ChorusOS 4.0.1 UltraSPARC-IIi Target Family Guide

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part Number 806-5259–10
July 2000

Contents

1. ChorusOS 4.0.1 UltraSPARC-IIi Target Family Guide 5

Preface 5

How This Guide is Organized 5

Related Books 6

Typographical Conventions 6

Shell Prompts 7

Ordering Sun Documents 7

Accessing Sun Documentation Online 7

Obtaining Technical Support 7

Development Environment 8

Solaris™ (SPARC™ Platform Edition) Reference Host Environment 8

Cross Compiler 8

Graphical Debugger 9

ChorusOS Supported Features 9

Libraries 12

Utilities 13

Target Utilities 13

Host Utilities 15

Reference Hardware 16

Reference Processors and BSPs 16

cp1500 Reference BSP 16

Reference Target Platforms 18

Validated Reference Targets 19

How to Build and Boot a System Image on the Target 19

▼ Building a ChorusOS System Image 20

▼ Configuring the Boot Server 22

▼ Booting the Target System 24

A. ChorusOS for UltraSPARC-IIi Additional Man Pages 25

java(1CC) 26

startjvm(1CC) 29

monitor(2K) 32

JVM(5FEA) 34

MONITOR(5FEA) 35

intel28F016SA(9DRV) 36

m48txx(9DRV) 39

B. ChorusOS 4.0.1 for UltraSPARC-IIi Product Packages 43

Binary Product — for Solaris Host 43

Flite Add-on for Solaris Host 45

Source Add-on for Solaris Host 45

Documentation for Solaris Host 45

CHAPTER 1

ChorusOS 4.0.1 UltraSPARC-IIi Target Family Guide

This guide describes how to run the ChorusOS™ 4.0.1 product for the UltraSPARC™ -IIi processor family.

Preface

How This Guide is Organized

ChorusOS UltraSPARC-IIi specific information is provided in the following major sections:

- "Development Environment" on page 8, includes supported hosts, host operating systems and development systems.
- "ChorusOS Supported Features" on page 9, includes kernel components and POSIX components.
- "Libraries" on page 12.
- "Utilities" on page 13, includes host and target utilities.
- "Reference Hardware" on page 16, includes supported reference platforms, supported devices, and validated reference platforms.
- "How to Build and Boot a System Image on the Target" on page 19.
- Appendix A, presents additional man pages for the extended BSP and the MONITOR and JVM features (these man pages are not available for on-line search using the `man` command).

- Appendix B, details the list of Solaris packages in the product components, and the associated part numbers.

Related Books

See the *ChorusOS 4.0 Installation Guide for Solaris Hosts* for a description of the installation process of the ChorusOS product on a host workstation running the Solaris™ operating environment. This document also describes how to set up a boot server running the Solaris operating environment.

See the *ChorusOS 4.0 Introduction* for a complete description of the ChorusOS features.

Typographical Conventions

The following table describes the typographic changes used in this book.

TABLE 1–1 Typographical Conventions

Typeface or Symbol	**Meaning**	**Example**
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls –a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type **`rm`** *filename*.
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You must be *root* to do this.

Shell Prompts

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE 1–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	#

Ordering Sun Documents

Fatbrain.com, an Internet professional bookstore, stocks selected product documentation from Sun Microsystems, Inc.

For a list of documents and how to order them, visit the Sun Documentation Center on Fatbrain.com at `http://www1.fatbrain.com/documentation/sun`.

Accessing Sun Documentation Online

The docs.sun.com[SM] Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Obtaining Technical Support

Sun Support Access offerings are available exclusively to members of the Sun Developer Connection Program. To get free membership in the Sun Developer Connection Program, go to `http://www.sun.com/developers`. For more information or to purchase Sun Support Access offerings, visit: `http://www.sun.com/developers/support` or contact the Sun Developer Connection Program office near you.

Development Environment

The ChorusOS product provides a host-target development environment. Applications are developed on a workstation (the host), and then downloaded and executed on a specific board (the target).

A cross development system is needed to build the applications that execute on the target board (see Section “Utilities” on page 13).

Solaris™ (SPARC™ Platform Edition) Reference Host Environment

Prerequisites for the Solaris host reference configuration are the following:

- Sun SPARCstation™.
- Solaris 2.6, or Solaris 7 (32-bit).
- Sun WorkShop™ 5.0 native compiler.

Note - In order for the CC compiler to work properly, all patches related to the CC compiler must have been installed on the Solaris system.

- JDK™ 1.1.8, for the installation tool.
- JDK 1.2, for the graphical configuration tool and for Java™ applications.

Cross Compiler

This development environment component is bundled with the ChorusOS for UltraSPARC-IIi product:

- Chorus Cross Development System 5.0, target UltraSPARC-IIi ELF.

The Chorus Cross Development System is based on the Experimental GNU Compiler System egcs 1.1.2 and binutils 2.9.1 and additional patches.

Graphical Debugger

This development environment component is bundled with the ChorusOS for UltraSPARC-IIi product:

- XRAY Debugger from Mentor Graphics, ELF format, version 4.4crd.

ChorusOS Supported Features

The following table shows the ChorusOS kernel and operating system optional features that are available for the UltraSPARC-IIi processor family. The availability status of a feature can be one of:

Y The feature is supported, and is configurable using the `configurator`(1CC) command, or with the *ews* GUI configuration tool.

Please refer to the note at the end of the table for information about specific conditions, or restrictions, for a given supported feature.

Some of the features (such as MSDOSFS, FLASH, FS_MAPPER, for example) require specific low-level drivers. These features operate only on platforms which provide these drivers.

N The feature is not supported.

