ChorusOS 4.0 Migration Guide

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

Part Number 806-4422–10
Mar 7, 2000

Adobe PostScript

Please
Recycle

Contents

Preface

ChorusOS 4.0 Migration Guide provides information about:

- The major differences between version 3.2 and version 4.0 of the ChorusOS operating system.
- Porting your existing software applications and drivers.
- How to write new software applications and drivers.

This book assumes you are familiar with the ChorusOS operating system and the C programming language.

Who Should Use This Book

Read the ChorusOS 4.0 Migration Guide if you are:

- Interested in migrating software applications or drivers from the ChorusOS 3.2 operating system to the ChorusOS 4.0 operating system.

Before You Read This Book

If you are considering migrating your applications or drivers to the ChorusOS 4.0 operating system, and you are not familiar with ChorusOS, you must first read:

- *ChorusOS 4.0 Introduction*. This book introduces the features and components of the ChorusOS operating system.

- *ChorusOS 4.0 Device Driver Framework Guide*. This book describes the device driver architecture of the ChorusOS operating system and explains how to add a new driver.

How This Book Is Organized

This book consists of the following chapters:

Chapter 1 provides an overview of the major changes between version 3.2 and version 4.0 of the ChorusOS operating system, and also lists the APIs which are new, or have changed, since version 3.2.

Chapter 2 lists the changes and new features in the design of the operating system.

Chapter 3 lists the changes and new features in the operating system libraries.

Chapter 4 lists the changes and new features in the operating system kernel.

Chapter 5 lists the changes and new features in the build tools.

Related Books

The following books contain additional information about version 4.0 of the ChorusOS operating system to help you with migration:

- The *ChorusOS 4.0 Hot Restart Programmer's Guide*. This book describes how to develop applications to use the hot restart functionality of the ChorusOS operating system.
- *ChorusOS 4.0 Flash Guide*. This book describes the support for flash memory provided in the ChorusOS operating system and explains how to use it.

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`.

What Typographic Conventions Mean

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

TABLE P–1 Typographic 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.
`Name`(Section)	`Name`(Section)	See `man`(1) for more information.

Shell Prompts in Command Examples

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

TABLE P–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	`#`

CHAPTER **1**

Overview of Enhancements to the ChorusOS 4.0 Operating System

This chapter provides an overview of the major changes between version 3.2 and version 4.0 of the ChorusOS operating system. It contains the following sections:

- Section 1.1 “Enhancements to the ChorusOS 4.0 Operating System” on page 9 lists the new features of ChorusOS 4.0.
- Section 1.2 “API Changes in ChorusOS 4.0” on page 10 summarizes the APIs which are new, or have changed, since version 3.2 of the ChorusOS operating system.

1.1 Enhancements to the ChorusOS 4.0 Operating System

Version 4.0 of the ChorusOS operating system provides the following enhancements over version 3.2:

- A full binary product:
  - Portable CPU family binary.
  - BSPs for porting on new platforms, adapting the boot method, changing and adding device drivers.
  - Reference implementations.
- Hot Restart:
  - Simplified high availability management.

  - Modular and fully configurable components.

- Enhanced TCP/IP:

  - Updated TCP/IP (BSD 2.2.8).
  - Multiple network interfaces.
  - Extended routing, IP forwarding and multicast DHCP.
  - Enhanced and simplified administration.

- Operating system configuration:

  - Independent operating system components: devices, file systems, networking, dynamic application loading.
  - Reduced memory footprint, higher modularity and improved scalability.

- Enhanced operating system features:

  - Dynamic libraries.
  - C++ exceptions.
  - Full support for flash memory.
  - Line disciplines.
  - Device file system.

- Sun Embedded WorkShop 4.0:

  - Open debugger integration framework.
  - GUI based configuration tool.
  - Uniform configuration data.
  - Uniform application, BSP and driver development environment.

