.
Directory format	basename 'basename' can be up to 8 characters. No extensions are allowed. Valid characters are the upper case English letters, numbers and the character '_'.
Directory levels	Maximum of 8 levels. The root directory has no name.
File arrangement	All sectors in a file are physically arranged contiguously.
Block size	2 KB

Table: The CD-ROM File System (ISO-9660 level 1 standard)

However, the PlayStation only supports lists of files and directories that can be stored in one sector (2048 bytes). As a result, there are certain restrictions specific to the PlayStation as shown in the table below.

Total number of directories	up to 45
Total no. of files per directory	up to 30

Table: CD-ROM Restrictions Specific to the PlayStation

* The file and directory control data structure in ISO-9660 is of variable length. In cases where many of the names are short, it is possible to have more directories and files than what is shown in the table above.

File Access

File Searching

The CdSearchFile() function is used to find the starting position of a file in the ISO-9660 file system. CdSearchFile() searches for the start of a file using the absolute path name of the file. The search results are saved in the Cd1FILE structure, which is shown below.

```
typedef struct {
    CdlLOC pos;          /* file position */
    unsigned long size;  /* file size */
    char   name[16];     /* file name (body) */
} CdlFILE;
```

In addition, the CdlLOC structure, which indicates the file position, has the following structure.

```
typedef struct {
    unsigned char minute; /* minute (BCD) */
    unsigned char second; /* second (BCD) */
    unsigned char sector; /* sector (BCD) */
    unsigned char track;  /* track (void) */
} CdlLOC;
```

Reading Data Files

The CdReadFile() function is used to read a data file from the CD-ROM. The following is an example of how CdReadFile() is used.

```
CdlFILE fp;
if (CdSearchFile(&fp, "\PSX\SAMPLE\RCUBE.TIM") != 0)
    CdReadFile(fp, sectbuf, 2048);
```

Note that CdReadFile() executes asynchronously as a non-blocking function and returns immediately after it is called. CdReadSync() is used to detect the actual completion of execution.

Reading Executable Files

The CdReadExec() function is used to load an executable file from the CD-ROM into memory. The file can be executed with the Exec() function provided by the kernel management service.

Note that CdReadExec() executes asynchronously as a non-blocking function and returns immediately after it is called. CdReadSync() can be used to determine the number of unread sectors remaining.

Read Synchronization

The CdReadFile() and CdReadExec() functions execute asynchronously as non-blocking functions that return immediately after they are called. CdReadSync() is used to detect the completion of these functions and to determine the number of unread sectors remaining.

Playback of CD-DA Audio Data

The CdPlay() function is used to play back audio data recorded in CD-DA audio sectors.

12

Peripheral Devices Management

The peripheral devices service manages standard Net Yaroze PlayStation peripherals such as Controllers and Memory cards.

Controller Management

Controllers are important devices that communicate the player's commands to the application. Two Controllers can be connected to the PlayStation. Four types of Controllers are supported by the Net Yaroze PlayStation: the standard Controller, a mouse, a Namco neGcon analog controller, and an analog joystick.

Reading Data From the Controllers

The GetPadBuf() function can be used to read data from the Controllers.

Communication with the Controllers takes place every vertical synchronization interrupt (VSync). The data read from the Controllers is saved in two internal system buffers, one for each Controller. Pointers to these buffers can be obtained using GetPadBuf().

The table below shows the data which is stored in the two Controller buffers.

Bytes from the Start of the Buffer	Content
0	0xff: no Controller, 0x00: Controller connected
1	Upper 4 bits: terminal type Lower 4 bits: size of received data (1/2 the number of bytes)
2~	Received data (maximum 32 bytes)

Table: Controller Receive Buffer Data Format (1)

Controller Types and Data Received

The Net Yaroze library supports the standard Controller, the Mouse, the Namco neGcon analog controller and the analog joystick.

GetPadBuf() can be used to obtain the contents of the Controller buffer. The contents of the buffer are different for each type of Controller as shown in the table below.

Terminal Type	Device Name	Byte Number	Content
0x1	Mouse	Byte 2	Not used
		Byte 3	Bit 2: right button Bit 3: left button Bit values: 1 - button released, 0 - button pushed
		Byte 4	Displacement in X direction -128 to +127
		Byte 5	Displacement in Y direction -128 to +127
		Bytes 6-7	Not used
0x2	neGcon Controller	Bytes 2-3 (a single 16-bit group)	Bit values: 1 - button released, 0 - button pushed *Refer to the Net Yaroze Web site for button bit locations.*
		Byte 4	Analog channel value -128 to +127
		Bytes 5-7	Analog channel values 0 to 255
0x4	Standard Controller	Bytes 2-3 (a single 16-bit group)	Bit values: 1 - button released, 0 - button pushed *Refer to the figure, below for button bit locations.*
0x5	Analog Joystick	Bytes 2-3 (a single 16-bit group)	Bit values: 1 - button released, 0 - button pushed *Refer to the Net Yaroze Web site for button bit locations*
		Bytes 4-7	Analog channel value 0 to 255

Table: Receive Buffer Data Format (2)

The button and bit assignments for the Standard Controller are shown in the figure and table below.

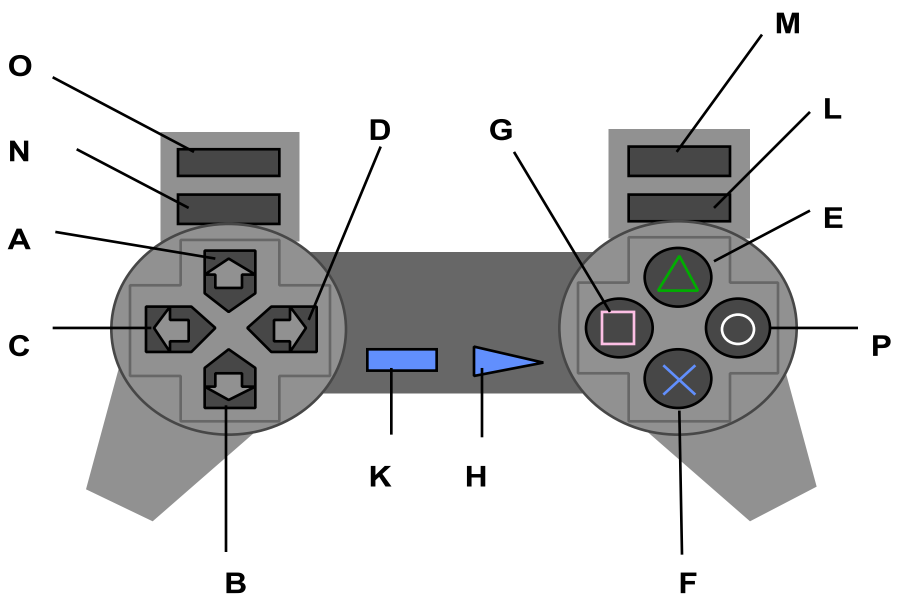


Figure: Standard Controller Button Assignments

Bit No.	Corresponding Button	Macro
15	C	PADLleft
14	B	PADLdown
13	D	PADLright
12	A	PADLup
11	H	PADstart
10		
9		
8	K	PADselect
7	G	PADRleft
6	F	PADRdown
5	P	PADRright
4	E	PADRup
3	L	PADR1
2	M	PADR2
1	N	PADL1
0	O	PADL2

Table: Standard Controller Bit Assignments

* The macros in this table are defined in the file ‘pad.h’ in the check program (\PSX\SAMPLE\CHECK).

Memory Card Management

The Net Yaroze library supports the function TestCard(), which tests whether a Memory card is inserted in the PlayStation. For details, please refer to the *Library Reference* manual.

13

Creating PlayStation Applications

This chapter presents an overview of how a PlayStation application is created using the Net Yaroze system.

An application is composed of data and code.

- Code is the program that runs on the machine. PlayStation program code is written using the standard GNU C development environment.
- Data consists of the 3D models, bitmaps and sound data that the program uses to generate the application's output. Although the PlayStation uses a set of proprietary data file formats, a wide range of commercial file formats can be converted easily into PlayStation formats.

Creating Graphics Data

The data used in a Net Yaroze PlayStation application can be classified into three general types: 2D graphics (often referred to as bitmaps), 3D graphics (often referred to as models) and sound data. Movies (often referred to as Streaming or Full Motion Video), are not available for use with the Net Yaroze system.

2D Graphics Data

The 2D graphics data used by the PlayStation (i.e. bitmaps) provide the imagery for sprites and textures for 3D graphics. 2D graphics data is saved in the PlayStation's TIM data format where it is handled directly by the integrated graphics service.

In the Net Yaroze system, 2D graphics data generated from paint tools on Windows and Macintosh systems can be converted to TIM format using special converters such as the Net Yaroze tool TIMUTIL.

For more information on the TIM data format, refer to the Net Yaroze Members' Web site.

Supported File Formats

The Net Yaroze system provides support for converting the following file formats into TIM data.

- Windows bit-mapped format (BMP)
- Macintosh PICT format (PICT)
- RGB format (RGB)

Tools

The Net Yaroze system provides the Windows-based TIMUTIL tool (timutil.exe) to convert files from existing formats to TIM.

Please refer to Chapter 14, *Graphics Tools* for details on how to use the TIMUTIL tool.

Creating Data and Converting to TIM

Use a painting tool which outputs data in one of the 2D file formats supported by the Net Yaroze system (BMP, PICT, RGB) then use the TIMUTIL tool to convert this file into TIM.

Verifying Data

The Net Yaroze system has a DOS-based tool called the TIM viewer (timv.bat) which runs on the Net Yaroze PlayStation. The contents of a TIM file can be viewed and verified using the TIM viewer tool.

Please refer to Chapter 14, *Graphics Tools,* for information on how to use the TIM viewer tool.

2D Tools List

Tool	Environment	Functions
timutil.exe	Windows	Converts data from BMP, PICT, RGB to TIM
timv.bat	MS-DOS	Displays TIM data on the PlayStation

Table: 2D Graphic Tools

3D Graphics Data

The 3D graphics data used by the PlayStation (i.e. modeling data) is used to draw 3D objects. 3D graphics data is saved in the PlayStation's TMD data format where it is handled directly by the integrated graphics service.

The Net Yaroze system provides several conversion tools for 3D graphics data. These tools convert models from the DXF format (a standard 3D modeling format) to the RSD format (an intermediate file format), then to the PlayStation's TMD format. The intermediate RSD format is an ASCII-based file which can be edited using a text editor. The final TMD file is a binary format which can be used directly by the Net Yaroze library functions.

Please refer to the Net Yaroze Members' Web site for more information on RSD and TMD data.

Supported File Formats

The DXF format is a standard 3D modeling format that is supported by virtually all of the commercial 3D modeling tools.

Please refer to the *AutoCad Reference Manual* published by AutoDesk Ltd for more information on the DXF format.

Tools

The following tools are provided to convert files from existing 3D graphics formats.

Please refer to Chapter 14, *Graphics Tools,* for more information on how to use these tools.

Tool	Environment	Functions
dxf2rsd.exe	MS-DOS	Converts DXF files to RSD files
dxf2rsdw.exe	Windows	Converts DXF files to RSD files
rsdlink.exe	MS-DOS	Converts RSD files to TMD files
rsd2dxf.exe	MS-DOS	Converts RSD files to DXF files
rsdcat.exe	MS-DOS	Links several RSD files into a single RSD
rsdform.exe	MS-DOS	Transformation and movement of RSD models

Table: 3D Graphic Tools

Creating Data and Converting to TMD

Create the original data file using a commercial modeling tool then convert it to TMD format using the tools described above. The modeling tool must be able to output data in the DXF data format.

The DXF file must first be converted into RSD format before it can be converted to TMD. The RSD format is an artists' format primarily used to apply texture mapping. RSD is actually composed of four different file types, as shown in the table below.

File name	Content
*.RSD	File connection information
*.PLY	Polygon information
*.MAT	Material information
*.GRP	Group information

These files are all text files and can be easily edited using a standard text editor. An RSD file must always be created even if editing is not required.

The conversion sequence is always DXF->RSD->TMD.

Verifying Data

The Net Yaroze system has a DOS-based tool called the RSD viewer (rsdv.bat) which runs on the Net Yaroze PlayStation. The contents of an RSD file can be viewed and verified using the RSD viewer tool.

Please refer to Chapter 14, *Graphics Tools,* for information on how to use the RSD viewer tool.

Tool	Environment	Functions
rsdv.bat	MS-DOS	displays RSD data on the PlayStation

Table: 3D Graphic Data Verification Tools

Creating Sound Data

Sound data used in a Net Yaroze PlayStation application can be classified into two general types: score data and waveform data.

Score data is stored in the PlayStation SEQ format. Waveform, or sampled data, is stored in the PlayStation VAB format. VAB is a multi-sound format which manages multiple VAG-encoded single-sound data units. The SEQ and VAB data formats can be handled directly by the Net Yaroze sound services.

Standard MIDI data (SMF) and AIFF data created using commercial sound tools can be converted into SEQ data and VAB data using special Net Yaroze conversion tools.

Please refer to the Net Yaroze Members' Web site for more information on SEQ and VAB data.

Supported File Formats

In the Net Yaroze system, the following standard file formats can be converted to SEQ and VAB data.

· Standard MIDI file Format 1 (SMF)

· AIFF (or alternatively 16-bit straight PCM waveform data, these are formats for sampled data)

Tools

The following tools are provided for converting to PlayStation sound data formats.

For details on how to use these tools, please see Chapter 15, *Sound Tools*.

Tool	Environment	Functions
smf2seq.exe	MS-DOS	Converts standard MIDI files into SEQ data
aiff2vag.exe	MS-DOS	Converts AIFF or 16-bit straight PCM into VAG format
mkvab.exe	MS-DOS	Creates VAB data based on VAG data and attribute definition files
vabsplit.exe	MS-DOS	Divides VAB data into VH data and VB data

Table: Sound Tools

Creating Data and Converting to PlayStation Formats

Create sound data using commercial sequencing or waveform editing software which can output data in one of the file formats supported by the Net Yaroze system. This data can be converted into PlayStation SEQ or VAB format using one of the tools listed above.

Verifying Data

The Net Yaroze system has DOS-based tools for playing back SEQ and VAB data files on the Net Yaroze PlayStation. These tools can be used to find out how converted sound data will be reproduced within the finished application.

Please refer to Chapter 15, *Sound Tools,* for information on how to use these tools.

Tool	Environment	Functions
sndplay.bat	MS-DOS	Reproduces SEQ data on the PlayStation
vabplay.bat	MS-DOS	Reproduces VAB data on the PlayStation

Table: Sound Data Verification Tools

Creating a Program

The Net Yaroze program development process uses the industry standard GNU program development tools and conforms to the standard process of program development in C. This section describes a simple example of how a program is created using this process. Please see the Net Yaroze Additional Reading List at the end of the *Start-Up Guide* for further information.

Create Code

1. Write C code and save it in a text file. Code can be created using a standard text editor and is usually saved with a file extension of ".c".

2. Compile the source code (from step 1) using *gcc*, the GNU compiler. The compiler will convert source code into object code suitable for linking into the program executable. Object code is often saved in a file with an extension of ".o" or ".obj" by the compiler.

3. Link the object code to the Net Yaroze library using *ld*, the GNU linker. The linker will combine the individual objects of your program with the library to produce the final program executable. By default, the executable is named "a.out" by the linker. Note that the compiler can be invoked so that it calls the linker automatically to generate the executable.

4. Run the executable on the PlayStation by downloading it using the SIOCONS program.

The process described above is both incremental and iterative. It is incremental because your source code will often have syntax errors that are detected by the compiler. You will need to edit your source code to fix the syntax errors before the compiler will generate an object file.

Once you have successfully linked your object code and created the executable, you will often find bugs that will cause the program to behave incorrectly. You will need to iterate through the entire process until you have fixed your source code to remove the bugs.

gdb, the GNU program debugger is a useful tool for finding bugs in your program. *gdb* allows you to step through your program and monitor its behavior.

Makefile

The process of compiling and linking a program can be quite complex. It is often necessary to specify many commands and options to run the development tools. A makefile can be used to simplify this process. A makefile makes it easier to compile and link a program by automatically issuing the proper commands and options, much like a DOS batch file.

The makefile can also be used to manage a project that consists of many separate files. *make*, the program maintenance utility, operates on makefiles and can be used to recompile only those files that change when the source code is updated.

Making a Library of Useful Routines

As you develop applications, over time you may create basic functions that you use repeatedly in your applications. Rather than copying the source code from one application to another, it is usually more efficient to group the set of functions into a library where they can be used by any application. Using libraries properly can reduce development time and make your code cleaner and easier to read.

The Net Yaroze system provides *ar*, the GNU librarian, and *nm*, the GNU object symbol manager for creating and maintaining libraries.

Creating a Release Version

After you have successfully debugged your executable code, you need to remove any debugging information from it to create a release version. The utility *strip*, removes symbol information (used by the debugger) from the executable program.