Feature Description	Feature Name	Availability
Actor management		
Dynamic actor loading management	ACTOR_EXTENDED_MNGT	Y
User-mode extension support	USER_MODE	Y
Dynamic libraries	DYNAMIC_LIB	Y
Compressed file management	GZ_FILE	Y
Scheduling		
POSIX round-robin scheduling class	ROUND_ROBIN	Y
Memory management		

Feature Description	Feature Name	Availability
Virtual (user and supervisor) address space	VIRTUAL_ADDRESS_SPACE	Y
On-demand paging	ON_DEMAND_PAGING	N
Hot restart and persistent memory		
Hot restart	HOT_RESTART	Y
Inter-thread communication		
Semaphores	SEM	Y
Event flag sets	EVENT	Y
Mutual exclusion lock supporting thread priority inversion avoidance	RTMUTEX	Y
Monitors	MONITOR	Y
Time management		
Periodic timers	TIMER	Y
Thread and actor virtual timer	VTIMER	Y
Date and time of day	DATE	Y
Real-time clock	RTC	Y
Inter-process communication		
Location-transparent inter-process communication	IPC	Y
Remote (inter-site) IPC support	IPC_REMOTE	Y
Remote IPC communications medium	IPC_REMOTE_COMM	Y
Mailbox-based communications mechanism	MIPC	Y
POSIX 1003.1-compliant message queues	POSIX_MQ	Y
POSIX 1003.1-compliant shared memory objects	POSIX_SHM	Y
LAP		
Local name server for LAP binding	LAPBIND	Y
LAP validity-check option	LAPSAFE	Y
Tools support		

Feature Description	Feature Name	Availability
Message logging	LOG	Y
Profiling and benchmark support	PERF	Y
System Monitoring	MON	Y
System debugging	DEBUG_SYSTEM	Y
C_INIT		
Basic command interpreter on target	LOCAL_CONSOLE	Y
Remote shell	RSH	Y
File system options		
Named pipes	FIFOFS	Y
MS-DOS file system	MSDOSFS	Y
NFS client	NFS_CLIENT	Y
NFS server	NFS_SERVER	Y
UFS file system	UFS	Y
I/O management		
Network packet filter	BPF	Y
Swap support	FS_MAPPER	N
Driver for IDE disk	IDE_DISK	N
`/dev/mem`, `/dev/kmem`, `/dev/null`, `/dev/zero`	DEV_MEM	Y
Support for RAM disk	RAM_DISK	Y
Support for FLASH media [1]	FLASH	Y
Virtual TTY	VTTY	Y
Driver for SCSI disk	SCSI_DISK	Y
Support for IPC	IOM_IPC	Y
Support for OSI	IOM_OSI	Y
Networking		
Serial link IP	SLIP	Y
POSIX 1003.1g-compliant sockets	POSIX_SOCKETS	Y
Point-to-point protocols	PPP	Y
Local sockets and pipes	AF_LOCAL	Y
Administration		

Feature Description	Feature Name	Availability
ChorusOS statistics	ADMIN_CHORUSSTAT	Y
`ifconfig` administration command	ADMIN_IFCONFIG	Y
`mount` administration command	ADMIN_MOUNT	Y
`rarp` administration command	ADMIN_RARP	Y
`route` administration command	ADMIN_ROUTE	Y
`shutdown` administration command	ADMIN_SHUTDOWN	Y
`netstat` administration command	ADMIN_NETSTAT	Y
JVM		
Java Virtual Machine	JVM	Y

1. Logical-to-Physical block mapping for flash file system support.

Libraries

The ChorusOS operating system provides the elementary libraries indicated in the following list:

ChorusOS embedded library [1]	`libebd.a`
ChorusOS extended library [1]	`libcx.a`
C++ library	`libC.a`
X11 related client libraries (not thread safe)	`libX11.a`, `libXaw.a`, `libXext.a`, `libXmu.a`, `libXt.a`
Specific BSD APIs (not thread safe)	`libbsd.a`
The SunRPC library	`librpc.a`
The mathematical library	`libm.a`

The ‘‘embedded’’ C library [2]	`stdc.a`
The microkernel ‘‘visu’’ library [3]	`visu.a`

1. The `libebd.a`, `libcx.a`, `libm.a` and `libC.a` libraries have been made thread-safe in order to support multithreaded actors.
2. Included in `libebd.a`.
3. This library is provided for the sake of backwards compatibility only. It is not documented. Its use is strongly discouraged.

Utilities

Target Utilities

The following utilities may be run on the target ChorusOS operating system:

chorusStat(1CC)

cp(1CC)

cs(1CC)

date(1CC)

dd(1CC)

df(1CC)

domainname(1CC)

ftp(1CC)

hostname(1CC)

java(1CC)

ls(1CC)

mkdir(1CC)

mkfifo(1CC)

mv(1CC)

netstat(1CC)

nfsstat(1CC)

pax(1CC)

PROF(1CC)

profctl(1CC)

rdbc(1CC)

rm(1CC)

rmdir(1CC)

startjvm(1CC)

touch(1CC)

uname(1CC)

ypcat(1CC)

ypmatch(1CC)

ypwhich(1CC)

arp(1M)

chat(1M)

chorusNS(1M)

chorusNSinet(1M)

chorusNSsite(1M)

dhclient(1M)

disklabel(1M)

flashdefrag(1M)

format(1M)

fsck(1M)

fsck_dos(1M)

ftpd(1M)

inetNS(1M)

inetNSdns(1M)

inetNShost(1M)

inetNSien116(1M)

inetNSnis(1M)

mkfd(1M)

mkfs(1M)

mount(1M)

mount_msdos(1M)

mount_nfs(1M)

mountd(1M)

newfs(1M)

newfs_dos(1M)

nfsd(1M)

portmap(1M)

route(1M)

shutdown(1M)

slattach(1M)

syncd(1M)

sysctl(1M)

telnetd(1M)

umount(1M)

ypbind(1M)

Host Utilities

The following utilities may be run on the host machine:

chadmin(1CC)

chconsole(1CC)

chlog(1CC)

chls(1CC)

ChorusOSMkMf(1CC)

chserver(1CC)

configurator(1CC)

configure(1CC)

ews(1CC)

mkmerge(1CC)

rdbs(1CC)

profrpg(1CC)

Reference Hardware

ChorusOS targets are described in this section from three different points of view:

Reference Processors and BSPs:

This subsection describes the processors on which the ChorusOS product can run, as well as the details of the BSPs included in the delivery.