1.2 API Changes in ChorusOS 4.0

Table 1–1 summarizes which APIs are new, or have changed, in ChorusOS 4.0.

TABLE 1–1 Status of APIs in ChorusOS 4.0

ChorusOS 3.2 API name	ChorusOS 4.0 API name	Status
BSD	BSD	Extended
CORE	CORE	Changed
—	DYNAMIC_LIB	New. See the DYNAMIC_LIB(5FEA) man page for more information.
RESTART	HOT_RESTART	New. See the HOT_RESTART(5FEA) man page for more information.
—	IOM_IPC	New. See the IOM_IPC(5FEA) man page for more information.
—	IOM_OSI	New. See the IOM_OSI(5FEA) man page for more information.
MEM	MEM	Changed
MEM_FLAT MEM_PROTECTED MEM_VIRTUAL	VIRTUAL_ADDRESS_SPACE FS_MAPPER	New. See the VIRTUAL_ADDRESS_SPACE(5FEA) and FS_MAPPER(5FEA) man pages for more information.
—	NFS_SERVER	New. See the NFS_SERVER(5FEA) man page for more information.
—	PERF	New. See the PERF(5FEA) man page for more information.
POSIX_MQ	POSIX_MQ	Extended
POSIX_SHM	POSIX_SHM	Extended
SCHED_CLASS	ROUND_ROBIN	Unchanged
VIRTUAL_TIME	VTIMER	Unchanged

Table 1–2 details the APIs which have changed or have been extended.

TABLE 1–2 Details of changed or extended APIs in ChorusOS 4.0

ChorusOS 4.0 API name	**Details**
`BSD`	The following functions have been added: `cfgetispeed()`, `cfgetospeed()`, `cfmakeraw()`, `cfsetispeed()`, `cfsetospeed()` and `cfsetpeed()`.
`CORE`	The Interrupt Management Service (IMS) has been removed: `f_imsIntrLevel()`, `f_imsProgLevel()`, `imsIntrMaskCount_f()`, `f_imsPreemptionCntlWord()`, `f_imsThreadEventRqst()`, `f_imsProcessSchedEvents()` and `f_imsProcessThreadEvents()`.
`MEM`	The following functions have been removed: `svPageContigAllocate()` and `svPageContigFree()`.
`POSIX_MQ`	The following function has been added: `fpathconf()`.
`POSIX_SHM`	The following functions have been added: `fchmod()`, `fchown()`, `fpathconf()` and `fstat()`.

CHAPTER **2**

ChorusOS 4.0 Operating System Design Changes

This chapter lists the changes and new features in the design of the ChorusOS 4.0 operating system.

2.1 Multiple RAM Disks

Up to 16 RAM disks are supported by the ChorusOS 4.0 operating system. This number does not include internal RAM disks used as memory banks. Configuration of multiple RAM disks is achieved by modifying the following tunables:

- `iom.ramdisk.sizeMax` specifies the maximum size (in bytes) of any RAM disk. The default value is 0x400000 (4 Megabytes). Any RAM disk whose size is greater than `iom.ramdisk.sizeMax` is reduced to the value of `iom.ramdisk.sizeMax`, and a warning message is displayed on the console when you attempt to access the RAM disk, by formatting it with the `disklabel` command for example.
- `iom.ramdisk`*x*`.size` specifies the size of RAM disk *x*, where *x* is a single hexadecimal digit between 0 and f. The default value for each instance of `iom.ramdisk.size` is –1 which means that the disk is not yet configured or usable.

These tunables can be read by the target, but not modified, using the `sysctl` command. Here are two examples:

To return the value of `iom.ramdisk.sizeMax`, the maximum size of any RAM disk, type:

```
$ arun sysctl drv.ramdisk.sizeMax
```

To return the size of RAM disk a, type:

```
$ arun sysctl drv.ramdiska.size
```

See the `sysctl`(1M) man page for more information.