Tools

Tool	Environment	Functions
gcc.exe	MS-DOS	Compiles .c (source code) files into .o or .obj (object) files
		Links object files together to make an executable program
ld.exe *	MS-DOS	Links object files together to make an executable program
siocons.exe	MS-DOS	Serial console program
gdb.exe	MS-DOS	Debugger
strip.exe	MS-DOS	Removes symbol information
make.exe	MS-DOS	Manages makefiles

* *gcc* (GNU compiler) is sufficient to create the executable file (it can compile and automatically call the linker), although you may prefer to use the dedicated linker tool, *ld*.

14

Graphics Tools

The Net Yaroze graphics tools consist of data converters and previewers. Data converters convert graphics files created by commercial tools (e.g. bmp, pict and dxf files) into data files in the PlayStation format. Previewers display graphics data on a TV monitor using the PlayStation.

Listed below are the dedicated graphics formats used in the PlayStation. Please refer to the Net Yaroze Web site for detailed information on each of these formats.

·	RSD data	3D object data (initial conversion format for artists)
·	TMD data	3D object data (second conversion format for programmers)
·	TIM data	Image data (2D)

Graphics tools run under Windows or MS-DOS. The rest of this chapter briefly describes the features of the most important graphics tools of Net Yaroze.

dxf2rsd, dxf2rsdw

dxf2rsd and **dxf2rsdw** convert standard 3D object files that are in DXF format into PlayStation 3D object files in RSD format. **dfx2rsd** is a DOS tool while **dfx2rsdw** is the Windows version.

rsd2dxf

rsd2dxf converts an RSD file set into a file in DXF format.

rsdcat

rsdcat combines multiple RSD files into a single RSD file.

rsdform

rsdform performs simple editing of RSD data (such as moving, expanding and contracting of objects).

rsdlink

rsdlink converts RSD data into TMD data.

TMD is the 3D graphics format used by the PlayStation. PlayStation applications can process TMD files directly using the graphics library, without further conversion.

rsdv

rsdv is the RSD data previewer. **rsdv** displays RSD data on a TV monitor using the PlayStation.

timutil

timutil is used to convert commonly-used image data formats into TIM format used by the PlayStation. The list below show the image formats which **timutil** can process.

- Windows BMP
- Macintosh PICT
- Standard RGB
- PlayStation TIM

timv

timv is the TIM data previewer. **timv** displays TIM data on a TV monitor using the PlayStation.

dxf2rsd.exe

dxf2rsd converts DXF files into PlayStation 3D object data files (RSD).

Usage

```
dxf2rsd [options] DXF-files
```

When a DXF file is provided as the argument, the following four files are created:

- An RSD file (*.rsd)
- A polygon file (*.ply)
- A material file (*.mat)
- A group file (*.grp)

Note: Wildcards can be used in the argument to convert more than one file at once. The file extension '.dxf' can be omitted.

Options

-o targetname
Specifies the name of the RSD output file (where *targetname* is the specified file name). The default output filename is the input filename minus the .dxf extension in the current directory.

-col r g b
Specifies the color of the overall model using RGB values (each in the range of 0-255). The default setting is gray (200 200 200).

-cf color-file
Specifies a color table file which contains color definitions (where *color-file* is the specified file name). The -cf option is almost always used in conjunction with the -cl option, described below.

-cl
Outputs a list of undefined colors to standard output. Also, orders polygons of the same color and outputs them to the MAT file. The default is OFF. (See Example 2, below.)

-info
Displays information about the DXF input file on the standard output. This option gives the approximate size and number of polygons in the file. No conversion is performed when this option is specified.

-max n-poly
Specifies the maximum number of polygons that can be converted (where *n-poly* is the specified number). The default is 10000.

-quad *or* ***-quad1***
When this option is specified, division of 4-vertex 3DFACE polygons into triangles is not performed. This option can reduce the number of polygons in the model. The default setting is OFF.

-quad2 (threshold-value)
With this option, adjacent pairs of triangles are formed into single quadrilaterals. The optional argument (*threshold-value*) must be a decimal number between 0.0 and 90.0. This argument controls the combining of pairs of triangles by specifying a maximum angle of orientation difference (normal line vectors) between

the triangles that can be combined. Any pair of triangles with an angle of orientation greater than the *threshold-value* is not combined.

When the argument is 0.0, only triangles with exactly the same normal line vectors will be combined. When the argument is 10.0, a difference in orientation of up to 10 degrees is allowed. The default value of this argument is 1.0. (See Example 4, below.)

-quad3
When this option is specified, triangles are created as quadrilaterals in which the 3rd and 4th vertices are identical to each other. This option causes all polygons to be defined as quadrilaterals.

-s** and **-g
Performs smooth (Gouraud) shading. The default is OFF.

-e distance-value
Reduces the number of polygons by causing all vertices within a sphere having the specified radius (the *distance-value*) to be treated as a single vertex. (Note that the radius calculation is carried out after scaling using the -sc option described below.)

-r
Reduces the number of normals by not creating identical normal vectors. This option is valid for flat shading. The default is OFF.

-n
Normals are not created. Use this option when there will be no light source calculation. The default is OFF.

-sc factor
Expands and contracts the model by a scaling factor (*factor*) specified as a decimal number. The default is 1.0.

-t x y z
Translates the position of the model left or right, up or down, and backward or forward by the specified amounts (*x*, *y* and *z*). Arguments are specified as decimal numbers. The default is (0.0, 0.0, 0.0).

-auto
Shifts the position of the model toward the origin (x, y and z as zero) and scales the model so that it fits within a cube with sides of 1000. The default is OFF.

-back
Reverses the direction of the normals of all polygons. The default is OFF.

-both
Creates all polygons as double-sided. The default is OFF.

-dup
Creates front and back polygons for each polygon, doubling the number of polygons. The default is OFF.

-nopl
Ignores POLYLINE data and converts only 3DFACE data. The default is OFF.

-Y+Z, +Y-Z, +Y+Z, -Z-Y, -Z+Y, +Z-Y, +Z+Y
Specifies how the coordinate system is converted. These options are used to specify the labelling and direction of the coordinate axes which extend towards the viewer and upwards, as if the viewer were looking at the modeller coordinate system from the front. The first value of the pair specifies the forward axis and the second value, the upward axis.

For example, *-Y+Z* indicates that the forward axis is the negative Y axis and the upward axis is the positive Z axis. The coordinate system referred to here is the coordinate system of the DXF file. This coordinate system does not necessarily coincide with the physical screen of the modeller.

The default coordinate system is -Y+Z.

dxf2rsd converts the specified coordinate system into the PlayStation coordinate system (-Z-Y). (In the PlayStation coordinate system, the forward axis is the negative Z axis and the upward axis is the negative Y axis.)

-v
Outputs detailed information concerning the conversion to standard output. (See Example 1.)

Restrictions

The current version of **dxf2rsd** has the following restrictions.

- Only ASCII format DXF files are supported.
- Among the DXF entities, only 3DFACE data and POLYLINE data are supported.
- Sometimes large POLYLINE polygons cannot be converted. Wherever possible use a modeller to convert POLYLINE data into 3DFACE data (triangles and quadrilaterals) before creating the DXF file.
- Occasionally quadrilaterals that have vertices which are not all on the same plane are not displayed correctly.
- The number of polygons that can be converted is influenced by the number of vertices created and the number of normal lines. As a general rule, this number can be considered to be about 5000.

Supplementary Notes

3DFACE and POLYLINE are both data formats used to express DXF polygons. 3DFACE data represents single polygons (triangles and quadrilaterals) using four vertices, while POLYLINE data represents more than one polygon using connected line segments.

For a given model, 3DFACE format files tend to be larger in terms of amount of data than POLYLINE, but also have a higher level of compatibility. By contrast, using POLYLINE data reduces the amount of data and affords a greater degree of freedom, but, there may be cases when data cannot be successfully exchanged between different modellers.

3DFACE data can be directly converted into RSD data, but with POLYLINE data, triangulation must be performed first. Sometimes triangulation is not successful (in which case a 'Fail to triangulate!' error message appears). Furthermore, even when triangulation is completed successfully, sometimes with POLYLINE data the orientation of some of the polygons ends up reversed. However, POLYLINE data created using the *3D Studio* software package, referred to as 'POLYFACE MESH' data, is equivalent to 3DFACE data and thus has no conversion problems.

- The maximum number of polygons that can be realistically moved about as a single object on the PlayStation is about 2000. Use this number as a guide when creating model data.
- When conversion with flat shading cannot be performed because there are too many polygons, it is sometimes possible to convert using Gouraud shading.
- Each time the Y and Z coordinate axes are exchanged, or the positive and negative directions of each axis are switched, the front and back surfaces of the polygons are reversed.
- Depending on the modeller used, polygons may sometimes end up back to front, even when the data has been output as 3DFACE data. If all of the polygons are reversed, either change the coordinate system (e.g. +Y+Z), or use the -back option. When only some of the polygons are reversed, correct them using a modeller.
- Use a modeller to create DXF data for conversion using **dxf2rsd** that can output the whole model as 3DFACE DXF data, then reverse polygons individually. (However, data from *some* modellers can be converted even if these conditions are not met. For example, even when data cannot be converted directly, it may be possible to alter the DXF file via a **dxf2rsd**-tool-compatible modeller. In other words, if the DXF file has been created by a **dxf2rsd**-tool-incompatible modeller, open and re-save it in a modeller that is compatible with the **dxf2rsd** tool.)
- With large files, use the -n option to create data without normal lines.
- The -r option is not valid with Gouraud shading (-s or -g options) and cannot be used with normal line MIMe (the latter because -r changes the number and order of normal lines).
- The -quad2 option may be ignored in areas where the two triangles to be combined have been set to different colors in the DXF file by the modeller. In these cases, the -cl option should be used together with the -quad2 option. (For example: `dxf2rsd -quad2 -cl DXF-files` [where 'DXF-files' are the files to be converted]).

Example 1: Example of dxf2rsd output with the '-v' option

```
C:\> dxf2rsd -v -auto +Z+Y -quad -s foo
```

Note: Input file = foo.dxf. Output files = foo.rsd, foo.ply, foo.mat, foo.grp

Input File (DXF)			**Explanation**
SIZE	40230 lines		No. of lines in DXF file
VERTEX	4320		No. of vertices in DXF file
POLYGON	1468 (estimate)		Estimated no. of polygons
	3-poly	1376	1376 triangles
	4-poly	32	32 quadrilaterals
	(>9)-poly	2	2 polygons with more than 10 sides
	polylines	2 (max size=32)	2 polyline figures (32 vertices)
RANGE	x: -1.015	+0.785	Min. value ... max value
	y: -2.533	+0.768	for each axis. Y and Z axes
	z: -1.161	+0.625	are converted to 'PS' if necessary.
SCALE	302.870		Scaling factor
MOVE	(dx,dy,dz) = (34.788,267.255,81.207)		Amount of movement
MATERIAL	0		No. of colored polygons
Output File (RSD)			**Explanation**
VERTEX	796		No. of vertices after conversion
POLYGON	1468		No. of polygons after conversion
	triangles	1436	1436 triangles
	quadrangles	32	32 quadrilaterals
RANGE	-272.477	+272.477	Min. value ... max value for
	-500.000	+500.000	axes after conversion
	-270.510	+270.510	
MATERIAL	0		No. of polygons with material
NORMAL	796		No. of normal line vectors created

Table: Example of dxf2rsd Output Using the '-v' Option

Example 2: Using color information

Use the -cl option to maintain color added by the modeller in the RSD file. Polygons in the resulting file will be modelled "appropriately," so that parts that are colored the same in the unconverted file will be assigned the same color in the RSD file.

Exact color matching cannot be achieved because the unconverted DXF file only holds color numbers rather than RGB values. To maintain colors, a text editor can be used to edit the MAT file after conversion, or the colors can be specified before conversion using a color table file as shown below.

1. Output a list of undefined colors for the foo.dxf file, piped to a new file, foo.cl.

```
C:\> dxf2rsd -cl foo > foo.cl
```

2. Display the contents of the resulting file. (This is a list of unassigned colors).

```
C:\> type foo.cl
183
40
253
0
8
```

3. Colors can be assigned to color numbers by entering their RGB values (in the range 0-255) after the color number.

```
C:\> edit foo.cl
183   100 100 200
40     58  20  43
253    10 100  10
0     212  20 100
8       0 128 126
```

4. Run **dxf2rsd** with the -cf option and the foo.cl file specified.

```
C:\> dxf2rsd -cf foo.cl foo
```

The newly created RSD file will have colors assigned according to the color table file.

Example 3: Converting large data

For data that is extremely detailed (i.e. voluminous), the -e option can be used to reduce the volume of data by combining several vertices into a single vertex.

In the following example the number of vertices, the number of polygons and the number of normal lines, are reduced by considering any two vertices separated by a distance of 100 or less to be a single vertex. (Note that the distance is calculated according to the scale after expansion.) Depending on the data, an appropriate distance value can reduce the volume of data with virtually no change in shape.

Using the file big.dxf,

```
C:\> dxf2rsd -v -e 100 -sc 1000 big.dxf
```

Note: Input File = big.dxf. Output files = big.rsd, big.ply, big.mat, big.grp

Input (DXF)				**Explanation**
SIZE	134628 lines			
VERTEX	18982			
POLYGON	8618 (estimate)			
	3-poly	1746		
	4-poly	3436		
RANGE	x	-1.644	+1.545	Because the scale is
	y	-2.352	+0.000	small, it is
	z	-3.649	+3.993	multiplied by 1000
SCALE	10000.000			
MATERIAL	0			
Output (RSD)				**Explanation**
VERTEX	1208			
POLYGON	2708 (68%reduced)			No. of polygons reduced by about 30%
	triangles	2708		
RANGE	x	-643.811	+1545.072	
	y	-2352.365	+0.000	
	z	-3649.154	+3992.687	

Table: Example Converting Large Data

Specify the -r option in addition to those above to reduce the number of normal lines.

Example 4: Forming quadrilaterals

The -quad2 option combines neighboring pairs of triangles into quadrilaterals. In the following example, all neighboring triangles with an angle between them of five degrees or less are converted into quadrilaterals.

Using the file, earth.dxf,

```
  C:\> dxf2rsd -v -quad2 5.0 earth
```

Note: Input file = earth.dxf. Output files = earth.rsd, earth.ply, earth.mat, earth.grp

Input (DXF)				**Explanation**
SIZE	88158 lines			
VERTEX	8811			
POLYGON	2937 (estimate)			
	3-poly	2937		Originally 2937 triangles
RANGE	x	-4.000	+3.986	
	y	-3.997	+3.997	
	z	-4.000	+4.000	
MATERIAL	0			
Output (RSD)				**Explanation**
VERTEX	2231			
POLYGON	1686			1686 polygons in all
	triangles	43		43 triangles combined
	quadrangles	1251		1251 quadrilaterals formed
MATERIAL	0			
NORMAL	1686 (quad2 < 4.9870) ***(see note below)***			

Table: Example Forming Quadrilaterals

Note: 4.9870 indicates the largest angle between pairs of triangles that were actually converted. This angle is less than or equal to the allowed angle (in this case 5.0 degrees). Thus, if the same dxf file were converted using **dxf2rsd** with the option '-quad2 4.986', a smaller number of quadrilaterals would be formed.

dxf2rsdw.exe

dxf2rsdw is a Windows application that converts the DXF format files output by a variety of commercial modellers into RSD format files for use on the PlayStation. **dxf2rsdw** is the Windows version of **dxf2rsd**, the DOS tool described above. **dxf2rsdw** reads a DXF file and creates the following four output files.

- An RSD file (*.rsd)
- A polygon file (*.ply)
- A material file (*.mat)
- A group file (*.grp)

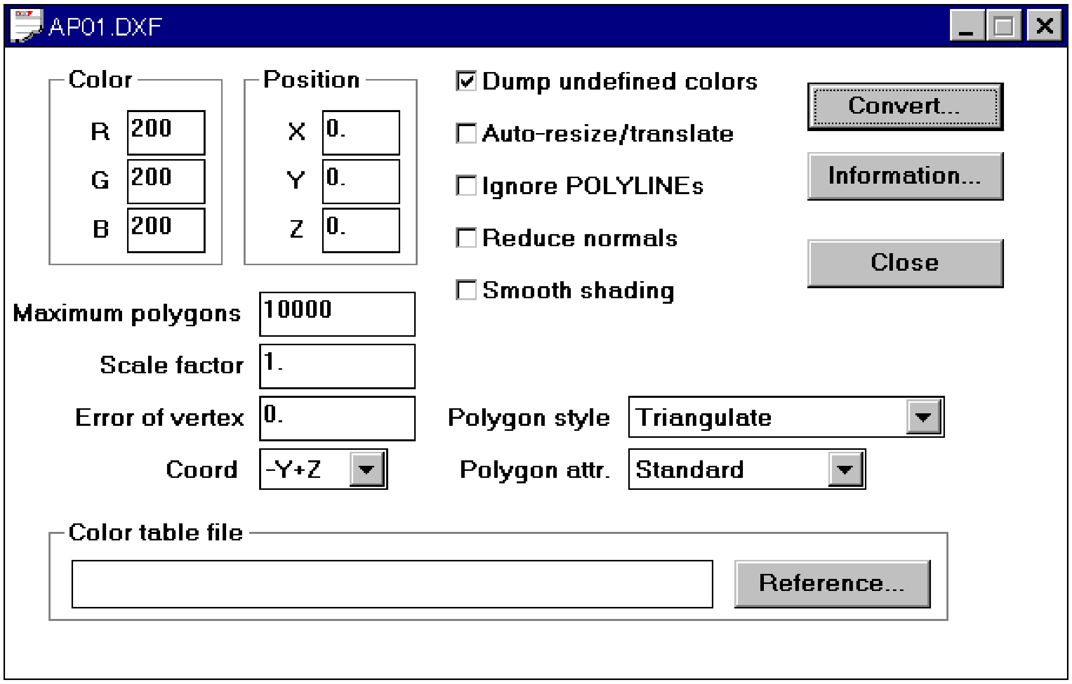
Usage

Double-click on dxf2rsdw.exe from the Windows File Manager or Windows Explorer. With the application running, perform the following steps.