Reference Target Platforms:

This section describes all the target platforms which can be used as references in the context of Sun support contracts.

Validated Reference Targets:

This section describes the precise platforms used to run the Sun QA tests; this may be useful, in case of bugs, as a hint or guide to help in identifying issues which are closely hardware related.

Reference Processors and BSPs

The ChorusOS system for UltraSPARC-IIi supports the following processor:

- UltraSPARC-IIi.

The ChorusOS system for UltraSPARC-IIi supports the following reference BSP:

- cp1500 Reference BSP.

cp1500 Reference BSP

Systems

The cp1500 reference BSP supports the following board and system:

- CP1500 — Sun Microsystems.

- Netra t1 Model 105 — Sun Microsystems.

Devices

The cp1500 reference BSP supports the following on board devices:

Device Id	ChorusOS Driver
/cpu	sun:usparc-tick-timer
/ric	sun:usparc-ric-ric
/sabre	sun:usparc-sabre-pci
/sabre/simba	sun:(pci,ric)-simba-pci
/sabre/simba/cheerio-ethernet	sun:pci-cheerio-ether
/sabre/simba/ebus (ebus bridge)	sun:pci-ebus-(bus,isa)
/sabre/simba/ebus/ns16550-1 (UART)	sun:bus-ns16550-uart
/sabre/simba/ebus/ns16550-2 (UART)	sun:bus-ns16550-uart
/sabre/simba-2	sun:(pci,ric)-simba-pci
/sabre/simba-2/dec21150 (cPCI bridge)	sun:pci-dec2115x-(bus,pci)
/sabre/simba/ebus/28f008 (system flash)	not supported
/sabre/simba/ebus/28f016 (user flash)	intel28F016SA
/sabre/simba/ebus/led (ready LED)	not supported
/sabre/simba/ebus/7seg (7-seg LED)	not supported
/sabre/simba/ebus/wdtimer (watchdog)	not supported
/sabre/simba/ebus/tempsensor	not supported
/sabre/simba/ebus/kbd (keyboard)	not supported
/sabre/simba/ebus/mouse	not supported
/sabre/simba/ebus/lpt (parallel)	not supported
/sabre/simba/ebus/fdd (floppy)	not supported

Device Id	ChorusOS Driver
/sabre/simba/ebus/ide	not supported
/sabre/simba/ebus/tod (time of day)	sun:bus-m48txx-(nvram,rtc)
/sabre/simba/ebus/nvram	sun:bus-m48txx-(nvram,rtc)
/sabre/simba/sym53c875 (SCSI)	sun:pci-ncr53c8xx-scsi

Reference Target Platforms

Reference target platforms are configurations to be used by customers covered by a Sun support contract.

SPARCengine Ultra CP1500 (Sun Microsystems)

Type: CompactPCI Board

Processors: UltraSPARC-IIi (270-333 Mhz)

Main memory: 64-512 MB

L2 cache: 256-1024 MB

Bus bridges: Processor to PCI, PCI to PCI, PCI to cPCI, PCI to ISA

Devices: Asynchronous serial ports (38.4 Kbaud), 10/100BaseT Ethernet, SCSI-3, Real-time clock, Timers, Flash memory (Intel 28F016SA)

Firmware: OpenBoot 3.10.x

Netra t1 Model 105 (Sun Microsystems)

Type: Rack-mounted System

Processors:	UltraSPARC-IIi (360-440 Mhz)
Main memory:	64-512 MB
L2 cache:	1-2 MB
Bus bridges:	Processor to PCI, PCI to PCI, PCI to ISA
Devices:	Asynchronous serial ports (38.4 Kbaud), 10/100BaseT Ethernet, SCSI-3, Real-time clock, Timers, Flash memory (Intel 28F016SA)
Firmware:	OpenBoot 3.10.x

Validated Reference Targets

This section describes the precise platform used to run the Sun QA tests:

- SPARCengine Ultra CP1500: 5093-01 REV-51 / 333 Mhz / 128 MB.
- Netra 1 model 105 One pack 360Mhz, 1 MB eCache, 64 MB.

How to Build and Boot a System Image on the Target

Note - UltraSPARC-IIi target systems do not boot archives directly, but instead require the inetboot application to boot.

As a result, the boot procedure that follows differs from the procedure described in the *ChorusOS 4.0 Installation Guide for Solaris Hosts*, and requires that you use a boot server that reads local files instead of the NIS™ database.

▼ Building a ChorusOS System Image

The following procedure assumes that the ChorusOS product has already been correctly installed on the host workstation. See the *ChorusOS 4.0 Installation Guide for Solaris Hosts*.

1. Create and change to a build directory where you will build system images:

```
$ mkdir build_dir
$ cd build_dir
```

2. Set an environment variable to use with the `configure`(1CC) command as a shortcut to the base directory.

For example:

Set the environment variable…	To the family-specific product directory. The default value is…
DIR	/opt/SUNWconn/SEW/4.0.1/chorus-usparc

3. Make sure your PATH has been set correctly to include the directory *install_dir*`/4.0.1/chorus-usparc/tools/host/bin`, where the default *install_dir* is `/opt/SUNWconn/SEW`.