After defining your RAM disks, you must associate them with a device. This is achieved using the `mknod` `C_INIT` command as follows:

```
$ mknod device_name device_type major_number minor_number
```

The *device_name* is the device name within `/dev`, *device_type* is either `c` for character, or `b` for block, *major_number* is the major device number, and *minor_number* is the minor device number. The *minor_number* determines what number is allocated to each RAM disk according to the following table:

TABLE 2–1 RAM Disk Identification

***minor_number* range**	**RAM disk number**
0 to 7	0
8 to 15	1
16 to 23	2
24 to 31	3
32 to 39	4
40 to 47	5
48 to 55	6
56 to 63	7
64 to 71	8
72 to 79	9
80 to 87	10
88 to 95	11
96 to 103	12
104 to 111	13

TABLE 2–1 RAM Disk Identification *(continued)*

***minor_number* range**	**RAM disk number**
112 to 119	14
120 to 127	15

See the mknod(1M) man page for more information.

Here is an example of device definitions for RAM disk 0:

```
# Devices for RAM disk #0
# must define both block and character modes
# Character mode
mknod /dev/rrd0a c 13 0
mknod /dev/rrd0b c 13 1
mknod /dev/rrd0c c 13 2
mknod /dev/rrd0d c 13 3
mknod /dev/rrd0e c 13 4
mknod /dev/rrd0f c 13 5
mknod /dev/rrd0g c 13 6
mknod /dev/rrd0h c 13 7

# Block mode
mknod /dev/rd0a b 14 0
mknod /dev/rd0b b 14 1
mknod /dev/rd0c b 14 2
mknod /dev/rd0d b 14 3
mknod /dev/rd0e b 14 4
mknod /dev/rd0f b 14 5
mknod /dev/rd0g b 14 6
mknod /dev/rd0h b 14 7
```

Which partitions you define will depend on which entries you have defined in your `/etc/disktab` file. See the disktab(4CC) man page for more information.

Here is an example `/etc/disktab` entry which defines RAM disk `rd1Meg` with two partitions:

```
rd1Meg:\
        :ns#4:nt#4:nc#2048 \
        :pa#1024:oa#0:ta=MSDOS: \
        :pb#1024:ob#0:tb=MSDOS: \
        :pc#2048:oc#0:tc=unused:
```

Your device definition file will look like this:

```
Devices for RAM disk rd1Meg
# must define both block and character modes
# Character mode
mknod /dev/rrd0a c 13 0
mknod /dev/rrd0b c 13 1
```

```
# Block mode
mknod /dev/rd0a b 14 0
mknod /dev/rd0b b 14 1
```

Label the RAM disk using the `disklabel` command:

```
$ arun disklabel -w rd0 rd1Meg
```

Format the partitions as follows:

```
$ arun newfs_dos /dev/rrd0a
$ arun newfs_dos /dev/rrd0b
```

Finally, to mount the partitions, do the following:

```
$ mount -t msdosfs /dev/rd0a /mnt/a
$ mount -t msdosfs /dev/rd0b /mnt/b
```

See *ChorusOS 4.0 File System Administration Guide* for more information on RAM disks.

2.2 Flash Feature

The `FLASH` feature in ChorusOS 4.0 provides an interface to access memory devices through the Flite 1.2 BSP component. Two commands are available:

- **`format`** ***raw_device***

  This command is new to ChorusOS 4.0 and performs a low-level format on the flash device. See the `format`(1M) man page for more information.

- **`flashdefrag`** ***raw_device*** **`[`*value*`]`**

  This command defragments flash memory and exists in ChorusOS 3.2 without the *raw_device* argument. See the `flashdefrag`(1M) man page for more information.

The `mkdosflashfs` and `dosfsck` commands, which create and check a DOS File system respectively, have been removed. Use the `newfs_dos` and `fsck_dos` commands instead. See the `newfs_dos`(1M) and `fsck_dos`(1M) man pages for more information.