1. Select a DXF file using 'Open...' in the File menu.
2. The Parameter Settings window will appear.
3. Change parameters as required in the window and select either 'Convert...' or 'Save as...' from the File menu. The save file dialog box will appear.
4. Specify the name for the converted file in this dialog box.
5. The file will be converted and saved under the specified name.

Alternatively:

Drag the DXF file to be converted from Windows Explorer or File Manager and drop it into the dxf2rsdw.exe window.



Color
R 200
G 200
B 200

Position
X 0.
Y 0.
Z 0.

[✓] Dump undefined colors
☐ Auto-resize/translate
☐ Ignore POLYLINEs
☐ Reduce normals
☐ Smooth shading

Maximum polygons 10000
Scale factor 1.
Error of vertex 0.
Coord -Y+Z
Polygon style Triangulate
Polygon attr. Standard

Color table file

Figure: The Parameter Settings Window

Description of Each Item in the Parameter Settings Window

For each item, the corresponding options in the MS-DOS version (dxf2rsd.exe) are noted. Please see the description of **dxf2rsd** in the previous section for more detailed information on each of the options.

Color

Specifies the color of the entire model in RGB format (each value in the range 0-255).

Corresponds to the '-col r g b' option in the MS-DOS version.

Position

Translates the model. The position should be specified.

Corresponds to the '-t x y z' option in the MS-DOS version.

Maximum polygons

Specifies the maximum number of polygons that can be converted.

Corresponds to the '-max n-poly' option in the MS-DOS version.

Scale factor

Performs scaling of the model. The scaling factor should be specified as a real number.

Corresponds to the '-sc factor' option in the MS-DOS version.

Error of vertex

All vertices within a sphere having the specified radius are treated as a single vertex. The distance calculation is made after scaling using the specified ‘Scale factor.’

Corresponds to the '-e distance-value' option in the MS-DOS version.

Coord

Specifies how the coordinate system is converted. The options are used to specify the labelling and direction of the coordinate axes which extend towards the viewer and upwards, as if the viewer were looking at the modeller coordinate system from the front. The first value of the pair specifies the forward axis and the second value, the upward axis.

For example, '-Y+Z' indicates that the forward axis is the negative Y axis and the upward axis is the positive Z axis. The coordinate system referred to here is the coordinate system of the DXF file. This coordinate system does not necessarily coincide with the physical screen of the modeller. The default coordinate system is '-Y+Z'.

dxf2rsdw converts the specified coordinate system into the PlayStation coordinate system (-Z-Y). (In the PlayStation coordinate system, the forward axis is the negative Z axis and the upward axis is the negative Y axis.)

*Corresponds to each option in the MS-DOS version. See the description of ' -Y+Z, +Y-Z, +Y+Z, -Z-Y, -Z+Y, +Z-Y, +Z+Y' under **dxf2rsd** for more information.*

Dump undefined colors

See Notes: Using Undefined Colors, below. (Unlike the MS-DOS version, the default setting for this option is ON.)

Corresponds to the '-cl' option in the MS-DOS version.

Auto-resize/translate
Shifts the position of the model toward the origin (x, y and z as zero) and scales the model so that it fits within a cube with sides of 1000. The default is OFF.

Corresponds to the '-auto' option in the MS-DOS version.

Ignore POLYLINEs
Ignores POLYLINE data and only converts 3DFACE data.

Corresponds to the '-nopl' option in the MS-DOS version.

Reduce normals
Reduces the number of normals by not generating identical normal vectors. Particularly effective with flat shading.

Corresponds to the '-r' option in the MS-DOS version.

Smooth shading
Performs smooth (Gouraud) shading.

Corresponds to the '-s' and '-g' options in the MS-DOS version.

Polygon style
The following options are available.

'Triangulate'
All polygons are divided into triangles.

‘No Triangulation'
Specifies that 4-vertex 3DFACE polygons should NOT be divided into triangles. The overall polygon count in the model can be reduced by using this option.

Corresponds to the '-quad' option in the MS-DOS version.

'Form Quadrilaterals'

Adjacent pairs of triangles are formed into single quadrilaterals. The optional value, 'threshold-angle' must be a decimal number between 0.0 and 90.0. This value controls the combining of pairs of triangles by specifying a maximum angle of orientation difference (normal line vectors) between the triangles that can be combined. Any pair of triangles with an angle of orientation greater than the 'threshold-value' is not combined.

When the argument is 0.0, only triangles with exactly the same normal line vectors will be combined, when the argument is 10.0, a difference in orientation of up to 10 degrees is allowed. The default value is 10.0.

'Quadrilaterals'
Triangles are created as quadrilaterals in which the 3rd and 4th vertices are identical to each other. Using this option, all polygons can be defined as quadrilaterals.

Corresponds to the '-quad2' option in the MS-DOS version.

Polygon attr.
The following options are available.

'Standard'
Polygons and normal lines are created exactly as they are specified in the DXF file.

'Invert Normal Lines'
Reverses the normal lines of all polygons.

Corresponds to the '-back' option in the MS-DOS version.

'Double-Sided Polygons'
All polygons are created as double-sided polygons.

Corresponds to the '-both' option in the MS-DOS version.

'Front and Back Polygons'
All polygons are created as backwards-facing polygons.

Corresponds to the '-dup' option in the MS-DOS version.

'Don't Create Normal Lines'
Normal lines are not created. Use this option when there will be no light source calculation.

When the amount of data is so large that conversion cannot be completed, this option is enabled automatically.

Corresponds to the '-n' option in the MS-DOS version.

Color table file
Specifies a color table file. When the "Reference ..." button is pressed, the File dialog box is displayed and a filename can be selected.

(See Notes: Using Color Information, below).

Corresponds to the '-cf color-file' option in the MS-DOS version.

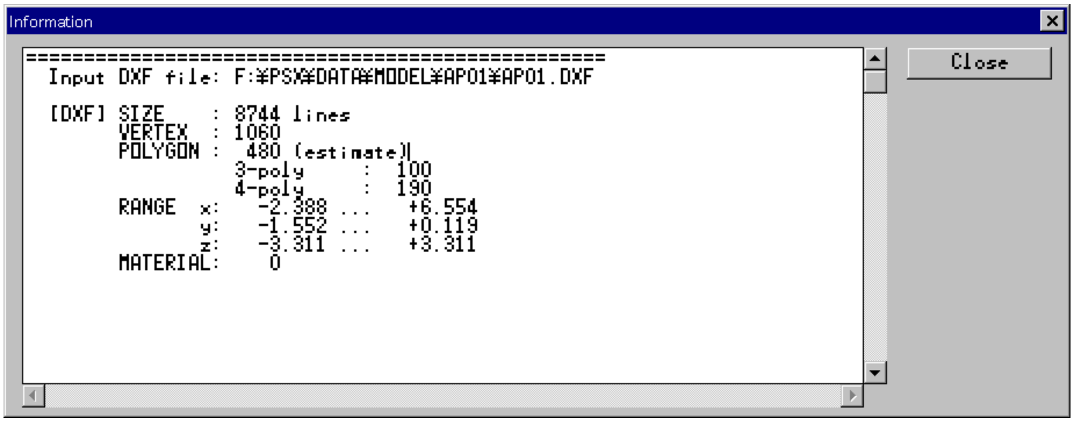
Convert…
Performs format conversion according to the specified parameters.

A Save File dialog box will be displayed. Enter the name of the converted file in the Save File dialog box. The default directory is the current directory.

This function is the same as the 'Save As...' command from the File menu.

Information…
Displays an information dialog box (shown below) containing details of the input DXF file, including the approximate size and number of polygons.

Corresponds to the '-info' option in the MS-DOS version.



```
==================================================
  Input DXF file: F:¥PSX¥DATA¥MODEL¥AP01¥AP01.DXF

  [DXF] SIZE    : 8744 lines
        VERTEX  : 1060
        POLYGON :  480 (estimate)
                  3-poly      :  100
                  4-poly      :  190
        RANGE  x:   -2.388 ...   +6.554
               y:   -1.552 ...   +0.119
               z:   -3.311 ...   +3.311
        MATERIAL:    0
```

Figure: Information Dialog Box

Note: Text can be copied to the Windows clipboard by selecting a range of text with the mouse and pressing Ctrl+C. To save the information in a file, the copied text can be pasted into a text editor.

.

Close

Closes the current window.

This function is the same as the 'Close' command from the File menu.

Menu Bar

'Open...' Command ([File] menu)

Opens an existing DXF file. A File Open dialog box is displayed. The DXF file to be opened is specified here.

'Save As...' Command ([File] menu)

Saves an opened DXF file under a specified filename. A File Save dialog box is displayed. An appropriate filename should be entered then the file is saved. Conversion to RSD format is performed according to the specified parameters.

Notes

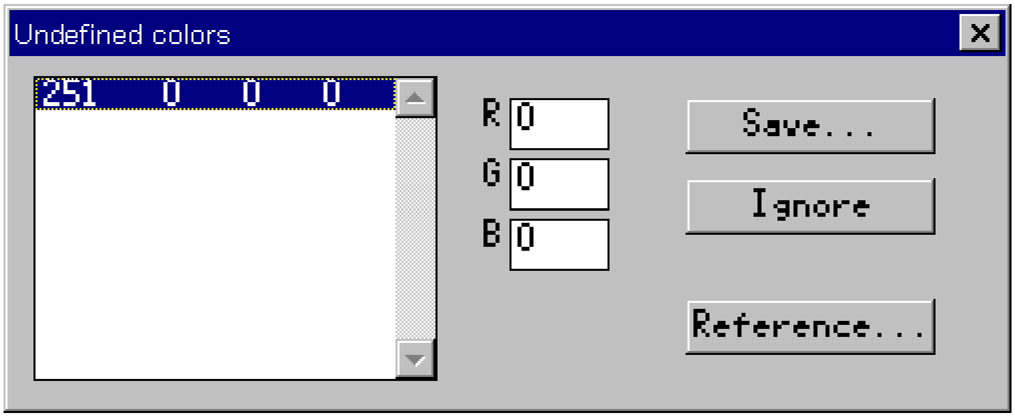
Using Color Information

To maintain color that has been added by the modeller in the RSD file, the DXF file must be converted twice. The first conversion is used to determine what colors are undefined. The second conversion allows you to specify how these colors will be converted. Note that when specifying undefined colors, exact color matching cannot be maintained because the unconverted DXF file only holds color numbers rather than RGB values.

The process is described below.

· *Find the undefined colors*

1. Select 'Dump undefined colors' in the conversion dialog box to get a listing of the undefined colors. After conversion, an Undefined Colors list is displayed in a dialog box as shown below.


Undefined colors

251 0 0 0

R 0
G 0
B 0

Figure: The Undefined Colors Dialog Box

· *Specify colors*

Use the dialog box to specify colors for the list as follows:

1. Select a line in the table and edit the R, G and B fields to provide the required color (using RGB values between 0-255).
2. Click 'Save' and specify a name for the resulting color table file in the File dialog box.
3. Redo the conversion specifying the name of this undefined color file in the 'Color table file' field.

Colors in the newly created RSD file will be assigned according to the color table file.

For other restrictions and additional notes, see “dxf2rsd.exe, Example 2: Using Color Information.”

rsd2dxf.exe

Converts an RSD format file into a DXF format file.

Usage

```
rsd2dxf [options] [ARG-files ...] RSD-files ...
```

An RSD file set is specified as an argument and is converted to the DXF format. The file set is specified as *name* and refers to the four files, *name*.rsd, *name*.ply, *name*.mat and *name*.grp. The output DXF filename is the input RSD filename appended with a ‘.dxf’ extension (*name*.dxf).

Wildcards can be used as arguments.

Options

-o output-file
Specifies the output dxf filename. The default output filename is the input filename appended with a ‘.dxf’ extension, with the file being placed in the current directory. This option can be used only when there is a single RSD input file.

-r
Reverses the orientation of all the polygons in the output DXF file.

-h
Displays a list of the available options, as well as brief instructions on usage.

-v
Displays information about the conversion.

[ARG-file]

Specifies an argument file containing command arguments and input filenames. If a filename has a '.arg' extension, it is assumed to be an argument file.

Argument files should be used when arguments are longer than 128 bytes. Lines beginning with "#" inside the file are assumed to be comments and are ignored.

Notes

- In the coordinate system of the generated DXF file, +X will be to the right, +Y will be downward, and +Z will be inward. These settings are identical to the default values assumed by **dxf2rsd** for DXF files. Therefore, when a generated DXF file is to be opened and edited by a modeller, it may be necessary to rotate the model to correspond to the coordinate system used by the specific modeller.
- The normal, color, and texture information contained in the RSD files is discarded.
- If the output filename is already being used, the contents of the file will be overwritten. Please note that no warning messages will be displayed.
- Argument files cannot be specified inside an argument file (they are ignored).
- Wildcards cannot be used in an argument file.

rsdcat.exe

Joins multiple RSD format files into a single RSD file set.

Usage

```
rsdcat [options] [ARG-files ...] RSD-files ...
```

The RSD file sets supplied as arguments are joined together, and a new RSD file set is created.

Wildcards can be used as arguments.

Options

-o output-file

Specifies the output filename. The default is ‘out.rsd’, ‘out.ply’, ‘out.mat’, and ‘out.grp’, with these files being placed in the current directory. When a filename is specified with this option, the output files are generated using the input filename appended with the ‘.rsd’, ‘.ply’, ‘.mat’ and ‘.grp’ extensions.

-h
Displays a list of the available options as well as brief instructions on usage.

-v
Displays information about the conversion.

[ARG-file]
Specifies an argument file containing command arguments and input filenames. If a filename has a ‘.arg’ extension, it is assumed to be an argument file.

Argument files should be used when arguments are longer than 128 bytes. Lines beginning with "#" inside the file are assumed to be comments and are ignored.

Notes

- For each RSD file from the input, a group is added to the generated group file (*.grp). All of the polygons from the RSD file are included as members of a group. The filename of the input RSD file without the extension is used as the group name.
- No compression will be performed by **rsdcat** even if texture, vertex, normal, or group data overlaps across multiple input RSD files. Consequently, the files output from **rsdcat** may need some manual editing.
- If the output filename is already being used, the contents of the file will be overwritten. Please note that no warning messages will be displayed.
- Argument files cannot be specified inside an argument file (they are ignored).
- Wildcards cannot be used in an argument file.

rsdform.exe

Transforms and moves 3D object data (artist-oriented RSD data).

Usage

```
rsdform [options] RSD-name
```

This command alters the form of objects by applying scaling, rotation and translation to RSD format object data. If more than one transformation operation is specified at the same time, the operations are carried out in the following order: scaling → rotation → translation.

Specify the RSD file name as either 'file.rsd' or 'file' (without the .rsd extension).

Options

-o RSD-name
Specifies the RSD name of the output file (where *RSD-name* is the specified file name). The output name cannot be the same as the input file name. The default filename is 'a'.

-s x y x
Specifies scaling values for the x, y and z axes. The values are specified as float values, where 1.0 indicates no change. Negative values can also be specified, in which case the corresponding axis will be output as a mirror image of the original.

-r x y z
Specifies angle of rotation around the x, y and z axes. The values are specified as float values. Positive values indicate a clockwise rotation (see Supplementary Notes, below). The default unit is degrees, but the -rad option can be used to specify angles in radians.

-t x y z
Specifies translation along the x, y and z axes. The values are specified as float values.

-rad
Specifies that radians should be used as the default unit for the rotation angle.

-l
Performs scaling and rotation around the center of the model. Used with the -s and -r options.

-c

Share files if possible. (See Supplementary Notes, below.)

-c[pmg]
Share PLY/MAT/GRP files. A shared file can be specified by combining the three letters (p, m, g). An example would be -cpg (this is equivalent to -cp -cg). When other options are specified and files cannot be shared (see Supplementary Notes), a warning message will be displayed and the share option will be ignored.