Also make sure that your PATH includes `/usr/openwin/bin`, which contains the `imake` utility.

4. Configure the build directory, using the `configure`(1CC) command:

If you are building from a binary distribution:

```
$ configure -b $DIR/kernel \
$DIR/os \
$DIR/tools \
-s $DIR/src/nucleus/bsp/drv \
$DIR/src/nucleus/bsp/usparc \
$DIR/src/nucleus/bsp/usparc/cp1500 \
```

(continued)

```
$DIR/src/iom
```

Note - The above command configures the build directory to include components installed during a “Default Install”. It does not include optional components, such as the X library or code examples, that you may choose to install separately on Solaris host workstations. For example, in order to include everything in your build environment:

```
$ configure -b $DIR/kernel \
$DIR/os \
$DIR/opt/X11 \
$DIR/tools \
-s $DIR/src/nucleus/bsp/drv \
$DIR/src/nucleus/bsp/usparc \
$DIR/src/nucleus/bsp/usparc/cp1500 \
$DIR/src/iom \
$DIR/src/opt/examples
```

If you are building from the source distribution, see the *ChorusOS 4.0 Production Guide*.

As a result of configuration, *build_dir* now contains a `Makefile`, which is used to generate the build environment, and a `Paths` file, which specifies paths to files required by, and created in, the build environment.

5. **Generate the build environment:**

```
$ make
```

6. **Build a system image:**

```
$ make chorus
```

The resulting system image file is located in the build directory, *build_dir* and is called `chorus.obp`.

Note - You can also make a smaller system image that includes only the operating system kernel:

```
$ make kernonly
```

▼ Configuring the Boot Server

Note - The target system and boot server must be on the same subnet.

See the *ChorusOS 4.0 Installation Guide for Solaris Hosts* for instructions on how to enable TFTP and RARP services on the boot server.

1. **Become root on the boot server:**

```
$ su
Password: root_password
#
```

2. **Copy the** `inetboot.sun4u` **file,** *install_dir*`/4.0.1/chorus-usparc/opt/unsupported/inetboot.sun4u` **to the** `/tftpboot` **directory.**

   By default, *install_dir* is `/opt/SUNWconn/SEW`.

3. **Create a soft link from the** `inetboot.sun4u` **file to a file called** `/tftpboot/`*target_IP_address_in_hex*. **For example, the file for the target system with IP address** `129.157.197.88` **is called** `819DC558`**, and is constructed as follows:**

   - `129` in decimal translates to `81` in hexadecimal.
   - `157` in decimal translates to `9D` in hexadecimal.
   - `197` in decimal translates to `C5` in hexadecimal.
   - `88` in decimal translates to `58` in hexadecimal.

   To create the soft link for the above example:

```
# ln -s /tftpboot/inetboot.sun4u 819DC558
```

4. **Create a** `/tftpboot/export/root/`*target*`/platform/sun4u` **directory on the boot server.**

5. **Copy the system image,** `chorus.obp`**, to the directory you just created.**

6. **Edit** `/etc/bootparams` **to include the following, creating the file if it does not yet exist:**

target `root=`*boot_server*`:/tftpboot/export/root/`*target*

   Where *target* is the target system hostname and *boot_server* is the boot server hostname.

7. **Edit** `/etc/hosts` **to include the following:**

target_IP_address *target*

   Where *target_IP_address* is the target IP address, such as `129.157.197.88`, and *target* is the target system hostname.

8. **Edit** `/etc/ethers` **to include the following:**

target_Ethernet_address *target*

   Where *target_Ethernet_address* is the target IP address such as `8:0:20:a7:d6:f3` and *target* is the target system hostname.

9. **Add the following line to** `/etc/dfs/dfstab`:

```
share -F nfs -o rw -d "ChorusOS boot"  /export/home
```

10. **Edit** `/etc/nsswitch.conf` **so that the** `hosts`**,** `ethers`**, and** `bootparams` **entries read files rather than NIS databases. For example:**

```
#
# /etc/nsswitch.nis:
#
# An example file that could be copied over to /etc/nsswitch.conf; it
# uses NIS (YP) in conjunction with files.
#
# "hosts:" and "services:" in this file are used only if the
# /etc/netconfig file has a "-" for nametoaddr_libs of "inet" transports.

# the following two lines obviate the "+" entry in /etc/passwd and /etc/group.
passwd:     files nis
group:      files nis

# consult /etc "files" only if nis is down.
```

```
hosts:      files [NOTFOUND=return] nis
networks:   nis [NOTFOUND=return] files
protocols:  nis [NOTFOUND=return] files
rpc:        nis [NOTFOUND=return] files
ethers:     files [NOTFOUND=return] nis
netmasks:   nis [NOTFOUND=return] files
bootparams: files [NOTFOUND=return] nis
publickey:  nis [NOTFOUND=return] files

netgroup:   nis

automount:  files nis
aliases:    files nis

# for efficient getservbyname() avoid nis
services:   files nis
sendmailvars:   files
```

11. Reboot the boot server.

▼ Booting the Target System

- **Boot the target by entering the following in the target system console:**

```
# ok boot net /platform/sun4u/chorus.obp
```

APPENDIX **A**

ChorusOS for UltraSPARC-IIi Additional Man Pages

The following additional man pages are not available for on-line use using the `man` command. They will be integrated with the package of man pages in a later major product release.

section 1CC: Host and Target Utilities

java(1CC)
startjvm(1CC)

section 2K: Kernel System Calls

monitor(2K), *monitorInit(2K)*, *monitorGet(2K)*, *monitorNotify(2K)*, *monitorNotifyAll(2K)*, *monitorRel(2K)*, *monitorWait(2K)*

section 5FEA: ChorusOS Features and APIs

JVM(5FEA)
MONITOR(5FEA)

section 9DRV: Driver Implementations

intel28F016SA(9DRV)
m48txx(9DRV)

NAME

java – Java interpreter

SYNOPSIS

rsh *target* **arun $JVM_ROOT/bin/java** [–quit] [–rehash [*ENVAR=VALUE*]] [–viewclasses] [–viewthreads] *classname* [*args*]

FEATURES

JVM

DESCRIPTION

`java` is a target utility.