For more information on the FLASH feature, see the FLASH(5FEA) man page. For more information on formatting a flash memory device, see "How to Format a Flash Memory Device" in *ChorusOS 4.0 File System Administration Guide*.

2.3 Swap System

In ChorusOS 3.2, the swap system was implemented over NFS. In ChorusOS 4.0, the swap system is implemented as a UFS-based file system which requires a local IDE or SCSI disk. Mount the swap system with the following command:

```
$ mount -t swap mount_point /swap
```

Swap areas greater than 2 Gigabytes are supported in ChorusOS 4.0. See "How to Activate a Swap Partition" in *ChorusOS 4.0 File System Administration Guide* for information on preparing and activating a swap partition.

2.4 Target Initialization and Administration

New support for target initialization and administration is provided in the ChorusOS 4.0 operating system:

- A new actor, `ADMIN`, provides administration utilities such as `ifconfig`, `route`, and `netmask`, which can be run before the file system is initialized.
- The `C_INIT` actor, loaded at system start-up, has a command interpreter which reads the `sysadm.ini` file embedded in the ChorusOS system image and executes any initialization instructions it finds. See the `C_INIT`(1M) man page for more information.

This is a more flexible and easy-to-configure approach than in ChorusOS 3.2, where network and file system initialization was handled by the `IOM` actor. See "System Administration in the Extended Environment" in *ChorusOS 4.0 Introduction* for more information.

You can customize system initialization using a `/etc/rc.chorus` file. See the `rc.chorus`(4CC) man page for more information.

To build a system image, you now need a `sysadm.ini` file in order to configure the network and any devices. See the `sysadm.ini`(4CC) man page for more information.

2.5 Compressed Executables

The storage space of executable files may be reduced by compressing them with the GNU zip utility `gzip`. When the `GZ_FILE` feature is enabled, the `afexec()` and `aload()` system calls will automatically uncompress `.gz` files before loading them into memory. Here is an example:

1. Compress the `ls` tool on your host:

   ```
   $ gzip ls.r
   ```

2. Run `ls` as normal, omitting the `.gz` extension:

   ```
   $ arun ls
   ```

   or

   ```
   $ arun ls.r
   ```

Dynamic libraries can also be compressed in the same way. See the `GZ_FILE`(5FEA), the `afexec`(2K), and the `aload`(2K) man pages for more information.

2.6 Running Executable Files From Memory

It is now possible to run executable files directly from memory by specifying the path to the relocatable binary in the following form:

`ram:0x`*start*`:0x`*length*`:`

start is the starting address at which the binary is stored and *length* is the length of the binary.

See the `afexec`(2K) man page for more information.

2.7 Hot Restart

Hot restart is now available on all platforms and is enabled with the `HOT_RESTART` feature. Although the essence of hot restart is the same in the ChorusOS 4.0 operating system, it has been reimplemented since the ChorusOS 3.2 operating system, with significant differences in behavior and API. See the *ChorusOS 4.0 Hot Restart Programmer's Guide* and the `HOT_RESTART`(5FEA) man page for more information.

2.8 PPP and SLIP Architecture

The PPP architecture has been redesigned in the ChorusOS 4.0 operating system, making its use and operation much closer to that used in BSD UNIX. There is no longer a specific API to dynamically manage serial lines as there was in the ChorusOS 3.2 operating system.

The `IOM` actor now includes standard `tty` line discipline functionalities, as well as support for pseudo ttys, which are devices that are not associated with any real hardware. They are used to simulate the functions of a real tty. A `tty` line can be opened in raw mode, then switched to PPP or SLIP line discipline using standard `ioctl()` calls. This is used by the PPP or SLIP daemon to configure the serial line for the PPP or SLIP protocol.

The ChorusOS 4.0 PPP daemon is an improved version of the BSD daemon, which is able to handle multiple serial lines by assigning a separate thread to each one. All the usual PPP functions, including dial-up on demand, are supported through the `pppd` options.