-k
Keeps the version of the original file. If this option is not specified, the result is @RSD940102, @PLY940102, @MAT940801 and @GRP940102.

-quiet
The modification history will not be added as comments to the output file. The default setting is OFF.

-v
Detailed information on the conversion is output to standard output.

Supplementary Notes

- RSD files are formed in sets of four files with the following extensions: .rsd, .ply, .mat, .grp. These files must be placed in the same directory, and are generally placed in the RSD directory within the project directory.
- Files can be shared if the following conditions are met.
- PLY files: No transformations are specified.
  MAT files: The output file is not a mirror image of the original data.
  GRP files: Can always be shared.

- The coordinate system of the PlayStation is represented as 16-bit values. The RSD format is not restricted to 16 bits, but when **rsdlink** is used to convert an RSD file to the TMD format, the result must fit into the PlayStation coordinate system.

· 'Clockwise rotation' refers to the direction in which the fingers curl when the thumb of the right hand is extended to point in the positive direction of the axis of rotation.

Example 1: Basic usage

In this example, the model is scaled along the x axis by a factor of 10, then a 45-degree rotation is performed around the x axis. Files are to be shared if possible.

Using the file cube.rsd as input,

```
  C:\> rsdform -v -s 10 1 1 -r 45 0 0 -c cube
```

Input RSD file (cube.rsd) Details					**Explanation**
SCALE		10	1	1	
ROTATION		45	0	0	(degrees) from transformation specification
TRANSLATION		0	0	0	
RANGE	x	+0.0	+1000.0	(center) +500	Size before transformation
	y	-500.0	+500.0	+500	(max, min and central
	z	-500.0	+500.0	+500	coordinate values for each axis)
Output RSD file (a.rsd) Details					**Explanation**
FILE TRANSFORMATION		cube.ply	becomes	a.ply	
		cube.mat	becomes	cube.mat (shared)	This file can be shared
		cube.grp	becomes	cube.grp (shared)	This file can be shared
		cube.rsd	becomes	a.rsd	
RANGE	x	+0.0	+10000.0	+5000.0 (center)	Size after transformation
	y	-707.1	+707.1	+0.0	(max, min and central
	z	-707.1	+707.1	+0.0	coordinate values for each axis)

Table: Example rsdform Basic Usage

Example 2: Performing local transformation

The -l option is applied to the deformation from Example 1.

With this option enabled, rotation and scaling can be performed around the model's center of gravity so the model's position remains unchanged. Compare this with the results from Example 1.

Using the file cube.rsd as input,

```
C:\> rsdform -v -l -s 10 1 1 -r 45 0 0 -c cube
```

Input RSD file (cube.rsd) Details					**Explanation**
SCALE		10	1	1	
ROATION		45	0	0	(degrees) from transformation specification
TRANSLATION		0	0	0	
RANGE	x	+0.0	+1000.0	(center) +500	Size before transformation
	y	-500.0	+500.0	+0.0	
	z	-500.0	+500.0	+0.0	
Output RSD file (a.rsd) Details					**Explanation**
FILE TRANSFORMATION		cube.ply	becomes	a.ply	
		cube.mat	becomes	cube.mat (shared)	This file can be shared
		cube.grp	becomes	cube.grp (shared)	This file can be shared
		cube.rsd	becomes	a.rsd	
RANGE	x	-4500.0	+5500.0	**+500.0** (center)	Size after transformation
	y	-707.1	+707.1	+0.0	No change to center after transformation
	z	-707.1	+707.1	+0.0	

Table: Local Transformation with rsdform

Example 3: Copying RSD files

rsdform can also be used to copy and rename RSD files. A batch file such as the one shown below can be useful for this.

Contents of batch file:

```
      @ECHO OFF
      rsdform -o %2 %1
```

Example 4: Overwriting RSD files

rsdform does not allow RSD files to be overwritten before transformation. However, a batch file such as the one shown below can be used to overwrite RSD files.

Contents of batch file:

```
@ECHO OFF
SET ARGS=
REM                                  read all arguments
:LOOP1
IF "%9"=="" GOTO LABEL1
SET ARGS=%ARGS% %1
SHIFT
GOTO LOOP1
:LABEL1
REM                                  convert to filename ' _tmp' for now
rsdform -o _tmp %ARGS% %1 %2 %3 %4 %5 %6 %7 %8
IF ERRORLEVEL 1 GOTO END
REM                                  find final argument (= input filename)
:LOOP2
IF "%1"=="" GOTO LABEL2
SHIFT
GOTO LOOP2
:LABEL2
REM                                  overwrite the input file (%0)
rsdform -o %0 -quiet _tmp
REM                                  delete '_tmp.*' and exit
```

```
DEL _tmp.*
:END
SET ARGS=
```

rsdlink.exe

Converts artist-oriented 3D object data files (RSD files) into object data files (TMD files) that can be handled by the PlayStation library.

Usage

```
rsdlink [options] rsd-names ...
rsdlink [options] rsd-names [options] rsd-names [options] ....
rsdlink [options] arg-files ...
```

Multiple RSD data files supplied as arguments are linked into a single TMD file. Scaling factors and translation values should be specified separately for each RSD data file, if required. (See Example 1.)

If there are no file paths specified for the required RSD filenames in the argument, **rsdlink** searches the current directory first, followed by the '.\RSD' directory.

When there are a large number of arguments, the arguments can be placed in an argument file. An argument file is a text file with a file extension of '.arg' that contains a list of arguments. Note that argument files cannot be specified inside an argument file. The input filename extension of an RSD file ('.rsd') may be omitted, but the filename extension of argument files ('.arg') must be specified. (See Example 2.)

Options

-o filename
Specifies the output filename. The default is 'a.tmd'.

-s factor
The RSD data specified in subsequent arguments is scaled by the specified scaling factor. This scaling factor will be applied to all RSD data thereafter until a new scaling factor is specified.

-sc factor

The scaling factor is rounded to 2^{factor}. The default value is 1.0. The coordinate values in TMD format are 16-bit integers, so the scaling factor must be specified appropriately to allow the data to fit within the PlayStation's coordinate system.

-t x y z
The RSD data specified in subsequent arguments is translated by the indicated offsets. The default is (0 0 0). The specified offsets will be applied to all RSD data thereafter until new translation offsets are specified.

-info
Detailed information about the object being converted such as its type, vertex coordinate values and texture information is sent to standard output (see Example 4).

-v
Detailed information about the conversion such as the number of polygons, is sent to standard output. The approximate size of each RSD in the PlayStation coordinate system is also output so the model's position and size can be confirmed before it is displayed on the PlayStation. (Range: vertex minimum and maximum values (x, y, z)) (See Example 3.)

The following restrictions apply to the current version.

Restrictions

· Linking may not be possible when a single RSD data set contains an extremely large number of polygons. The maximum number of polygons in a file where linking is guaranteed is roughly 5000, although the actual limit varies according to the number of vertices, the number of normals, and the amount of free memory available. However, this restriction only applies to a single RSD data set. For practical purposes, there is no limit to the number of RSD data sets to be linked or to the number of polygons in the entire TMD.

Note

· It is recommended that RSD translation and scaling operations be performed using the commands:

`dxf2rsd -sc -t` and `rsdform -s -t`.

This will result in data that is more accurate and easier to manage.

- The texture file (TIM file) specified in an .rsd file must be in the same directory as the RSD file (.\) or in the ..\TIM directory because **rsdlink** will search for TIM files in this order.

Example 1: Basic usage

```
C:\> rsdlink -v -o boxes.tmd box1 -s 2.0 -t 100 100 100 box1a box2 -t 200 -
200 200 box3
```

In this example the following four objects are combined to form a single TMD file ('boxes.tmd')

box1	original size
box1a	box1 scaled to twice the size and translated by (100 100 100)
box2	box1 scaled to twice the size and translated by (100 100 100)
box3	box1 scaled to twice the size and translated by (200 -200 200)

Example 2: Collecting arguments together in a file

If there are a large number of arguments as in Example 1, the arguments can be saved in an argument file as shown below.

1. Create the argument file: test.arg as a text file.

```
box1
-s 2.0 -t 100 100 100 box1a box2
-t 200 -200 200 box3
```

2. Use test.arg as an argument.

```
C:\> rsdlink -v -o boxes.tmd test.arg
```

Example 3: Example output with '-v' option

Using the input file dino.rsd,

```
C:\> rsdlink -v dino -s 100 box
```

Output		Description
1 - RSD		1st RSD ("dino.rsd")
RSD files	\PSXGRAPH\DATA\RSD\dino.ply, dino.mat, dino.grp	
TEX[0]	dino0.tim	Texture filenames
TEX[1]	dino1.tim	
TEX[2]	dino2.tim	
TEX[3]	dino3.tim	
TEX[4]	dino4.tim	
TEX[5]	dino5.tim	
POLYGON	2724	no. of polygons
VERTEX	1376	no. of vertices
NORMAL	2671	no. of normal lines
MATERIAL	2592	no. of materials
RANGE	(-180, -210, -1690) - (180, 580, 290)	maximum and minimum range values (x, y, z)

Table (Part 1): Example rsdlink Output

(See part 2, below.)

2 - RSD			2nd RSD ("box.rsd")
RSD FILES	\PSXGRAPH\DATA\RSD\box.ply, box.mat, box.grp		
POLYGON	12		
VERTEX	8		
NORMAL	12		
MATERIAL	1		
SCALE	128		scaling factor : rounded to 2^n
RANGE	(-6400, -6400, -6400) - (6400, 6400, 6400)		Maximum and minimum range values (x, y, z)
TMD			
OUTPUT TMD	a.tmd		Output filename
TMD HEADER		(12 bytes)	TMD file header size
OBJECTS	2	(56 bytes)	Total no. of RSDs
PRIMITIVES	2736	(65640 bytes)	Total no. of primitives
	12	Flat Colored Triangles	
	136	Gouraud Colored Triangles	Breakdown for each mode
	2434	Flat Textured Triangles	
	154	Gouraud Textured Triangles	
VERTICES	1384	(11072 bytes)	Total no. of vertices
NORMALS	2683	(21464 bytes)	Total no. of normal lines
Total File Size: 98244 bytes			Output file size

Table (Part 2): Example rsdlink Output

(See part 1, above.)

Example 4: Sample output with '-info' option

When the -info option is enabled, it is possible to see what the converted TMD data looks like.

```
C:\> rsdlink -info box
```

Output		Description
INPUT RSDS	1 object(s)	No. of objects in the TMD file
RSD[0]	"box"	Name of each RSD
TOTAL VERTICES	8	Total no. of vertices
TOTAL NORMALS	12	Total no. of normal lines
TOTAL PRIMITIVES	12	Total no. of primitives
Box		
FLAT TEX 3-POLY(0x24000507) LIGHT: ON = 0		
Vert-0: (-150, -150, -150) (#2)		
Vert-1: (150, -150, -150) (#6)		
Vert-2: (-150, 150, -150) (#0)		
Norm-0: (0, 0, -4096) (#0)		
UV 0-2: (0 0) (47 0) (0 47)		
Pixel mode : 4bit CLUT : (x y)=(0 480)		
Texture Page: 10 Texture No. : 0		
FLAT TEX 3-POLY(0x24000507) LIGHT: ON = 1		

For each primitive, the following information is displayed:

- [the polygon number], flat or Gouraud shading (FLAT/GOURAUD),
- presence of absence of texture (TEX/NO TEX), semi-transparency ON/OFF(TRANS),
- two-sided or one-sided polygons (TWO-SIDED), gradation (GRADATION),
- primitive type (3-POLY, 4-POLY, LINE, SPRITE), primitive header in hexadecimal notation (0x...),
- light source calculation ON/OFF (LIGHT: ON/OFF).

Next, the coordinate values of each of the vertices of that primitive are shown as follows:

```
 Vert-0: (-150,-150,-150) (#2)
```

'(#...)' being the vertex number used in the PLY file.

Normal lines are displayed in the same way:

```
Norm-0: (0,0,-4096) (#0)
```

(With RSD, normal lines are usually normalized to a size of 1 [in this case (0, 0, -1.0]), but with TMD the values shown above occur because the floating point number 4096 is considered as having the value 1.)

Finally, material information such as UV coordinates and colors of the textures are displayed.

rsdv.bat (RSD Previewer)

rsdv.bat is an RSD data previewer which displays RSD data on a TV monitor using the Net Yaroze PlayStation. **rsdv.bat** is a DOS batch program.

Usage

```
C:\>rsdv  RSD_file
```

The PlayStation cannot directly process RSD data, so the **rsdv** command first converts the RSD file passed as an argument into a TMD file using the **rsdlink** command. The TMD file and the corresponding TIM file are merged into a single file which is transferred to the PlayStation. Then the RSD preview program (**rsdview**) is transferred to the PlayStation and the previewer displays the file on the PlayStation's TV monitor.

Using the Tool

The PlayStation Controller can be used to explore the model. (See Controller functions shown below.)

Note: Before using the **rsdv** tool, check to see that:

- the PC and the PlayStation are connected by the communications cable DTL-H3050,
- the Net Yaroze PlayStation boot disk is in the Net Yaroze PlayStation,
- the power is switched on.

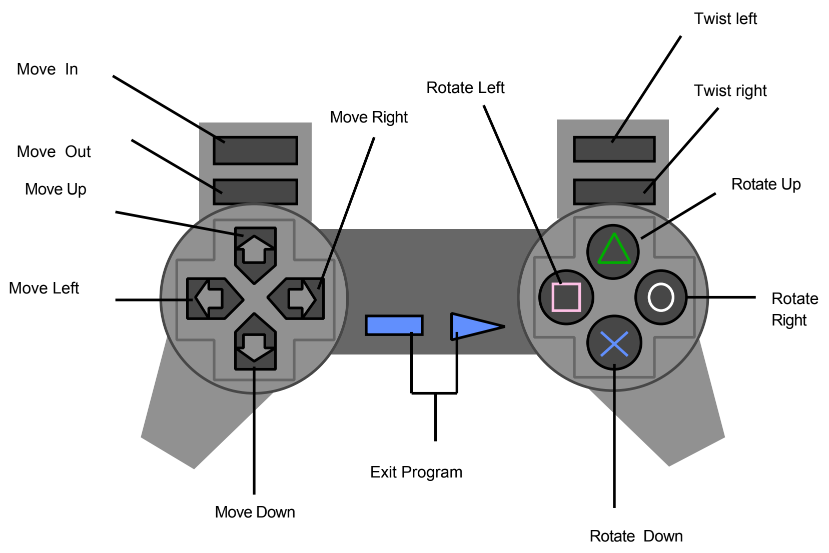


Figure: Controller Button Functions

timutil.exe (TIM utility)

timutil is a Windows application which converts files between each of the following bitmapped formats: PlayStation TIM, Windows BMP, Macintosh PICT, and ordinary RGB.

Usage

To launch **timutil**, double-click on its icon from the Windows File Manager or Windows Explorer. With the application running, perform the following steps.

1. Select a bitmap file by choosing 'Open...' from the File menu.
2. The Parameter Settings window will appear.

.

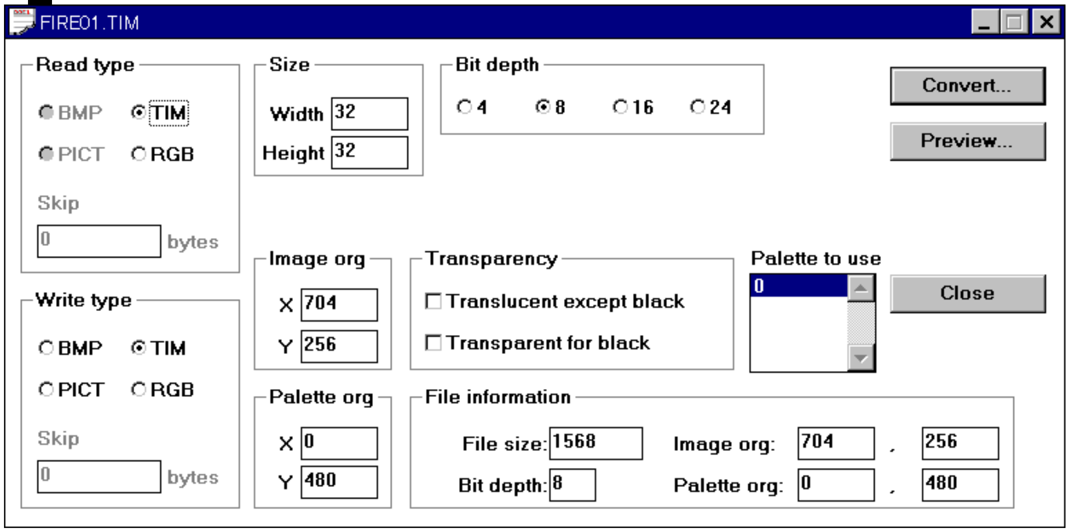
3. Change these parameters as needed and press the 'Convert...' button or select 'Save as...' from the File menu. The Save File dialog box will be displayed.