The `java` command executes Java bytecodes created by the Java compiler, javac, on the host system.

The *classname* argument is the name of the class to be executed and must be fully qualified by including the package in the name, for example:

```
example% rsh target arun JVM_ROOT/bin/java java.lang.String
```

Note that any arguments that appear after *classname* on the command line are passed to the **main()** method of the class.

The bytecodes for the class are put in a file called *classname*`.class` by compiling the corresponding source file with javac. All Java bytecode files end with the filename extension `.class`, which the compiler automatically adds when the class is compiled. The *classname* argument must contain a **main()** method defined as follows:

```
class Aclass {
     public static void main(String argv[]){
     . . .
     }
}
```

The `java` command returns control to the command interpreter as soon as it has succeeded in loading the class. It then executes the **main()** method and exits unless **main()** creates one or more threads. In this case, `java` does not exit until the last thread exits. Note that exiting a class *never* causes the Java Virtual Machine to exit in the context of ChorusOS.

When defining classes, specify their locations using the APP_CLASSPATH environment variable, which consists of a colon-separated list of directories that specifies the path.

OPTIONS

The following options are supported:

`–quit` — Kill all running Java threads and terminate the jvmd actor.

`–rehash` — Reload environment variables.

If the environment variables to reload are provided as a set of whitespace-separated

variable-value argument pairs to the –`rehash` option, the Java Virtual Machine will reload only those environment variables that are specified. Otherwise the –`rehash` option forces the Java Virtual Machine to reload all relevant environment variables. (See *ENVIRONMENT VARIABLES*.)

–`viewclasses` List all Java currently loaded classes in the jvmd actor.

–`viewthreads` List all Java threads currently running in the jvmd actor.

ENVIRONMENT VARIABLES

The following environment variables are supported:

JVM_ROOT Base directory where the Java Virtual Machine is installed.

Default value: `/opt/jvm` (as seen from the target system).

JVM_CLASSPATH Search path for non-verified bootstrap classes and resources.

Default value: `${`*JVM_ROOT*`}/classes`.

JVM_LIBPATH Search path for bootstrap native libraries.

Default value: `${`*JVM_ROOT*`}/lib`.

JVM_DEBUG Enables or diables tracing. Values assigned to this environment variable may be: `none` (no tracing), `all` (full tracing), `loading` (provide traces from internal primordial classloader), `verifying` (provide traces from the verifier), `loading, verifying` or `verifying, loading`.

Default value: `none`

JVM_GC Enables or disables garbage collection according to the mark and sweep method. Values assigned to this environment variable may be either `enable` or `disable`.

Default value: `enable`

APP_CLASSPATH Search path for application classes and resources.

Default value: None (user-defined).

APP_LIBPATH Search path for application native libraries.

Default value: None (user-defined).

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

startjvm(1CC), *JVM*(5FEA).

NOTES

The jvmd actor behaves somewhat differently from Java Virtual Machines designed for other general purpose operating systems. Refer to *startjvm*(1CC) for more information about the jvmd actor.

NAME

startjvm – run the Java Virtual Machine actor

SYNOPSIS

rsh *target* **arun /jvm/bin/startjvm**

FEATURES

JVM

DESCRIPTION

`startjvm` is a target utility.

The `startjvm` command starts the Java Virtual Machine for ChorusOS (the jvmd actor) initializing it according to the environment variables described in the *ENVIRONMENT VARIABLES* section of *java*(1CC). The jvmd actor is started only once, as all java applications executed on the target system run in the context of that actor.

Note that the `startjvm` command is not located under JVM_ROOT.

After the jvmd actor has been started using the `startjvm` command

```
Main JVM created
```

should appear on the ChorusOS console.

The corresponding command to terminate the jvmd actor is:

```
example% rsh target $JVM_ROOT/bin/java -quit
```

The jvmd actor

The jvmd actor runs on the target system.

The jvmd actor provides the Java Virtual Machine for ChorusOS. It may be terminated using the –`quit` option of the `java` command, but does not terminate simply because no Java applications are running.

The Java Virtual Machine component of ChorusOS is implemented as a single supervisor actor. That single actor holds all Java threads associated with all Java applications running on the target. In other words, all Java applications run in the supervisor space and all Java applications and associated threads belong to a single ChorusOS actor.

The following table indicates what is and is not supported.

SUPPORTED	NOT SUPPORTED
All Java™ 2 Platform, Standard Edition, v1.2.2 application programming interfaces *except AWT*, including the following packages: java.beans, java.io, java.lang, java.math, java.net, java.rmi, java.security, java.sql, java.text, java.util, sun.beans, sun.dc, sun.io, sun.jdbc, sun.misc, sun.net, sun.rmi, sun.security, sun.tools, sunw.io, sunw.util	AWT and packages that depend on AWT
The Java™ Native Interface with native code running in supervisor space	The Java Native Interface with native code running in user space only
Loading of dynamic libraries whose symbols are known to the Java Virtual Machine actor	

The following particularities, limitations and restrictions apply:

Single Supervisor Actor

The Java Virtual Machine runs as a single actor.