The PPP API is provided by the actor `pppstart` and three `C_INIT` commands, `pppd`, `pppstop`, and `pppclose`. See the `pppstart`(1M), `pppd`(1M), `pppstop`(1M), and the `pppclose`(1M) man pages, and "Setting Up PPP" in *ChorusOS 4.0 Network Administration Guide* for more information.

2.9 POSIX Message Queues

The `mq_open()` system call establishes the connection between an actor and a message queue with a message queue descriptor. In the ChorusOS 3.2 operating system, the identifier returned by `mq_open()` was from an identifier space specific to the POSIX message queue. In the ChorusOS 4.0 operating system, this identifier is a standard file descriptor, similar to the file descriptor returned by `open()`, as stated in the POSIX standard.

See the `mq_open`(2POSIX) man page for more information.

2.10 BSD Compatible Interfaces

The 4.3 BSD compatible interfaces in the ChorusOS 3.2 operating system have been replaced by 4.4 BSD compatible interfaces in the ChorusOS 4.0 operating system. IOM is no longer compiled with the `COMPAT_43` option.

As a result of this change, the `sockaddr` structure has a new field, `sa_len`:

```
struct sockaddr {
    u_char sa_len;    /* total length */
    u_char sa_family; /* address family */
    char sa_data[14]; /* actually longer; address value */
}
```

Any applications using the old form of `sockaddr`, without the `sa_len` field, must be updated to initialize the structure correctly.

2.11 RTTY

The raw serial line device feature, `RTTY`, has been removed from the ChorusOS 4.0 operating system. Line disciplines, used to control input from the tty, have been implemented instead. The following line disciplines are available:

- General Line Discipline.
- PPP Line Discipline.
- SLIP Line Discipline.

See the `tcsetattr`(3POSIX) and `tcgetattr`(3POSIX) man pages for more information.

2.12 POSIX_FILEIO

`POSIX_FILEIO`, the POSIX compliant I/O system calls feature, has also been removed from the ChorusOS 4.0 operating system as compliant I/O system calls have been integrated into the file system.

For information on POSIX compliant I/O system calls on top of the MSDOS File System, see the `MSDOSFS`(5FEA) man page. For information on POSIX compliant I/O system calls on top of the UNIX File System, see the `UFS`(5FEA) man page. For information on POSIX compliant I/O system calls on top of the Network File System implementation, see the `NFS_CLIENT`(5FEA) man page.

CHAPTER **3**

ChorusOS 4.0 Library Changes

This chapter lists the changes and new features in the ChorusOS 4.0 libraries.

3.1 `strsep()`

It is no longer possible to modify string constants in the ChorusOS 4.0 operating system. As a result, the `strsep()` function will cause a segmentation fault if called with a pointer to a literal string. This is demonstrated in Code Example 3–1.

CODE EXAMPLE 3–1 Code example of `strsep()` causing a segmentation fault

```
int main(int argc, char **argv)
{
    char *s = "aaaa/bbbb";
    char *r;
    char **sp = &s;

    r = strsep(sp, "/"); /* a segmentation fault is raised */
}
```

Code Example 3–1 does not cause a segmentation fault in the ChorusOS 3.2 operating system because the `gcc .rodata` section was mapped to a writable memory region. In the ChorusOS 4.0 operating system, the gcc `.rodata` section is mapped to a read-only memory region.

Code Example 3–1 can be corrected by calling `strsep()` with a non-literal string, shown in Code Example 3–2.

CODE EXAMPLE 3–2 Corrected code example using `strsep()`

```
int main(int argc, char **argv)
{
    char *s = "aaaa/bbbb";
    char *r;
    char **sp = &s;
    char *tmp;

    tmp = strdup(s);

    if (tmp == NULL) {
        printf("out of memory\n");
        return 0;
    }

    sp = &tmp;
    r = strsep(sp, "/"); /* this works */
}
```

See the `strsep`(3STDC) man page for more information.