4. Specify the name for the converted file in this dialog box.

5. The file will be converted and saved under the specified name.

Alternatively:

Drag the RSD file to be converted from Windows Explorer or File Manager and drop it into the **timutil** window.



FIRE01.TIM

Read type: BMP [✓ TIM] PICT RGB
Skip: 0 bytes

Write type: BMP [✓ TIM] PICT RGB
Skip: 0 bytes

Size: Width 32, Height 32

Bit depth: 4 [✓ 8] 16 24

Image org: X 704, Y 256

Palette org: X 0, Y 480

Transparency: ☐ Translucent except black ☐ Transparent for black

Palette to use: 0

File information: File size: 1568; Bit depth: 8; Image org: 704 , 256; Palette org: 0 , 480

Figure: The Parameter Settings Window

Description of Each Item in the Parameter Settings Window

Read type

Displays the format of the input file to be read. When this option is set to TIM, BMP, or PICT, RGB can also be selected. In this case, the TIM, BMP, or PICT header information is ignored, and the tool is forced to read the data as RGB data.

'TIM'

Selected when the input file is a PlayStation TIM format file.

'BMP'
Selected when the input file is a Windows BMP format file.

'PICT'
Selected when the input file is a Macintosh PICT format file.

'RGB'
Selected when the input file was neither a TIM, BMP, nor PICT format file.

Skip
Used to specify the number of bytes to skip when the input file format is 'RGB'.

Write type
Selects the file format for the converted (output) file.

'TIM'
Select this to convert to PlayStation TIM format.

'BMP'
Select this to convert to Windows BMP format.

'PICT'
Select this to convert to Macintosh PICT format.

'RGB'
Select this to convert to ordinary RGB format.

Skip
Used to specify the number of bytes to zero-fill when the output file format is 'RGB'.

Size
Used to specify the byte arrangement when the input file is 'RGB'. This information is essential for RGB image data interpretation. Reports an error if: the size of the image data (calculated from the values entered here and the value entered in the 'skip' box) is larger than the size of the input file. Displays a warning if the size of the image data is smaller than the size of the input file.

Image org
This sets the coordinates of the origin of the PlayStation frame buffer image for a TIM output file.

Palette org
This sets the coordinates of the origin of the PlayStation frame buffer image palette (CLUT) for a TIM output file when the bit depth is 8 or less.

Transparency
Used to specify the transparency when the file format is TIM, and the bit depth is 16 or less.

When 'Translucent except black' is selected, the transparency control bit is set to '1' for all palette entries which have at least one non-zero R, G, or B value.

When 'Transparent for black' is selected, the transparency control bit is set to '0' for all palette entries in which the R, G, and B values are all '0'. If this is not selected and if the R, G, and B values are all '0', the color will be an opaque black.

Bit depth
Specifies the number of bits per pixel in the output file.
When the output format is BMP only the values 4, 8 and 32 can be selected. When the output format is PICT, only 4, 8 and 16 can be selected, and when it is RGB only 24 can be selected.

Palette to use
When the input file has more than one palette and the output format is TIM, this selects which palette(s) will be used in the converted (output) file. The palette(s) selected here are also used when the image is displayed using the “Preview…” button.
When the ‘Link to Palette org ‘ option is enabled for ‘Palette to use’ in the ‘Setup’ command from the File menu, the Y coordinate of the origin of the selected palette is automatically increased or decreased to match how far the selected entry is from the top of the write palette list.

File information
Displays the size and pixel depth of the input file.
For an input file in TIM format, the image origin is displayed. For an input file with a bit depth of 8 or less, the palette origin is displayed.

Convert…

Performs format conversion according to the specified parameters.
Enter the name for the converted file in the Save File dialog box. The default directory is the current directory.

This function is the same as 'Save As...' from the File menu.

Preview…
Reads and displays the specified input file.
If the bit depth of the input file is larger than the depth of the display being used, colors will be approximated.

Close
Closes the current window.
This function is the same as 'Close' from the File menu.

The Menu Bar

'Open...' Command ([File] menu)
Opens an existing bitmapped file. Specify the file to open in the Open File dialog box.
The TIM utility supports PlayStation TIM, uncompressed Windows BMP, 32-bit mode, as well as
PICT and ordinary RGB formats which contain one or more bitmaps (the bit depth for the output file is restricted to 4, 8, 16, and 24 bits).
RSD format model data can also be selected. In this case all TIM files specified in the model data are opened.

'Close' Command ([File] menu)
Closes the current active window.

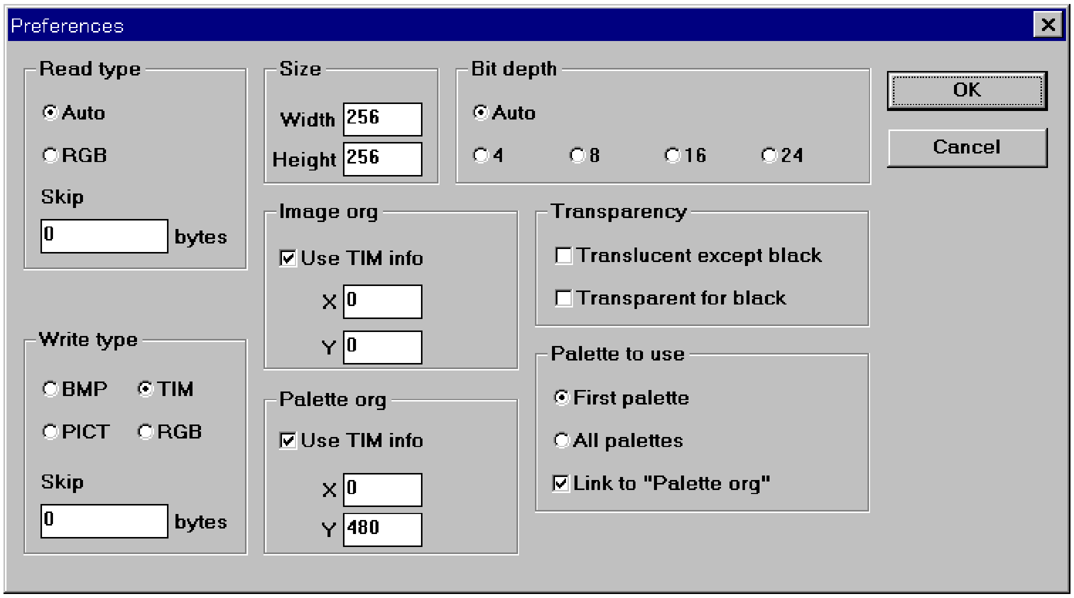
'Save As...' Command ([File] menu)
Saves the bitmap being worked on with a new filename. In the File Save dialog box, enter a suitable name for the converted file and save it. Conversion will be performed according to the specified parameters.

'Save All Files...' Command ([File] menu)

Saves all bitmaps being worked on. When the Directory Selection dialog box is displayed, specify the directory where the output files will be written. Format conversions will be performed for
each bitmap according to the specified parameters. The filenames will be the same as the original filenames, except that the filename extensions are replaced by extensions appropriate to the output format.

'Setup...' Command ([File] menu)

Specifies the initial values displayed in the Parameter Settings window when a file is opened. (As shown below.)



Preferences

Read type: [✓ Auto] RGB; Skip 0 bytes

Size: Width 256; Height 256

Bit depth: [✓ Auto] 4 8 16 24

Image org: [✓ Use TIM info]; X 0; Y 0

Transparency: ☐ Translucent except black; ☐ Transparent for black

Write type: BMP [✓ TIM] PICT RGB; Skip 0 bytes

Palette org: [✓ Use TIM info]; X 0; Y 480

Palette to use: [✓ First palette] All palettes; [✓ Link to "Palette org"]

Figure: The Setup Dialog Box

Notes on The Setup Dialog box

· When 'Read type’ is set to 'Auto', the file format displayed in the Parameter Settings window will match the format of the input file. When 'Read type’ is set to 'RGB', the format is set to RGB regardless of the type of file. The same applies to 'Bit depth'.

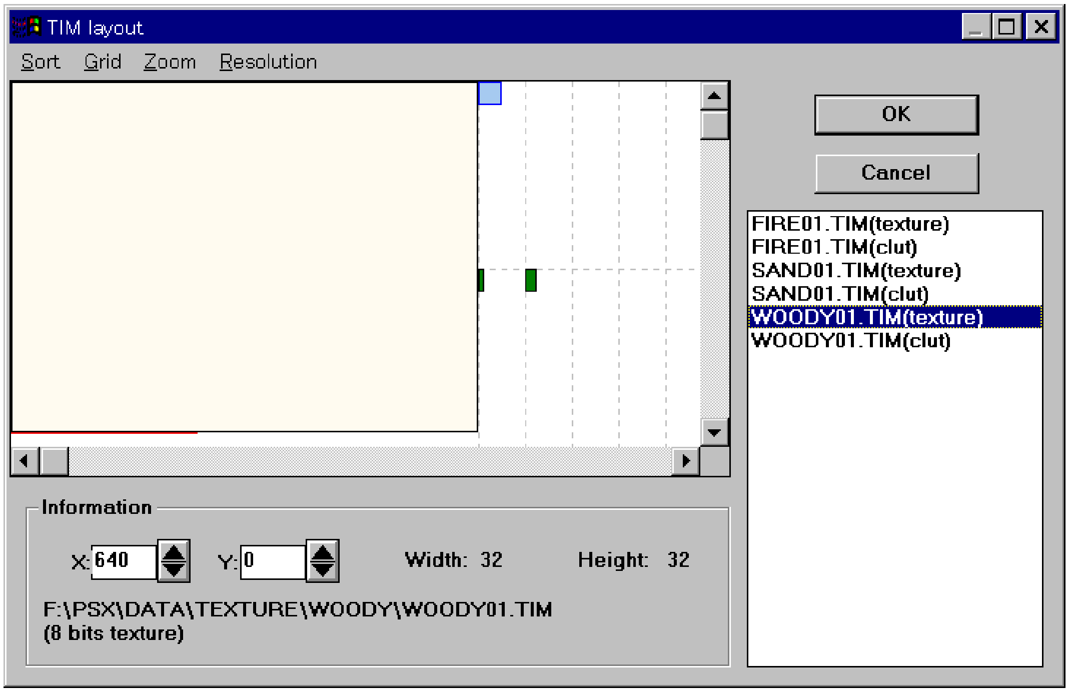
- When the read format is not TIM or when 'Use TIM info' is not checked, the utility uses the values entered for 'Image org' and 'Palette org' as the conversion parameter default values.
- For an input TIM file with more than one palette (CLUT), there is a palette list in the Parameter Settings window to select the required palette(s). Set a default selection state for this palette list using the 'Palette to use' option.

  If 'First palette' is selected, only palette 0 is used. If 'All palettes' is selected, all the palettes will be selected.

  If the 'Link to Palette org' option (under 'Palette to use') is checked and 'First palette' is selected from the write palette list, the palette origin will be set to the Y coordinate of the palette origin obtained from the input TIM file, plus 1.
- All parameters not described above are simply used as the initial values for each item in the Parameter Settings window.
- The items set up in this dialog box are saved when the TIM utility is closed.

'TIM Arrangement...' Command ([Window] menu)

The 'TIM Arrangement' window indicates graphically where the currently open TIM format file(s) are located within the frame buffer.



FIRE01.TIM(texture)
FIRE01.TIM(clut)
SAND01.TIM(texture)
SAND01.TIM(clut)
WOODY01.TIM(texture)
WOODY01.TIM(clut)

X: 640 Y: 0 Width: 32 Height: 32

F:\PSX\DATA\TEXTURE\WOODY\WOODY01.TIM
(8 bits texture)

Figure: the TIM Arrangement Dialog Box

The 'TIM Arrangement' window is roughly divided up into the following four areas:

- A frame buffer image area
- An information area
- A selection list area
- A menu bar

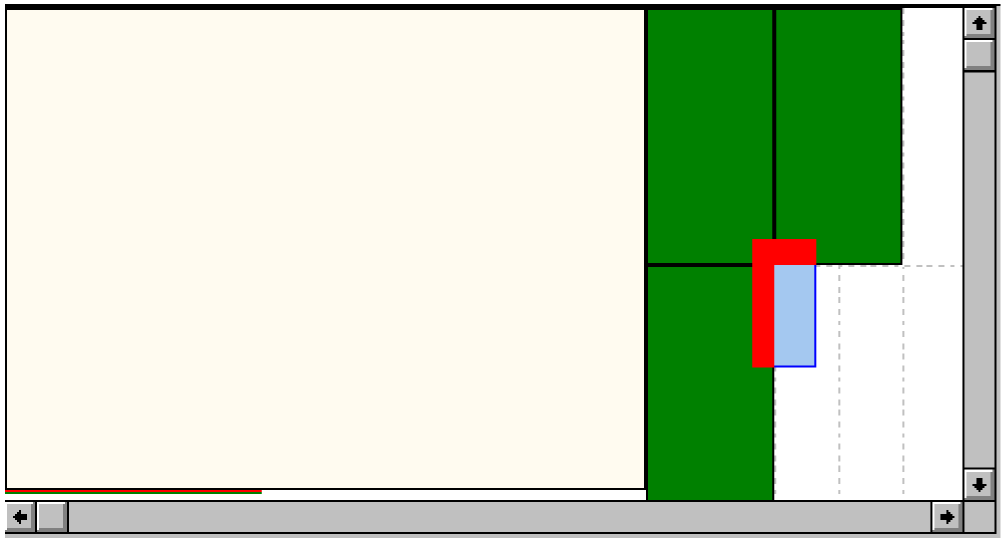


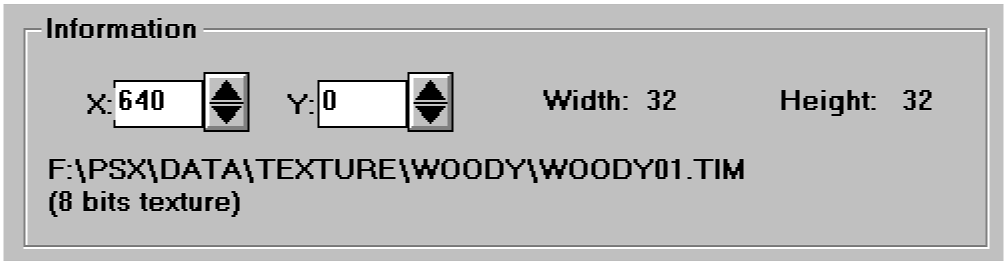
Figure: The Frame Buffer Image Area

This area displays a graphical image of the PlayStation frame buffer.

In the figure above,

- Green, red and blue rectangles on the right represent the images and palettes (CLUTs) of the TIM file that is currently open. These objects are located at the x and y coordinates where they are stored in the frame buffer.
- The two white rectangles on the left represent the screen display areas.
- The dotted lines represent the texture page boundaries. The texture page boundaries are the x and y coordinates where a texture page TIM file *must* start for the PlayStation hardware to display it correctly. The top and left sides of the rectangle should be aligned with these boundaries.
- A green rectangle indicates an image or palette rectangle that does not overlap with other image or palette rectangles or the display region.
- A red rectangle indicates an image or palette that is overlapping or extends beyond the frame buffer.

- ♦ A blue rectangle indicates one which is selected by the mouse. Rectangles can be selected with the left mouse button and can be moved by dragging. Multiple image and palette rectangles can be selected by clicking the left mouse button while holding down the Shift key or the Ctrl key.



Information

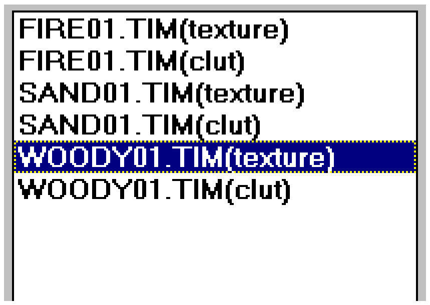
X: 640 Y: 0 Width: 32 Height: 32

F:\PSX\DATA\TEXTURE\WOODY\WOODY01.TIM
(8 bits texture)

Figure: Information Area

The Information window displays information relating to the currently selected image or palette such as the coordinates of the rectangle origin (top left corner), the size of the file, the filename and bit depth. If more than one file is selected, no information is displayed in this window.

The coordinates of the origin can be modified by clicking the up arrow button or by directly entering a numerical value in the text box.



FIRE01.TIM(texture)
FIRE01.TIM(clut)
SAND01.TIM(texture)
SAND01.TIM(clut)
WOODY01.TIM(texture)
WOODY01.TIM(clut)

Fig: Selection List Area

The Selection List area shows a list of currently open images and palettes. This list can also be used to select image and palette rectangles. The currently selected file (the light blue rectangle in the frame buffer image area) will be highlighted. Click on a file on this list to change the selection (hold down the 'Shift' or 'Ctrl' key for multiple selection).