All Java applications running on the target depend on this single actor, jvmd. All Java applications must be started after the jvmd actor has been launched, using the *java*(1CC) command.

System.exit() Stub

System.exit() takes no action, and simply returns control to the caller.

Java Virtual Machine implementations for other host platforms allow the developer to use **System.exit()** to terminate the Java Virtual Machine currently running. As the Java Virtual Machine actor for the ChorusOS operating system runs multiple Java applications, **System.exit()** is not designed to terminate the Java Virtual Machine itself.

Java Native Interface

Developers writing applications that use the Java Native Interface must use system calls that are available for use by supervisor actors.

The Java Virtual Machine is implemented as a supervisor actor. Some symbols that can be seen in a user space view of the system are not visible in supervisor space.

Messages

All messages from the Java Virtual Machine are directed to the ChorusOS console.

Developers who need to manage messages in some other way must implement their own mechanisms for doing so, for example, by using inter-process communication or log files written to a file system.

Thread Priority

By default, threads running in the Java Virtual Machine share the same priority scale than other system threads.

Before mapping Java thread priorities to the system priorities, developers may first tune the ChorusOS system to set the maximum priority for Java threads using the `jvm.thread.maxPriority` tunable. This value forces threads running in the Java Virtual Machine to be assigned a lower overall priority than other system threads.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

java(1CC), *JVM*(5FEA)

NAME

monitor, monitorInit, monitorGet, monitorNotify, monitorNotifyAll, monitorRel, monitorWait – initialize a monitor; acquire a monitor; release a monitor; wait within a monitor for notification; notify a thread waiting within a monitor; notify all threads waiting within a monitor

#include <sync/chMonitor.h>

KnError **monitorInit**(KnMonitor *monitor);

KnError **monitorGet**(KnMonitor *monitor);

KnError **monitorNotify**(KnMonitor *monitor);

KnError **monitorNotifyAll**(KnMonitor *monitor);

KnError **monitorRel**(KnMonitor *monitor);

KnError **monitorWait**(KnMonitor *monitor, KnTimeVal *timeout);

FEATURES

MONITOR

DESCRIPTION

A monitor is a synchronization object used to protect shared procedures and data against simultaneous access. Once the monitor is acquired by a thread, the thread can suspend its ownership and wait until it is notified or a timeout occurs.

Monitors are `KnMonitor` structures allocated in memory.

monitorInit() initializes the monitor whose address is *monitor*. The monitor is initialized as unlocked.

Statically allocated monitors can be initialized using the K_KNMONITOR_INITIALIZER macro, which initializes the monitor as unlocked. This macro is used as follows:

```
KnMonitor myMonitor = K_KNMONITOR_INITIALIZER
```

monitorGet() is used by a thread to acquire a monitor. If the monitor is unlocked, it becomes locked by the thread and the caller continues its execution normally. If the monitor is already locked by the current thread, execution also continues normally. If the monitor is locked by another thread, the caller is blocked until the monitor is released.

monitorWait() is used by a thread which has acquired a monitor to relinquish its lock on it, to lie dormant until another thread notifies it using **monitorNotify()** or **monitorNotifyAll()**, or until the amount of time specified by timeout has elapsed, and finally to re-acquire its lock on the monitor.

monitorNotify() is used by a thread which has acquired the monitor specified by *monitor* to notify a thread waiting within **monitorWait()** to resume. The calling thread must then call **monitorRel()** so that the waiting thread may actually resume.

monitorNotifyAll() notifies all threads waiting within **monitorWait()** to resume.

monitorRel() is used by a thread which has acquired a monitor to release it. If threads are blocked behind the monitor, one of them is awakened.

A blocking **monitorGet()** is NONABORTABLE (see `threadAbort`(2K)). **monitorWait()** is ABORTABLE, that is, when a **threadAbort()** is addressed to a waiting thread, it behaves as if its time-out had expired.

RESTRICTIONS

A user application and a supervisor application may not share a monitor.

Conversely, two applications running in the same mode (user or supervisor) may share a monitor by mapping it in both address spaces. Such shared monitors must be dynamically allocated monitors. In supervisor mode, the same address may be used by both applications, but care must be taken to keep the monitor's region allocated because the system may crash otherwise.

RETURN VALUES

Upon successful completion, `0` is returned. Otherwise a negative error code is returned.

ERRORS

K_EFAULT	Some of the data provided are outside the address space of the current actor.
K_EINVAL	`waitLimit` is not a valid `KnTimeVal`.
K_EINVAL	The calling thread is not the current owner of the monitor on *monitorRel*, *monitorNotify*, *monitorNotifyAll*, *monitorWait*.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`mutexGet`(2K), `mutexInit`(2K), `mutexRel`(2K), `CORE`(5FEA), *JVM*(5FEA), *MONITOR*(5FEA), `MUTEX`(5FEA)

NAME

JVM – Java Virtual Machine component

FEATURE SUMMARY

The JVM feature provides support for the Java Virtual Machine component. This feature requires the MONITOR feature to be set.

This feature allows the system to provide support for Java applications using the Java Virtual Machine actor, jvmd actor.

API

The JVM feature does not itself export an API.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

java(1CC), *startjvm*(1CC), *MONITOR*(5FEA).

NAME

MONITOR – monitors

FEATURE SUMMARY

Monitors are a way of synchronizing concurrent threads. A monitor is a set of functions in which only one thread may execute at a time. It is possible for a thread running inside a monitor to suspend its execution so that another thread may enter the monitor. The initial thread waits for the second one to notify it (for example, that a resource is now available) and then to exit the monitor. By extension to object-oriented languages such as the Java™ language, monitor objects are associated with the set of functions. The functions take a monitor object as argument. Only one thread at a time uses a given monitor object. In this context, the term "monitor" often refers to the monitor object itself.