3.2 malloc()

The standard memory allocation package, including the `malloc()` and `free()` functions have been reimplemented in the ChorusOS 4.0 operating system. To maintain compatibility with ChorusOS 3.2 `malloc()`, use the `lib/classix/libomalloc.a` library.

The package is based on the Solaris `libc` implementation and extended to release freed memory to the operating system. Yet, calling `free()` does not automatically return memory to the system. Since memory chunks are allocated by `malloc()` from page-aligned regions, these regions are only returned when all the chunks in the region have been freed. Furthermore, `free()` buffers memory chunks so that they can be reused by `malloc()`. As a result, memory will not be returned to the operating system until `malloc()` is called again. Use `malloc_trim()` to explicitly release this memory.

See the `malloc`(3STDC) man page for more information.

3.3 getpass()

The `getpass()` system call has been removed from the ChorusOS 4.0 operating system. As host and target communications are based on commands sent by `rsh` from the host to the target, the echoing is carried out by the host and cannot be controlled by the target. As a result, it is not possible to silence password echoing. To avoid password echoing, do not use `rsh` for host/target communications.

CHAPTER **4**

ChorusOS 4.0 Kernel Changes

This chapter lists the changes and new features to the ChorusOS 4.0 kernel.

4.1 Boot Framework

The boot framework has been reimplemented in the ChorusOS 4.0 operating system. See the *ChorusOS 4.0 Porting Guide* for detailed information.

4.2 Boot Process

The ChorusOS 3.2 `netboot` utility, to load and boot ChorusOS system images, has been replaced by `bootMonitor` in the ChorusOS 4.0 operating system. Whereas `netboot` is only available for Intel platforms, and designed to be run from diskette, `bootMonitor` is available on all platforms and can be run from diskette or flash memory.

The file format of system images has changed from SVR5 in the ChorusOS 3.2 operating system to ELF in the ChorusOS 4.0 operating system. System images created in the ChorusOS 3.2 operating system are not designed to boot in the ChorusOS 4.0 operating system.

See the `bootMonitor`(1CC) man page, and the appropriate book in the *ChorusOS 4.0 Target Family Documentation Collection* for further information.

4.3 Processor Family Specific I/O

Processor family specific I/O routines, previously located in a CPU library in the ChorusOS 3.2 operating system, are now part of the kernel DKI module and bus drivers in the ChorusOS 4.0 operating system. Device drivers must not communicate directly with the hardware, but must communicate through the DKI and appropriate bus DDI interface using the `*busIoOps.load_8()`, `*busIoOps.store_8()` calls.

See the *ChorusOS 4.0 Device Driver Framework Guide* for detailed information.

4.4 System Calls

The following system calls have been removed from the ChorusOS 4.0 operating system:

TABLE 4–1 Obsolete System Calls

Obsolete ChorusOS 3.2 System Call	Reason for removal	Equivalent ChorusOS 4.0 System Call
`actorPersistency()`, `actorPersistCap()`, `actorRestartCapability()`	The revised hot restart interface no longer relies on persistent actor support at kernel level.	None.
`rgnPhysMap()`	The ChorusOS 4.0 operating system does not support device memory on demand mapping in the user address space.	None.
`sysBench()`	Superseded.	This call has been superseded by the `sysTimerStartFreerun`(2K) and `sysTimerReadCounter`(2K)system calls.
`svProfHandler()`	Superseded.	This call has been superseded by the `sysTimerStartPeriodic`(2K) system call.

TABLE 4–1 Obsolete System Calls *(continued)*

Obsolete ChorusOS 3.2 System Call	**Reason for removal**	**Equivalent ChorusOS 4.0 System Call**
`svMaskAll()`, `svUnmask()`, `svUnmaskAll()`	Superseded.	The supervisor calls have been superseded by the `imsIntrMask_f`(9DKI) and `imsIntrUnmask_f`(9DKI) system calls.
`svSysIo()`	The system console driver is implemented by the external `dbgBsp` component.	None.