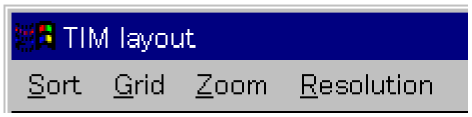


Figure: Menu Bar

The 'Sort' Menu

'Textures'

The upper left corner of each of the textures is aligned to the nearest texture region or display region boundary. The textures can be arranged neatly by first using the mouse for approximate placement and then using this menu command.

'Palettes'

The palettes are all aligned vertically, starting at the bottom end of the current display region. (This option works for NTSC but does not work for PAL where the display height is 256.)

The 'Grid' menu

'None'

Disables the grid function. Textures are arranged at the coordinates specified using the Mouse.

'Texture page'

Textures are moved from the coordinates specified with the Mouse, and aligned in rows at the nearest texture-page boundary. When 'XY' is selected, both X and Y coordinates are aligned to the texture-page boundary. When 'X' is selected, alignment is done only for the X coordinate, and when 'Y' is selected, alignment is done only for the Y coordinate.

'Magnet'

Textures are moved from the coordinates specified with the Mouse, and aligned with the edge of the nearest texture area or display area. When 'XY' is selected, both X and Y coordinates are aligned {with the texture-page boundary). When 'X' is selected, alignment is done only for the X coordinate, and when 'Y' is selected, alignment is done only for the Y coordinate.

The 'Zoom' Menu

Sets the scaling factor for image display in the 'frame buffer image area'. This command can be used when selecting and moving small textures and palettes that would be difficult to select under the standard zoom factor.

The 'Resolution' Menu

Changes the display resolution used on the PlayStation's display. Changing this setting also changes the range of the display region.

Press the "OK" button to have the edited frame buffer image reflected in the Parameter Settings window. If "Cancel" or "Close" is selected, the image and palette modifications will be cancelled.

Notes

- The following file types cannot be opened for input: compressed format BMP, JPEG, compressed PICT, 32-bit PICT, and PICT files that do not contain at least one bitmap.
- The following files cannot be output: compressed format BMP, JPEG, and compressed PICT files.
- The bit depth of output files is limited to 4, 8, 16 and 24 bits.
- Colors will be approximated when conversion is accompanied by a reduction in bit depth, e.g., when 16-bit data is converted to 8-bit data. The approximation will be performed by approximating the R, G, and B data to a uniformly assigned color map rather than using a method such as color compression. Thus, color quality may decrease.
- As a rule, the input file cannot be overwritten. Overwriting is possible, however, when TIM format is used for both input and output formats, and if the image origin or the palette origin is changed.

timv.bat

timv is a TIM (image data) previewer. **timv** displays TIM data on a TV monitor using the PlayStation.

Usage

```
timv  TIM_file
```

Before starting this tool, ensure that the PC and Net Yaroze PlayStation are connected by the Net Yaroze communications cable, the Net Yaroze PlayStation boot disk is in the PlayStation, and the power is turned on.

When this command is executed, the TIM file supplied as the argument is transferred to the PlayStation. Then the TIM previewer (**timview**) is transferred to the PlayStation and the previewer starts. Use the Controller to operate the previewer with the functions shown below.

Note that the TIM previewer (**timview**) only works when the baud rate is set to 9600 (the default baud rate with no memory card in the right-hand slot).

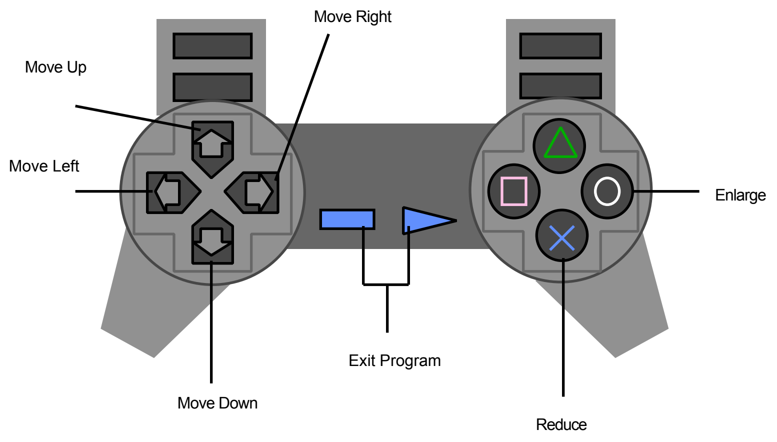


Figure: Controller Button Functions

15

Sound Tools

The Net Yaroze sound tools consist of data converters and players. Data converters convert sound data (waveform and score data) created using commercial tools into the PlayStation format. Players permit sound data to be played back on the PlayStation through a TV monitor so it can be verified. The data converters can also be used to create and edit waveform data.

The dedicated PlayStation sound data formats and their applications are listed below.

· SEQ data:Score data	
· VAG data:	Single waveform data
· VAB data:	Sampled bank data
· VH data:	Sampled data (attribute part)
· VB data:	Sampled data (waveform part)

The Net Yaroze sound tools all run from MS-DOS on the PC that is connected to the PlayStation. The remainder of this chapter provides a description of these tools.

smf2seq.exe
smf2seq takes Standard MIDI File (SMF) format-1 data created using commercial sequencer software (e.g. a score creation editor), and converts it into PlayStation SEQ data.

aiff2vag.exe
aiff2vag takes 16-bit straight PCM data or AIFF (Audio Interchange File Format) data created using commercial waveform editing software, and converts it into PlayStation VAG data.

mkvab.exe
mkvab takes PlayStation VAG data and attribute definition files, and generates PlayStation VAB data.

vabsplit.exe
vabsplit splits a PlayStation VAB data file into a VH component and a VB component.

sndplay.bat
sndplay plays back PlayStation SEQ data on the PlayStation using the standard waveform source found on the Net Yaroze PlayStation boot disk.

vabplay.bat

vabplay plays back PlayStation SEQ data on the PlayStation using a waveform source created on the PC.

smf2seq.exe

smf2seq converts Standard MIDI File (SMF) format-1 data into PlayStation score data.

Usage

```
  smf2seq [options] SMF-files ...
```

smf2seq creates a SEQ file (i.e. a PlayStation score data file) from an SMF file provided as an argument. More than one SMF file can be specified for batch conversion. The '.smf' file extension of the input SMF file(s) can be omitted. The output filenames will be written with '.seq' extensions.

Options

-Q

Conversion is performed in Quiet mode. No warning messages are displayed.

-V

Conversion is performed in Verbose mode. A list of control changes and meta-events used in the SMF data are displayed.

-B

Forcibly deletes bank changes that were used in the SMF data.

Restrictions

The current version of **smf2seq** has the following restrictions.

_ SMF format-0 is not supported.

_ Files of 64 KB or more are not supported.

_ The control changes listed below are supported.

Bank Select(#0)

Data Entry(#6)

Main Volume(7)

Panpot(10)

Expression(11)

NRPN(98, 99)

aiff2vag.exe

aiff2vag converts Audio Interchange File Format data (AIFF), windows WAV Format data or 16-bit straight PCM data (without header(s)) into PlayStation waveform data files. All data must be either 16-bit straight or monaural waveform data.

Usage

```
  aiff2vag [options] aiff/WAV-files...
```

aiff2vag creates a PlayStation VAG file (*.VAG) from an AIFF, WAV format or 16-bit straight PCM format file. (The PlayStation's VAG file is a compressed waveform data file.) More than one input file can be specified for batch conversion. The '.aif' or '.wav' file extension can be omitted.

Options

-1
A waveform containing loops will be unconditionally encoded as a waveform without loops.

-L
A waveform without loops will be unconditionally encoded as a waveform with loops.

-R fs
Specifies the sampling rate for 16-bit straight PCM data input. *fs* is given in Hertz.

-E
Endian conversion (byte ordering) will not be performed.

Restrictions

The current version of **aiff2vag** has the following restrictions.

- Only 16-bit uncompressed data is supported. 4-bit, 8-bit and compressed format data are not supported.
- Only monophonic data is supported. Channels should be converted separately when converting data from a stereo source.

mkvab.exe

mkvab generates a VAB (PlayStation sampled bank) data file from an attribute definition file and one or more specified VAG (PlayStation format waveform) data files. **mkvab** can also analyze an existing VAB file to re-create the definition file originally used to create that VAB file. Note that VAG data must be created using **aiff2vag**.

Usage

1) mkvab -f def_file -o vab_file vag_files.....

Parses a user-created definition file (def_file) and outputs a VAB file (vab_file) containing VAG files (vag_files) in the order they appear on the command line.

Example

```
mkvab -f rob1.def -o rob1.vab boing.vag grunt.vag
```

will create vab file "rob1.vab" which consists of a vab header which is defined by "rob1.def" and a vab body containing 2 vags - "boing.vag" and "grunt.vag".

2) `mkvab -r vab_file [-o def_file ]`

Parses a VAB file and either prints the output to the screen (if output file option not entered) or outputs the VAB definition file to def_file.

Options

-f def_file

Specifies the definition file *def_file* used to create the attribute table(s).

-r vab_file

Analyses a VAB file and outputs attribute definition files.

-o out_file
Specifies the output file.

-o def_file
Specifies the definition file to be output when also using the -r vab_file option.

Restrictions

The current version of **mkvab** has the following restrictions.

_ File size cannot be user-specified. The size of the VAB file and the size of each of the VAG files is calculated automatically. Any specified value in the attribute definition file is ignored.

_ ADSR linear mode is not supported; only exponential mode.

_ DOS allows a maximum number of 254 characters on a command line. Due to this limitation, the maximum number of VAGs which can be contained in a VAB created by **mkvab**, is 119. Use short file names (no extensions necessary) for def_file, vab_file, and vag_files to maximize VAG entry.

def_file Format

The def_file is a text file containing four main sections (in sequence), as follows:

VabHdr	One user-specified section per definition file. See table n below.
ProgAtr X	One ProgAtr section for each specified program, where X=program number, ranging from 0 (1st program) - 127 (maximum program number). See table n below.
ToneAtr X Y	One ToneAtr section for each specified tone contained in each specified program, where X=program number ranging from 0 (1st program) - 127 (maximum program number) and Y=tone number ranging from 0 (1st tone in program) -15 (maximum tone number in program). ToneAtr sections should be grouped consecutively based on program number, i.e. X=0 Y=0, X=0, Y=1...,X=1 Y=0, X=1 Y=1...,X=last program used Y=last tone used in last program used. See table n below.
vsize	One vsize section which is automatically calculated by **mkva**b; it does not need to be input.

Input for each section of the definition file consists of a series of labels and related values in the format:

```
VabHdr
          label = value
          label = value
          .
          .
          .
ProgAtr 0
          label = value
          label = value
          label = value
          .
          .
          .
ProgAtr (num programs-1)
          label = value
          label = value
          label = value
ToneAtr 0 0
          label = value
          label = value
          label = value
ToneAtr 0 1
          label = value
          label = value
          label = value
ToneAtr 1 0
          label = value
          label = value
          label = value
          .
          .
          .
ToneAtr (number programs-1, number tones in last program-1)
          label = value
          label = value
          label = value
```

Table: VabHdr Section of def_file

Label	Value	Explanation
VabHdr	**-no value-**	Master attributes of VAB.
form	**'VABp'**	Format identifier
ver	**0x07**	Format version number
id	**0**	VAB id (always 0)
fsize	**0**	File size (mkvab calculates this automatically.No input necessary).
ps	**1~128**	Total no. of programs in the VAB data
ts	**1~2048**	Total no. of tones in the VAB data
vs	**1~254**	Total no. of VAGs in the VAB data

Table: ProgAtr section def_file (equivalent to instrument level)

Label	Value	Explanation
ProgAtr X	-no value-	Program attributes, where X=program number (0=1st, 127=max)
tones	0~15	No. of tones in the program
mvol	0~127	Program volume value (0=min, 127=max)
mpan	0~127	Program panning value (0=all left, 64=center, 127= all right)

Table: ToneAtr section def_file

Label	Value	Explanation
ToneAtr X Y	-no value-	Tone Attributes, where X= program number (0=1st, 127=max), Y= tone number (0=1st, 15=max)
prior	0~127	Tone priority level - larger values have higher priority
mode	0,4	Sound source mode (0=normal, 4=reverberation effect***)
vol	0~127	Tone volume value (0=min, 127=max)
pan	0~127	Tone panning value (0=all left, 64=center, 127=all right)
center	0~127	Center note (in semitone units)

shift	0~127	Center note fine tuning
min	0~127	Note limit minimum value. Must be <=max (see next entry)
max	0~127	Note limit maximum value. Must be >=min (see previous entry)
pbmin	0~127	Maximum value for downwards pitchbend (in semitones)
pbmax	0~127	Maximum value for upwards pitchbend (in semitones)
ar	0~127	Attack rate (change rate after key-on until highest volume reached)
dr	0~15	Decay rate (change rate from the highest volume to the sustain level)
sr	-127~127	Sustain rate (change rate after the threshold level has been reached. to enter negative values, write the value followed by a minus sign)
rr	0~31	Release rate (change rate of attenuation after key-off)
sl	0~15	Sustain level (threshhold level)
prog	0~127	Program number containing the tone
vag	1~254	VAG number referenced by the tone

Table: VSize section def_file

Label	Value	Explanation
vsize	-no value-	Sizes of VAGs in VAB file. Automatically calculated by mkvab and only output by usage 2 of mkvab. Input will be ignored.
	size vag 1	filesize of vag 1. Automatically calculated.
	size vag 2	filesize of vag 2. Automatically calculated.
	...	
	size last vag	file size of last vag in VAB

*** To obtain a reverberation effect, the reverberation must be set using a separate sound function.

Supplementary Notes

ADSR (envelope) rates, i.e. the rates for attack, decay, sustain and release, can be set individually. Linear or exponential function curves can be specified for the rate of change. The sustain level can also be specified.

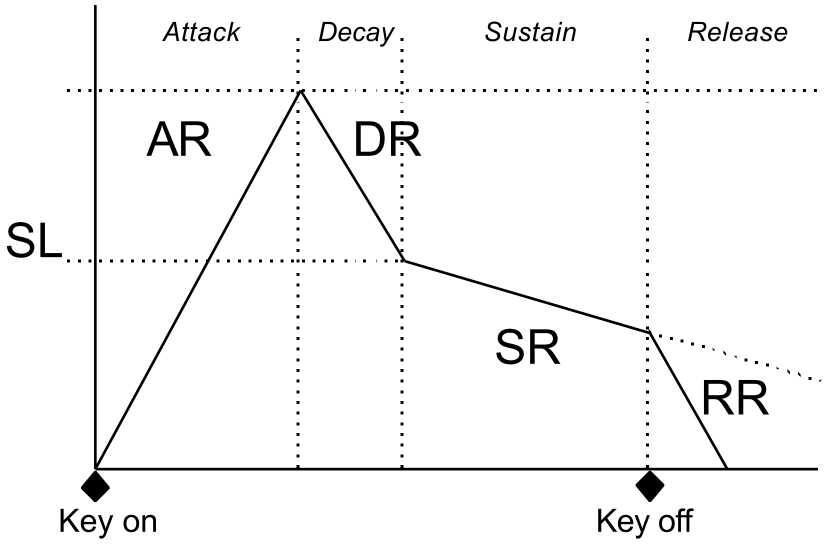


Figure: ADSR Concept Diagram

vabsplit.exe

vabsplit splits sampled bank PlayStation VAB data previously created using **mkvab** into an attribute table part (VH) and a waveform data part (VB). VH data must be memory-resident during playback of music, however, VB data need not be resident in memory once it has been transferred to the SPU.

Usage

```
vabsplit vab-files...
```

vabsplit splits PlayStation VAB data into a sampled bank attribute table part (VH) and a waveform data part (VB). More than one VAB file can be specified to perform batch conversions. The '.vab' file extension can be omitted.

There are no options or restrictions associated with **vabsplit**.

Sound Players

Running the Sound Players

When the Net Yaroze system is installed, the batch files, 'seqplay.bat' and 'vabplay.bat', are placed in the ‘command’ directory of the PC, while sample sound files are placed in the ‘data’ directory. Included among the sample files is 'sample1.seq', which is described in the example below.

Follow the steps below to play back the sample file using the PlayStation’s sound player.

Turn on the PlayStation after making sure that the PC and the PlayStation are connected via the Communications cable and that the boot disk is mounted in the PlayStation. Type the following command from the 'data\sound' directory to load the standard waveform source files (STD0.VH and STD0.VB) from the boot disk and play back the sample SEQ file.