API

The MONITOR feature API is summarized in the following table:

monitorGet()	Obtains the lock on the given monitor.
monitorInit()	Initializes the given monitor.
monitorNotify()	Notifies one thread waiting in **monitorWait()**.
monitorNotifyAll()	Notifies all threads waiting in **monitorWait()**.
monitorRel()	Releases a lock on a given monitor.
monitorWait()	Causes a thread that owns the lock on the given monitor to suspend itself until it receives notification from another thread.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

monitorGet(2K), *monitorInit*(2K), *monitorNotify*(2K), *monitorNotifyAll*(2K), *monitorRel*(2K), *monitorWait*(2K), `mutexGet`(2K), `mutexInit`(2K), `mutexRel`(2K), `MUTEX`(5FEA)

NAME

intel28F016SA – intel28F016SA compatible flash driver

SYNOPSIS

drv/src/flash/intel28F016SA/intel28F016SA.h - driver structures and constants

drv/src/flash/intel28F016SA/intel28F016SA.c - driver code

drv/src/flash/intel28F016SA/intel28F016SAProp.h - driver specific properties

FEATURES

DRV

DESCRIPTION

The intel28F016SA driver implements the flash driver interface for intel28F016SA family chips, including the intel28F160 chip.

EXTENDED DESCRIPTION

The driver uses the common bus driver interface provided by a parent bus driver. Thus, the driver is generic and may be applied to any bus providing the same type of interface.

The driver does not provide the **drv_probe()** entry. The intel28F016SA driver does not therefore enumerate the bus, detect an intel28F016SA device or create an associated device node. When the intel28F016SA driver is used, associated device nodes should be created either statically by a boot program, or dynamically by a separate bus enumerator driver. This type of enumerator driver could be developed for this particular bus architecture.

The driver does not provide the **drv_bind()** routine. In other words, the driver does not support dynamic driver-to-device binding. When the intel28F016SA driver is used, the driver should be explicitly bound to the device using the `PROP_DRIVER` property. It is assumed that the device-to-driver binding is done either by a device node creator or by a separate binder driver. This type of driver-to-device binder could be developed for this particular bus architecture.

The driver provides the **drv_unload()** entry and supports driver component unloading. This allows the driver component to be unloaded if it is no longer being used (if it was dynamically loaded at run time).

The driver does not support hot-plug bus events.

The table below summarizes characteristics of the intel28F016SA flash driver.

driver name:	"sun:bus-intel28F016SA-flash"
hardware:	intel28F016SA compatible flash chip
exported interfaces:	"flash" (FLASH_CLASS)
exported interface versions:	0 (FLASH_VERSION_INITIAL)
imported parent interface:	"bus" (BUS_CLASS)
minimum parent interface version:	0 (BUS_VERSION_INITIAL)
device probing (auto-detection):	not supported

driver unloading:	supported
system (emergency) shut-down:	not supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	not supported

The table below lists device node properties used by the intel28F016SA driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the "default value" column may show a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"mem-rgn"	BUS_PROP_MEM_RGN	<bus class specific>	m	
"size"	FLASH_PROP_SIZE	FlashPropSize	m	
"region-write"	FLASH_PROP_RGN_WRITE	FlashPropAccess	o	
"region-exec"	FLASH_PROP_RGN_EXEC	FlashPropAccess	o	
"region-cache"	FLASH_PROP_RGN_CACHE	FlashPropAccess	o	

The `BUS_PROP_MEM_RGN` property specifies the base address of the media and is used by the driver as a parameter for the **bus->mem_map()** call. The property value is bus class specific.

The `FLASH_PROP_SIZE` property specifies the byte size of the media.

The upper layer should use optional properties `FLASH_PROP_RGN_WRITE`, `FLASH_PROP_RGN_EXEC` and `FLASH_PROP_RGN_CACHE` to map locked flash regions. The presence of each property gives information to the driver client about whether the locked region is writable, executable in place and cacheable.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO bus(9DDI), isa(9DDI)

NAME

m48txx – SGS m48txx real time clock device driver

SYNOPSIS

drv/src/rtc/m48txx/m48txx.c - driver code

drv/src/rtc/m48txx/m48txxProp.h - driver specific properties

FEATURES

DRV

DESCRIPTION

The m48txx driver implements the RTC and NVRAM device driver interface.

The driver uses the common bus driver interface provided by a parent bus driver. Thus, the driver is generic and may be applied to any bus providing this type of interface.

The m48txx driver does not provide the **drv_probe()** routine. In other words, the m48txx driver does not enumerate the bus, nor does it detect an m48txx device or create an associated device node. When the m48txx driver is used, associated device nodes should be created either statically by a boot program or dynamically by a separate bus enumerator driver. Such an enumerator driver could be developed for this particular bus architecture.

The driver does not provide the **drv_bind()** routine. In other words, the driver does not support dynamic driver-to-device binding. When the m48txx driver is used, the driver should be explicitly bound to the device using the PROP_DRIVER property. It is assumed that the device-to-driver binding is done either by a device node creator or by a separate binder driver. Such a driver-to-device binder could be developed for this particular bus architecture.

The driver supports all bus events specified by the common bus driver interface. As a consequence, the driver may be used with a hot-pluggable bus (for example, PCMCIA).