4.5 KBIM Replacement

The KBIM kernel module has been removed in the ChorusOS 4.0 operating system, taking with it the following services:

```
svDeferInt()
svIntrGetCurHdl()
svIntrSetMngt()
svIntrGetMngt()
svPicMaskAll()
svPicUnmask()
svPicTestIntr()
svUnDeferIntr()
svReleaseIntr()
```

Interrupt management is now handled by the kernel DKI module and the bus drivers. Device drivers must not communicate directly with the hardware, but must communicate through the DKI and appropriate bus DDI interface.

Table 4–2 lists ChorusOS 4.0 equivalent DKI and bus DDI interfaces for ChorusOS 3.2 KBIM services.

TABLE 4–2 Replacement DKI and bus DDI interfaces for KBIM services

ChorusOS 3.2 KBIM Service	**ChorusOS 4.0 DKI**	**ChorusOS 4.0 Bus DDI Interface**
`svIntrConnect( )`	`svIntrAttach( )`	`*BusOps.intr_attach( )`
`svIntrDisconnect( )`	`svIntrDetach( )`	`*BusOps.intr_detach( )`
`svPicMaskIntr( )`	N/A	`*BusIntrOps.mask( )`
`svPicUnmaskIntr( )`	N/A	`*BusIntrOps.unmask( )`

See "Driver Kernel Interface Overview" in *ChorusOS 4.0 Device Driver Framework Guide* for more information.

4.6 PowerPC Board Configuration

There are differences in the way the PowerPC MVMExxxx board family is configured in the ChorusOS 4.0 and ChorusOS 3.2 operating system.

Here are the original settings for the ChorusOS 3.2 operating system:

- Boot File Load Address = 001F0000 (set with the `niot` command).
- Boot File Execution Address = 001F0000 (set with the `niot` command).
- Network PReP-Boot Mode Enable = Y (set with the `env` command).

Here are the settings you will need for the ChorusOS 4.0 operating system:

- Boot File Load Address = 00400000 (set with the `niot` command).
- Boot File Execution Address = 00400000 (set with the `niot` command).
- Network PReP-Boot Mode Enable = N (set with the `env` command).

See the *ChorusOS 4.0 PowerPC 60x/750 Target Family Guide* for more information.

CHAPTER **5**

ChorusOS 4.0 Build Tool Changes

This chapter lists the changes and new features to the ChorusOS 4.0 build tools.

5.1 New Build Tools

Included as part of Sun Embedded Workshop is a new graphical configuration tool called `ews`. This tool provides a user-friendly interface for configuring the operating system and shows the dependencies between components. A command-line interface for configuration, called `configurator`, is also available. See “Configuration Tools” in *ChorusOS 4.0 Introduction*, *ChorusOS 4.0 Production Guide* and the `ews`(1CC) and `configurator`(1CC) man pages for more information.

The `mkconfig`, `mkenv` and `mktune` commands, included in ChorusOS 3.2, are no longer provided. Any scripts using these commands must be adapted to use the `configurator` command.

5.2 System Configuration Files

System configuration information is no longer held in a single file, called `config`, as it was in the ChorusOS 3.2 operating system. In the ChorusOS 4.0 operating system, configuration information is spread across several files within the `conf` directory of the ChorusOS build directory. As a result, you must copy the entire `conf` directory to transfer the configuration of one system image build directory to another.

See “Configuration Files” in *ChorusOS 4.0 Introduction* for a breakdown of the files within the `conf` directory.

5.3 `imake` Build Rules

New `imake` build rules are provided with the ChorusOS 4.0 operating system as a single file called `Imake.rules`. If you wish to use the `imake` build rules of the ChorusOS 3.2 operating system, integrate the `Imake32.rules` file with your `Project.tmpl` file.

See "Developing Applications" in *ChorusOS 4.0 Introduction* for more information about `imake` build rules.

Index