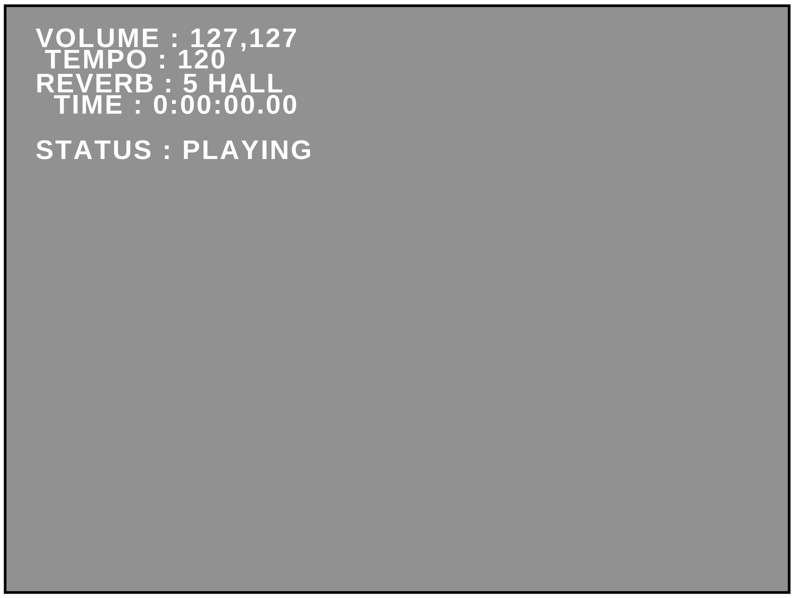
```
C:\> seqplay sample1.seq
```

You can play sound data using a waveform set other than the standard waveform source file by transferring the data via the serial port with the following command.

```
C:\ > vabplay my.seq my.vh my.vb
```

Using the Sound Player

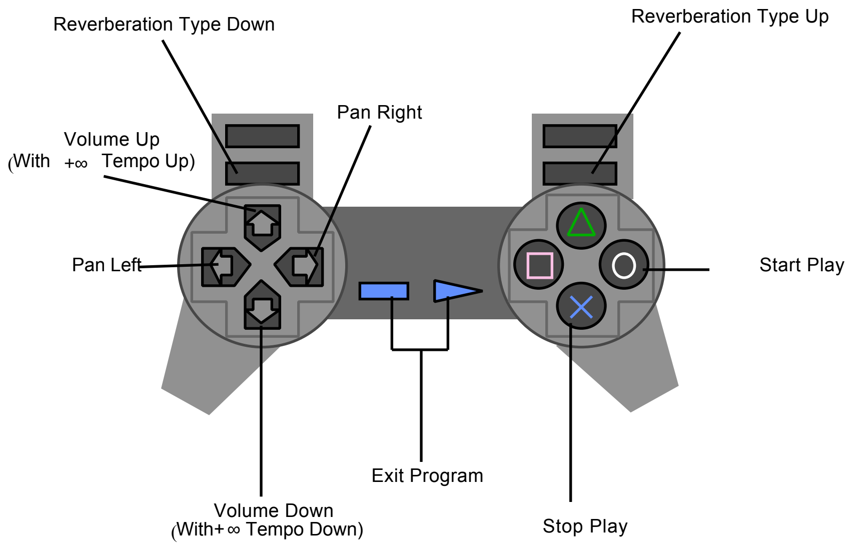
If the program is running correctly, the PlayStation will play back music and output an image to the TV monitor, similar to the one shown below.



```
VOLUME : 127,127
 TEMPO : 120
REVERB : 5 HALL
  TIME : 0:00:00.00

STATUS : PLAYING
```

Use the Controller buttons shown in the diagram below to control the sound playback.



Waveform Data Specifications

Sampled waveform data can be used in the same way as MIDI waveform data. For more information, please check the Net Yaroze Web site.

16

Programming Tools

The Net Yaroze programming tools consist of the GNU C compiler and associated utilities. These are shown in the table below.

Compiler	gcc.exe
Linker	ld.exe
Debugger	gdb.exe
Librarian	ar.exe
Maintenance utility	make.exe
Symbol information remover	strip.exe
Object Manager	nm.exe
Assembler	as.exe, etc.
Others	size.exe

This chapter describes only the most commonly used tools such as **gcc**, **ld**, **strip** and **make**. For information on the remaining tools, please refer to the documentation that accompanies the GNU C compiler, or to related commercially available sources such as those mentioned in the Additional Reading List at the end of the *Start-Up Guide*.

The gcc Compiler

The **gcc** compiler creates object and executable files from C source files.

Filename Extensions and Actions

gcc is actually a front-end processor which can call one or more tools used to link and compile the input C source files. Which tools are called by **gcc** depends on the extensions of the input files and the options that are supplied on the command line. The table below shows a list of file extensions and what tools **gcc** will call when these file types are provided as input.

Note that files with filename extensions that the compiler cannot recognize are treated as object files. These files are passed directly to the linker as arguments.

Extension	Tools Called and Calling Order
.C	C preprocessor → C compiler → assembler → linker
.I	C compiler → assembler → linker
.CC	C preprocessor → C++ compiler → assembler → linker
.CPP	C preprocessor → C++ compiler → assembler → linker
.II	C++ compiler → assembler → linker
.S	Assembler → linker
others	Linker

Table: Tool Calling Order

gcc can also be used with various options to halt or control the execution flow during preprocessing, linking, assembling or compiling.

The name of the object file or executable file that is output by **gcc** depends on the compiler options you have specified.

Usage

To execute **gcc**, type the following command from the MS-DOS prompt.

```
gcc  [-option]  <source files>...
```

[-option]

Options are preceded by a '-' (hyphen) and multiple options are separated by spaces. Options are case sensitive, so '-o' and '-O' have different meanings. The name of the source file is entered after the option(s) and is preceded by a space.

The following sections give examples of some of the commonly used options.

Examples (using -c or -o options)

· To compile a source file called 'test.c' and create an object file 'test.o', use the following command:

```
gcc -c test.c         // Creates the object file 'test.o' from test.c.
```

· To compile 'test.c' and create the executable file 'test', use the following command:

```
gcc -o test test.c    // Creates the executable file ‘test' from test.c.
                         Note that the ‘test’ after the ‘-o’ is the
                         argument for the ‘-o’ option (the output
                         filename), and is not the name of the source
                         file.
```

· To compile more than one source file, the individual filenames should be specified on the command line, separated by spaces. In this example, ‘test1.c’, ‘test2.c’, and ‘test3.c’ are compiled, and object files corresponding to each source file are generated.

```
gcc -c test1.c test2.c test3.c
```

Options

There are a number of options for **gcc** and only the most important ones are described here. Please refer to the documentation that accompanies the GNU C compiler, or to related commercially available sources such as those mentioned in the Additional Reading List at the end of the *Start-Up Guide,* for additional information.

Compiler-Specific (gcc) options

· **Default (no options)**
With no options specified, **gcc** calls the linker and creates an executable file. The default filename for the executable file is 'a.out' if no other name is specified.

```
gcc test.c                //output to a.out
```
or
```
gcc test.c -o temp3.exe      //output to temp3.exe
```

· **Execute the preprocessor only (-E)**
When the -E option is specified, only preprocessing is performed without compiling or linking. If no output filename is specified, the results will be displayed on the standard output (the screen).

```
gcc -E test.c                //Output to screen
```
or
```
gcc -E test.c > test.pre     //Output to file test.pre
```

- **Output Assembler Code (-S)**

  When the -S option is specified, the compiler stops after assembly code is generated and linking is not performed. An assembler file is generated when a C source file is provided as input. If no output filename is specified, the output file will have the same name as the input file but with a '.s' extension.

  ```
  gcc -S test.c                //output to test.s
  or
  gcc -S -o temp1.s test.c     //output to temp1.s
  ```

- **Output an Object File (-c)**

  When the '-c' option is specified, compiling and assembling is performed, and an object file is generated, however, no linking is performed. If no output filename is specified, the output file will have the same name as the input file but with a '.o' extension.

  ```
  gcc -c test.c                //output to test.o
  or
  gcc -c test.c -o object1.o           //output to object1.o
  ```

- **Support ANSI (-ansi)**

  When the '-ansi' option is specified, the compiler supports all ANSI standard C programs. This option turns off certain features of GNU C that are incompatible with ANSI C where a warning or error would normally be generated.

  ```
  gcc -ansi test.c             //compile an ansi C file and output to a.out
  ```

- **Create Debugging Information (-g)**

  When the '-g' option is specified, debugging information is generated and embedded in the executable file. This option should always be used when compiling a program that will be debugged later using **gdb**, the GNU debugger.

  ```
  gcc -g test.c -o test        // output to test - with debugging
                           information
  ```

Optimization Options

These options tell the compiler to improve the efficiency of the code that it creates**.**

Note: It is not advisable to use optimization options with the debugger option (-g).

- **No optimization (-O0)**

  When "-O0" is specified, no optimization is performed. This is the default setting and generally does not need to be explicitly specified.

- **Standard Levels of Optimization (-O, -O1, -O2, -O3)**

  The '-O', '-O1', '-O2' or '-O3' options specify levels of optimization. '-O' specifies the lowest level of optimization while '-O3' is the highest.

  ```
  gcc -O2 test.c
  ```

Linker Options

The following options are available for controlling the linker.

- **Do Not Link the Standard Library (-nostdlib)**

  When "-nostdlib" is specified, the Net Yaroze standard library is not linked, and only object files which are explicitly specified on the command line are sent to the linker.

  ```
  gcc -nostdlib test.c      //the standard library is not included in the
                            executable
  ```

- **Specify Library (-l<libname>)**

  The '-l' option is used to specify the name of a library to be explicitly linked in the application. (Note that there is no space between the option and the library name.) This option is used to link user-defined libraries created using **ar**, the librarian utility.

  ```
  gcc -lmylib.lib test.c    //link the library 'mylib.lib' with 'test.o'
                            //when creating the executable
  ```

- **Specify Linker Option (-Xlinker <linker option>)**

  The '-Xlinker' option can be used to pass options directly to **ld**, the linker. However, if the linker option to be specified contains a space, the separate sections of the option must be split up and specified separately with individual '-Xlinker' options.

  ```
  gcc -Xlinker -Map -Xlinker mapfile test.c  // creates a map file
  ```

  Note that -Xlinker is called to specify the option and called again to specify the output file.

or

```
gcc -Xlinker -Ttext -Xlinker 80140000
```

General Options

The following options are available for providing overall control of **gcc**.

- **Warning Messages (-W, -Wall)**

  When either the '-W' or '-Wall' option is specified, warning messages from the compiler will be displayed in response to various events.

  ```
  gcc -W test.c
  ```
  or
  ```
  gcc -Wall test.c          //set maximum warning level
  ```

- **Define Macro (-D<NAME>, -D<NAME=VALUE>)**

  When the '-D' option is specified, the macro identified by '<NAME>' will be asserted during compilation. '<VALUE>' can also be specified to assign a numerical value to the macro. For example:

  Specify the Macro 'DEBUG'

  ```
  gcc -DDEBUG test.c
  ```
  or

  Specify the Macro 'DEBUG' as '0'

  ```
  gcc -DDEBUG=0 test.c
  ```

- **Undefine a Macro (-U<NAME>)**

  When the '-U' option is specified, the macro identified by '<NAME>' will be undefined during compilation.

  Undefine the Macro 'DEBUG'

  ```
  gcc -UDEBUG test.c
  ```

- **Display Detailed Information (-*v*)**

  When the '-v' option is specified, **gcc** will display informational messages during execution.

  ```
  gcc -v test.c
  ```

-o filename

The ‘-o’ option specifies the output filename. Only one file can be specified with ‘-o’. If more than one file is to be compiled, this option should only be used if a single executable is to be generated from the input files.

For example:

```
  gcc test1.c test2.c test3.c -o test
```

which will compile and link the three input source files, and create the executable file ‘test’.

In the example below, more than one source file is specified together with the ‘-c’ option. In this case, linking is suppressed and individual object files will be created. However, the ‘-o’ option will cause these files to be overwritten. In other words, the output file will always contain the results from compiling the last file that was specified (in this case, ‘test’ will always contain the results (object code) from compiling test3.c).

```
  gcc -c test1.c test2.c test3.c -o test
```

The Linker 'ld'

ld takes separate objects and links them to create a single executable.

Usage

ld is executed by entering the following command from the MS-DOS prompt.

```
  ld [-o <output filename>] <obj files>.... [-option]
```

The object files to be linked are specified as arguments.

[-o <output filename>]

When ‘-o’ is specified, the option and output filename must be specified immediately after **ld**.

[-option]

Options are preceded by a ‘-‘ (hyphen). If multiple options are specified, they must be separated by spaces.

The following is a description of the most commonly used options of **ld**.

Options

· **Specify Output File (-o <output filename>)**

When '-o' is specified, the option and output filename must be specified immediately after **ld.** If '-o' is not specified, 'a.out' will be used as the default output filename.

```
ld -o test test.o     //outputs to 'test.o'
```

· **Define Symbol (-defsym <symbol=expression>)**

The symbol to be defined and its value are specified after '-defsym'. The name of the symbol and its assigned value are separated by an "=" (equal sign). "-defsym" and the name of the symbol must be separated by a space.

```
ld -o test test.o -defsym DEBUG=1    //outputs to 'test.o' and defines
                                     'DEBUG' as 1
```

· **Create Mapfile (-Map <mapfile> or -M)**

These options cause the linker to generate a load map, which is a list of external symbols in the program.. The name of the mapfile should be specified after '-Map'. When the "-M" option is used, the load map is sent to standard output.

```
ld -o test test.o -Map mapfile              //outputs object file to
                          'test.o'
                                     and map information to 'mapfile'
```

or

```
ld -o test test.o -M                 //outputs object file to 'test.o' and
                                     map information to screen
```

· **Display Symbol (-y <symbol>)**

When the name of a symbol is specified after '-y', the object file where the symbol is defined or referenced is displayed.

```
ld -o test test.o -y DEBUG           //outputs object file to 'test.o' and
                          displays name of file where DEBUG is defined
```

strip (Symbol Information Remover)

strip is a utility for removing symbol information from an executable file.

Symbol information is used during debugging and program development but becomes unnecessary once the application is released and is to be distributed to other Net Yaroze members. **strip** can be used to remove symbol information and reduce file size.

Usage

strip is executed by entering the following command from the MS-DOS prompt.

```
  strip  [-option]  <file>
```

Options are preceded by a '-' (hyphen). If multiple options are specified, they must be separated by spaces. The executable filename is specified after the options.

For example, the following command removes symbol information from 'test'.

```
  strip test
```

For a description of the available options for **strip**, please refer to the documentation that accompanies the GNU C compiler, or to related commercially available sources such as those mentioned in the Additional Reading List at the end of the *Start-Up Guide*.

The make Maintenance Utility

make is a maintenance utility which automates the building and rebuilding of programs. **make** can be used to automate the process of compiling and linking C source files.

make rebuilds only the individual elements of a program (i.e. the source and object files) that are necessary by using a control file known as a 'Makefile'. The Makefile contains instructions to **make** that describe the steps needed to build the program together with a list of dependency relationships that exist between the elements.

make compares the time stamp of each of the source and object files of the program with the target file it is instructed to create, to determine whether or not an individual element needs to be rebuilt.

In general, the target file will be rebuilt if any one of the source files was changed since the last time the target file was created.

Using Makefiles can eliminate the need to repeatedly enter complex DOS commands and thus allow projects to be developed more efficiently.

As an example, suppose your program consists of three source code files, test1.c, test2.c, and test3.c. Further suppose that after compiling and linking these three files into an executable called ‘test’, you update one of the source files, test3.c.

With a properly constructed Makefile, you would simply issue the command:

```
    make
```

This would automatically recompile test3.c and rebuild ‘test’ without making it necessary to recompile the other two source files which did not change. Thus, the time required to rebuild the executable has been reduced and the build process has been simplified by issuing just a single command.

Usage

To execute **make**, the following command is entered from the MS-DOS prompt.

```
    make [-f makefile] [options] <target>...
```

Options are preceded by a hyphen ('-'). If multiple options are specified, they must be separated by spaces.

The target of **make** is specified after the options. If no target is specified, the first target within the Makefile is built.

The default filename for Makefile is ‘Makefile’. The Makefile can be explicitly specified with the ‘-f’ option.

The following is a description of the most commonly used options of **make**. Please refer to the documentation that accompanies the GNU C compiler, or to related commercially available sources such as those mentioned in the Additional Reading List at the end of the *Start-Up Guide* for additional information.

Options

- **Specify Makefile Filename (-f <filename>)**

  The filename for the Makefile is specified after ‘-f’ and separated by a space. The name of the default Makefile is ‘Makefile’.

```
make                          //assumes existence of 'Makefile'
```
or
```
make -f test1.mak             //runs the Makefile called 'test1.mak'
```

- **Ignore Errors (-i)**

  Specify '-i' to force **make** to ignore all errors. **make** will continue to run even when it detects an error.

  ```
  make -i all                   //continues to run the Makefile, regardless of
                                errors
  ```

- **Display Debugging Information (-d)**

  Specify '-d' to display debugging information.

  ```
  make -d all                   //displays debugging information
  ```

- **Suppress Display (-s)**

  Specify '-s' to force **make** to execute silently. Commands will not be displayed while **make** is running.

  ```
  make -s all                   //make executes silently
  ```

- **Query (-q)**

  Specify '-q' to request **make** to return the status of the target (returns '0' if the target file has been updated, '1' if not).

  ```
  make -q all                   //returns the status of the target
  ```

Makefile

A 'Makefile' is a standard text file which describes a program's construction sequence and dependency relationships. A Makefile consists of *explicit rules* and *implicit rules*. Explicit rules define the commands and relationships needed to create a specific target. Implicit rules define the commands needed to create one file type from another (e.g. a '.o' from a '.c'). Explicit rules are also known as dependency rules. Each of these rule types is described in the sections below.