The table below summarizes characteristics of the m48txx driver:

driver name:	"sun:bus-m48txx-(nvram,rtc)"
hardware:	SGS m48txx chip
exported interfaces:	"rtc" (RTC_CLASS) "nvram" (NVRAM_CLASS)
exported interface versions:	0 (RTC_VERSION_INITIAL) 0 (NVRAM_VERSION_INITIAL)
imported parent interface:	"bus" (BUS_CLASS)
minimum parent interface version:	0 (BUS_VERSION_INITIAL)
device probing (auto-detection):	not supported
driver unloading:	supported

system (emergency) shut-down:	supported
normal device shut-down:	supported
hot-plug (surprise) device removal:	supported

The table below lists device node properties used by the m48txx driver. Note that the column "m/o" specifies whether a given property is mandatory or optional. For optional properties, the "default value" column may show a default value which is used by the driver when a given property is not specified.

Name	Alias	Type	m/o	Default value
"io-regs"	BUS_PROP_IO_REGS	<bus class specific>	*	
"mem-rgn"	BUS_PROP_MEM_RGN	<bus class specific>	*	
"nvram-layout"	NVRAM_PROP_LAYOUT	<NvramPropChunk[]>	o	
"year-base"	M48TXX_PROP_YEAR_BASE	<M48txxYearBase>	o	1900

* Indicates that one of these properties must be present. If both are defined, the `BUS_PROP_IO_REGS` will be used.

The `BUS_PROP_IO_REGS` property specifies the m48txx I/O registers' range. The property value is bus class specific.

The `BUS_PROP_MEM_RGN` property specifies the m48txx memory range. The property value is bus class specific.

The `NVRAM_PROP_LAYOUT` property specifies a physical layout of the NVRAM space. The property value is an array of `NvramPropChunk` structures. Each `NvramPropChunk` descriptor specifies access permissions to a given NVRAM chunk. `NVRAM_PROP_LAYOUT` covers all NVRAM space from the beginning to the end. A start offset of a given chunk is therefore calculated as the sum of the <size> fields of all previous chunks.

The `M48TXX_PROP_YEAR_BASE` property specifies the m48txx century base year.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO bus(9DDI), isa(9DDI), nvram(9DDI), rtc(9DDI),

APPENDIX **B**

ChorusOS 4.0.1 for UltraSPARC-IIi Product Packages

The tables below list the Solaris packages available with this release of the product, and indicate the part number for each distinct product component.

Binary Product — for Solaris Host

Part Number	**CLX401-SSU0**
Package Name	**Description**
SUNWewbu	Sun Embedded Workshop for UltraSPARC-IIi BSP source
SUNWewcd	Sun Embedded Workshop PDF Format Common Documentation
SUNWewch	Sun Embedded Workshop HTML Format Common Documentation
SUNWewcp	Sun Embedded Workshop PostScript Format Common Documentation
SUNWewdu	Sun Embedded Workshop for UltraSPARC-IIi XRAY Debugger
SUNWewgu	Sun Embedded Workshop for UltraSPARC-IIi GUI Tools
SUNWewiu	Sun Embedded Workshop for UltraSPARC-IIi IOM source
SUNWewju	Sun Embedded Workshop for UltraSPARC-IIi JVM

Part Number	CLX401-SSU0
Package Name	**Description**
SUNWewku	Sun Embedded Workshop for UltraSPARC-IIi Kernel
SUNWewm	Sun Embedded Workshop On-Line Manual Pages
SUNWewou	Sun Embedded Workshop for UltraSPARC-IIi OS
SUNWewpu	Sun Embedded Workshop for UltraSPARC-IIi Examples
SUNWewsd	Sun Embedded Workshop PDF Format Specific Documentation
SUNWewsh	Sun Embedded Workshop HTML Format Specific Documentation
SUNWewsp	Sun Embedded Workshop PostScript Format Specific Documentation
SUNWewtu	Sun Embedded Workshop for UltraSPARC-IIi Build Tools
SUNWewuu	Sun Embedded Workshop for UltraSPARC-IIi Debugger and Profiling Support
SUNWewxu	Sun Embedded Workshop for UltraSPARC-IIi X11 Library
SUNWewzu	Sun Embedded Workshop for UltraSPARC-IIi egcs Toolchain
SUNWewcab [1]	ChorusOS 4.0.1 Common Documentation Collection
SUNWewsab [1]	ChorusOS 4.0.1 Target Family Documentation Collection
SUNWewmab [1]	ChorusOS 4.0 Reference Manual Collection

1. Answerbook packages cannot be installed using the graphical installer. See the Sun document *Installing and Administering an AnswerBook2 Server* for a complete description of the AnswerBook2 documentation installation process.

Flite Add-on for Solaris Host

Part Number	**FLT401-SSU0**
Package Name	**Description**
SUNWewfu	Sun Embedded Workshop for UltraSPARC-IIi Flite

Source Add-on for Solaris Host

Part Number	**CLX401-SSU0-S**
Package Name	**Description**
SUNWewhu	Sun Embedded Workshop for UltraSPARC-IIi OS source
SUNWewlu	Sun Embedded Workshop for UltraSPARC-IIi Kernel source

Documentation for Solaris Host

Part Number	**CLX401-SAA0-D1N**
Package Name	**Description**
SUNWewcd	Sun Embedded Workshop PDF Format Common Documentation
SUNWewch	Sun Embedded Workshop HTML Format Common Documentation
SUNWewcp	Sun Embedded Workshop PostScript Format Common Documentation
SUNWewm	Sun Embedded Workshop On-Line Manual Pages
SUNWewsd	Sun Embedded Workshop PDF Format Specific Documentation

Part Number	CLX401-SAA0-D1N
Package Name	**Description**
SUNWewsh	Sun Embedded Workshop HTML Format Specific Documentation
SUNWewsp	Sun Embedded Workshop PostScript Format Specific Documentation
SUNWewcab [1]	ChorusOS 4.0.1 Common Documentation Collection
SUNWewsab [1]	ChorusOS 4.0.1 Target Family Documentation Collection
SUNWewmab [1]	ChorusOS 4.0 Reference Manual Collection