A target file is any file that the compiler can create, such as an object file. Target files depend on the files that were used to create them. These dependencies form *dependency relationships* which **make** uses when building the target file.

For example, a C source file that is used to generate a particular object file may include a C header file. In this case, a dependency is created because the object file depends on the header file. If the header file is changed, then the object file will become out-of-date and the source file will need to be recompiled. After recompilation, the new object file will reflect the changes that were made to the header file.

Dependency rules in a Makefile describe the dependency relationships that exist between files for a particular target. In addition to rules, Makefiles also contain commands that are needed to build each target file.

The dependencies together with the commands describe to **make** how to build each of the target files.

When a target needs to be built, **make** first searches for dependency rules for that target in the Makefile. If no dependency rules are found, **make** uses implicit rules to build the target.

Dependency Rules

Dependency rules are specified in the Makefile as follows:

```
Targetfile.exe:   <depfile1> <depfile2> <depfile3> <depfile4>
                  command
                  ...
                  command
```

In this example, Targetfile.exe is the name of the target file. The target file is placed at the beginning of the line, followed by a ':' (colon). The target file is followed immediately by a list of source files on which the target file depends. These source files are known as *dependency files*. Dependency files are separated by spaces, and multiple files can be specified. In this example, Targetfile.exe depends on <depfile1>, <depfile2>, <depfile3>, and <depfile4>.

Note that a long list of dependency files can be split over many lines using the backslash, as shown below:

```
Targetfile.exe:     <depfile1> <depfile2> <depfile3> <depfile4>\
                    <depfile5> <depfile6> <depfile7> <depfile8>
```

Dependency files can be either C or C++ source files or header (.h) files.

After the dependency files, the commands that need to be executed to build the target are specified. The commands are tabbed from the beginning of the line and are entered one line at a time in the order of execution.

Note that the command lines must begin with a *single* tab as spaces will cause **make** to report errors incorrectly.

Detailed Explanation

Consider the following dependency rule.

```
Targetfile.exe:     <depfile1> <depfile2> <depfile3> <depfile4>
                    command
                    ...
                    command
```

The relationships between each field, and the resulting actions are as follows:

- The first line specifies that the file Targetfile.exe depends on <depfile1>, <depfile2>, <depfile3>, and <depfile4> (the dependency files).
- If any of the dependency files are newer than the target file, or if the target file does not exist, **make** builds the target file by executing the commands that follow.
- If no dependency files are listed, **make** always builds the target file.
- If one or more of the dependency files does not exist, **make** tries to create the missing dependency file(s) before executing the commands to create the target file. However, if **make** cannot find the rules that define how to create the necessary file(s), it stops and reports an error.

Examples of Dependency Rules

Example 1

```
main.exe:  main.c main.h
           gcc -o main.exe main.c
```

In Example 1, main.exe is dependent on main.c and main.h. If either of these files is newer than main.exe, or if main.exe does not exist, the command **gcc -o main.exe main.c** will be executed to create or update the target file main.exe.

Example 2
```
main.exe:  main.c main.h
           gcc -c main.c               // create the object file main.o
           ld -o main.exe main.o       // create the executable file
```

In Example 2, two commands are executed to build main.exe

Example 3
```
clean:
           del *.o
```

In Example 3, the target is named 'clean' and depends on no other files. The action is to execute the MS-DOS command, **del *.o**, which deletes all files with .o extension.

This example shows that **make** can be used as a simple action-labeller. In this case, no target file is created. This rule simply indicates that the name 'clean' refers to the action 'delete all files with .o extension'.

Implicit Rules

If there are no commands specified for building the target file, **make** searches for implicit rules that define how to build the target file.

Implicit rules are general rules that define how to create one type of file from another, for example, how to convert an '.ASM' file into an '.EXE' file.

Implicit rules take the following form.

```
.<source extension>.<target extension>:
                                         command
                                         ...
                                         command
```

'target extension' specifies the extension of the target file, and 'source extension' specifies the extension of the file(s) which is used to create the target file. Subsequent lines must specify at least one command that will be executed to build a file that has the target filename extension.

Example

```
.c.o:
```

```
                gcc -c $@ $<
```

In this example **make** creates files with the extension '.o' from files with the extension '.c' using **gcc**. (See the section below for an explanation of the symbols “$@”and “@<”).

Command Searching

make searches for commands to execute in the following order.

1. The current directory.
2. The directory specified by PATH.

If the specified command is not an EXE or a COM file, or if the command is a BAT file, **make** calls COMMAND.COM to execute the command or batch file. Thus, MS-DOS commands such as CD and DEL can be used within Makefiles (as shown in Example 3, above).

Command Prefix

The '@' prefix can be used when specifying commands in dependency rules and implicit rules. The '@' prefix tells **make** not to display the command.

Example

```
@ld -o test test.o
```

make normally displays the command it is executing if the '-s' option is not specified. To prevent commands from being displayed, an '@' should be added to the beginning of the command to be executed.

Macros

A macro is a symbol that represents a string of characters. When a macro is used it is replaced with the string that it represents. Thus, macros are a form of shorthand labelling.

Macros take the following form.

```
Macro_name = Macro_text
```

The name of the macro (‘Macro_name’) and the text that defines its content ('Macro_text') are separated by an equal sign ('='). Upper case letters are distinct from lower case letters.

If a macro definition refers to another macro, the macro will be expanded when it is referenced. Macros used in rules are expanded immediately.

Macros can be redefined at any time.

When a macro appears, its contents are replaced by the character string defined by 'Macro_text'. A macro that has been defined can be referred to in the following manner.

```
$(macro_name)
```

Example 1

```
C_FLAGS = -O2 -DDEBUG
...
.c.o:            // implicit rule for constructing .o files from .c files
gcc $(C_FLAGS) -c $@ $<
```

In this example, '$(C_FLAGS)' in the **gcc** command line will be replaced by '-O2 -DDEBUG'.

Example 2

```
PROJECT = main                                  // name of executable
OBJECT_FILES = main.o pad.o tim.o tmd.o         // list of object files
LINKER = -Xlinker -Ttext -Xlinker 80100000      // linker option
$(PROJECT): $(OBJECT_FILES)
gcc $(LINKER) $(OBJECT_FILES) -o $(PROJECT)
```

Predefined Macros

Predefined macros all begin with the dollar symbol ('$'), and can be used instead of filenames in dependency or implicit rule command lines.

The table below shows examples of predefined macros.

Macro	Meaning
$@	Name of target file
$?	Updated dependency file
$<	First dependency file

Table: Examples of Predefined Macros

Example1

```
    LINKER = -Xlinker -Ttext -Xlinker 80100000     // linker option


    main: main.o pad.o tim.o tmd.o
                gcc  $(LINKER) -o $@                // equivalent to: gcc
$(LINKER)                                                          -o main
```

Directives

The following directives can be used within a Makefile.

Directive	Meaning
define <variable>	Assert 'variable'.
endef	Undefine the 'variable' asserted using 'define'.
ifdef <variable>	Test if 'variable' has been asserted. If 'true' execute the next line. If 'false' jump to 'else' or 'endif'.
ifndef <variable>	Test if 'variable' has not been asserted. If 'true' execute the next line. If 'false' jump to 'else' or 'endif'.
ifeq (A,B)	Test if A and B are equal. If equal execute the next line. If not equal jump to 'else' or 'endif'.
ifeq "A" "B"	(Same as above)
ifeq 'A' 'B'	(Same as above)
ifneq (A,B)	Test if A and B are not equal. If not equal execute the next line. If equal jump to 'else' or 'endif'.
ifneq "A" "B"	(Same as above)
ifneq 'A' 'B'	(Same as above)
else	If the preceding if ... condition is not met, execute the next line.
Endif	End the 'if....' statement condition clause(s)
include <file>	Include the file 'file'

Table: Makefile Directives

Comments

make treats a line with a hash symbol at the beginning (#) as a comment.

Example

```
# whole line comment: main.exe only depends on main.c
main.exe: main.c
```

Line Division

If a command is too long to fit on a single line, a backslash ('\') can be added at the end of the line to allow
the command to continue on to the next line.

Example 1

```
main.exe:  main.c header1.h header2.h \
           header3.h header4.h
```

Example 2

```
OBJECT_FILES =    file1.c file2.c file3.c\
                  file4.c file5.c file6.c file7.c
```

17

The Console Tool

SIOCONS is a DOS console tool that allows you to download programs and data to the Net Yaroze PlayStation where they can be executed.

An Overview of SIOCONS

SIOCONS is a user interface front-end program that controls the Net Yaroze PlayStation from a DOS environment. Its operating requirements are listed below.

Operating machine type:	IBM-PC compatible computer
Operating environment:	Net Yaroze PlayStation
Operating OS:	DOS Version 5 or DOS Version 6,
	Windows 3.1 (DOS window), Windows 95 (DOS box)
Driver required:	ANSI.SYS

Usage

```
    siocons [-pport address,IRQ] [-Bbaud rate] [auto file]
```

Perform the following steps before executing SIOCONS on your PC.

1. First, check to be sure that ANSI.SYS is included in your PC's CONFIG.SYS file. If it is not, edit your CONFIG.SYS file, then reboot your PC.
2. Check that your PC and PlayStation are connected using the Communications cable.
3. Insert the Net Yaroze boot disk in the Net Yaroze PlayStation and turn the power on. If the Access card is not in the PlayStation at this point, insert it into Memory card slot 1.
4. Start SIOCONS on your PC by typing:

   `C:>`**`siocons -<option1>`** (where '<option1>' is an optional parameter described below)

Options

-pport- address,IRQ	specifies the communication port address and IRQ setting

-Bbaud rate	specifies the communication rate
auto file	specifies a batch file to be automatically executed

Example

```
    siocons -p0x3f8,4 -B115200
    siocons batch1
```

Operation

With SIOCONS, normal keyboard input is sent to the PlayStation, and characters received from the PlayStation are displayed on the PC monitor. (This means that the C function <printf>, called by a program running on the PlayStation, will output to the SIOCONS console on the PC). However, function keys and cursor movement keys are processed locally and are not sent to the PlayStation.

Monitor Commands

You can use certain PlayStation monitor commands with SIOCONS. These are listed below.

Command	Function
DW/DH/DB	Display memory contents in hexadecimal
SW/SH/SB	Alter memory contents in hexadecimal
FW/FH/FB	Write memory continuously
DR	Display register contents in hexadecimal
SR	Alter register contents in hexadecimal
GO	Execute program
DIS	Disassemble memory contents
AC/DC/SC	Save user defined commands
AD/DD	Save device drivers
HELP/?	Display help messages
RDB	Transfer to GNU debugging monitor mode
DIR	Display a list of files in directories
CD	Change or display current directory

READ	Copy data (file → memory)
WRITE	Copy data (memory → file)
REN	Rename file
DEL	Delete file
FORMAT	Format file
LOAD	Read in PLAYSTATION EXE (specify filename)
EXEC	Execute PLAYSTATION EXE (specify filename)
WAR	Copy data (memory → waveform memory)
WAW	Copy data (waveform memory → memory)
VAR	Copy data (memory → frame buffer)
VAW	Copy data (frame buffer → memory)
PLAY	Play DA (specify track)
BAUD	Set communication speed
CB	Display color bar
CLS	Clear console screen

Table: SIOCONS PC Monitor Commands

Local Commands

SIOCONS also provides a number of local commands for performing various functions. These commands can be executed by pressing a function key after the SIOCONS prompt.

The following is a list of local commands supported by SIOCONS.

[F1]	Display help messages.
[F2]	Prompt for a DOS command and execute the command.
[F3]	Prompt for the name of a batch file and execute the file.
[F4]	Prompt for the name of an object file and download the file.
[F5]	Prompt for the name of a log file and begin logging operations.
[F8]	Toggle the local line editor function ON/OFF.
[F9]→[F4]	Prompt for the name of a binary file and download the file.

[F9]→[F5]	Stop logging operations.
[F10]→[F2]	Exit the SIOCONS program.
[F10]→[F4]	Prompt for a filename and upload a memory image.

Editing Functions When Executing Local Commands

Some local commands prompt for filenames. When executing these commands, the following keys can be used to edit the keyboard entry.

Left Arrow, Ctrl-S	Move cursor to the left.
Right Arrow, Ctrl-D	Move cursor to the right.
Up Arrow, Ctrl-E	Move back through history buffer.
Down Arrow, Ctrl-X	Move forward through history buffer.
Ctrl-G	Delete character at cursor position.
Ctrl-Y	Delete line. If the history buffer is being viewed, the line is deleted from the history buffer.
Ctrl-K	Delete from cursor position to end of line.
TAB	Move back through the history buffer until an initial substring match is found with the current line.
Ctrl-J	Move forward through the history buffer until an initial substring match is found with the current line.
ESC	Push current line into the history buffer without executing it.

Downloading and Executing Files

Downloading Programs

A program can be downloaded to the PlayStation by pressing the [F4] key at the SIOCONS prompt. A 'Load[x]' prompt will be displayed at which time the filename to be downloaded should be entered. The specified file must be a PS-X EXE executable file, although the filename need not end in EXE.

Example

```
Load[1]: main
```

Note that you need to download the data used by a program before the program can run.

Downloading Data

Data can be downloaded to the PlayStation by pressing the [F9] key then the [F4] key at the SIOCONS prompt. A ‘Dload[x]’ prompt will be displayed, and the filename and the hexadecimal address where the file will be loaded into main memory should then be entered. More than one file and address can be specified.

Example

```
Dload[2]: wheel256.tim 80190000
```

This example loads the binary TIM file (‘whee1256.tim') into main memory starting at address 0x80190000.

Example

```
Dload[3]: giulieta.tmd 801a0000
```

This example loads the binary TMD file ('giulieta.tmd') into main memory starting at address 0x801a0000.

Program Execution

To execute a program that has been downloaded to the PlayStation, enter the monitor command 'go' from the SIOCONS command prompt. (The execution start address can be specified as an argument, but this is usually not necessary).

Terminating SIOCONS

To terminate SIOCONS, press the [F10] key, followed by the [F2] key, at the SIOCONS command prompt. Alternatively, force-quit the program by pressing [Esc].

Auto-execution

SIOCONS has a simple auto-execution facility. A text file containing commands entered from the keyboard can be read by SIOCONS and immediately executed, just like a batch file.

To use the auto-execution facility, press the [F3] key at the SIOCONS command prompt. An ‘Auto[x]’ prompt will be displayed, and the filename of the auto-execution file should be entered. The contents of the file will be loaded and the commands will be executed.

[Note that the auto-execution facility executes the first line of the batch file unconditionally, so ensure that the SIOCONS command prompt is displayed when you use this facility. In other words, make sure that the Net Yaroze PlayStation is ready to download before attempting to run a batch file.]

Example

```
Auto[3]: batch1
```

Local Commands from Batch Files

To execute a SIOCONS local command from within a batch file, first type the word 'local', then the local command and its arguments, as follows:

```
local local-command [argument....]
```

The following local commands can be used from batch files.

Commands	Action
help	Display help messages.
dos <argument>	Execute the DOS command supplied as the argument.
auto <batchfile>	Execute the batch file supplied as the argument. Nesting of batch files up to 16 levels is allowed.
dload <filename address>	Download a binary file. Specify a filename and address pair as the argument.
load <filename>	Download an executable file. Specify a filename as the argument.
log [filename]	If the name of a log file is supplied as the argument, logging will begin. If there is no argument, logging will be terminated.
dsave <filename address size>	Upload a memory image file. Specify a combination of filename, address and size as the argument.
beep	Sound the buzzer
pause	Wait for keyboard input. Continue when a suitable key is pressed.
echo <string>	Display the character string supplied as the argument.
sleep <second>	Pause for the number of seconds specified in the argument.
wait-prompt	Wait for the monitor prompt to be displayed.
auto-again	Re-execute a batch file from the start.
quit	Cause the SIOCONS program to terminate.

Table: SIOCONS Local Commands

Examples of Batch Files

In the example below, a TIM file and a program are downloaded after the PlayStation is reset. The batch file waits for a key to be pressed then executes the program.

Example

```
local dload wheel256.tim 80190000
local load program 80080000              // download program
local beep
sr sp 801ffff0
local echo Type Any Key to Execute Program
                                  // print text to SIOCONS console
local pause                       // wait for key press on the PC
go 80080100
```

Batch files can also be executed by specifying their names as an argument to SIOCONS from the MS-DOS prompt as shown below.

```
siocons  <autoexecution-file-name>
```