Developing OpenStep Applications Using NEXTSTEP 3.2 (Preliminary)

NineInchRule.eps ¬

BothLogosFront.tiff ¬

SunSoft Part No: 801-7650-01
Revision 02, April 1994

NeXT Part No: 6452.00
Revision 02, April 1994

FullRule.eps ¬

Solaris Porting Guide by SunSoft ISV Engineering, Michele Ann Goodman, Manoj Goyal, and Robert A. Massoudi (Published by SunSoft Press and PTR Prentice Hall, ISBN 0-13-030396-8. This book is available at local bookstores, or can be ordered directly from Prentice Hall by calling (201) 592-2863

ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. OR NEXT COMPUTERS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

747917_NineInchRule.eps ¬

Table of Contents

617471_NineInchRule.eps ¬

1. Overview of OpenStep

OpenStep is a specification currently under development by NeXT Computer with support from SunSoft. This specification will be published no later than June 30, 1994.

The OpenStep specification, based on NEXTSTEP Release 3.2, will provide application portability across multiple implementations of OpenStep, whether based on NEXTSTEP, Solaris, or another implementation developed independently. There are two differences between OpenStep and the NEXTSTEP product:

1. A number of interfaces that are part of NEXTSTEP Release 3.2 will be *excluded* from OpenStep.
2. A number of interfaces that are part of NEXTSTEP Release 3.2 will be *modified* for OpenStep.

Both SunSoft and NeXT Computer will produce OpenStep implementations shortly after the specification is published. Other vendors are also expected to produce OpenStep-compliant implementations.

1.1 How OpenStep Fits Into Project DOE

The Project DOE technology represents SunSoft's support of open systems in the new object paradigm. Project DOE technology is a major SunSoft initiative to bring the leverage of open systems to the application arena; greatly increasing the same open systems benefits achieved in Sun's hardware and UNIX SVR4 products. It extends the Solaris model with the full range of capabilities, services and partnerships required by an

enterprise-ready, distributed object environment.

Project DOE technology enables you to create new kinds of applications based on the distributed object model. Applications, in general, are built differently when using Project DOE technology.

When designing an application for the Project DOE environment, you must define and use standardized interfaces for subcomponents of an application. In the object paradigm, the developer assembles components into applications (objects). This means it is the developer, not the user, who decides which components inter-operate.

Through Project DOE technology, SunSoft has a major strategic commitment to deliver the power of the distributed object paradigm with an evolutionary growth path for current applications. Project DOE technology encourages the innovation and rapid development of software to meet user application needs. Its focus is on the distributed architecture to structure computer resources to meet these requirements.

Project DOE essentially consists of two parts: an OMG-compliant distributed infra-structure and an OpenStep-compliant application environment based on NeXTSTEP 3.2.

OpenStep is a major portion of Project DOE and will greatly accelerate the delivery of the Project DOE object-oriented technology. The OpenStep technology will be integrated with SunSoft's distributed object infrastructure, enabling fully-distributed enterprise-wide object applications. The NEXTSTEP technology is proven and well established as an effective set of object interfaces; and the NEXTSTEP application environment is widely acclaimed to yield the highest application development productivity in the industry. The OpenStep specification is complementary with the OMG CORBA standard. CORBA is also an X/Open XPG4 Preliminary Specification. In addition, all of the Common Open Software Environment (cose) partners have embraced the OMG standards and pledged

to deliver products based on them. NeXT and SunSoft will work together to propose the entire OpenStep specification to the appropriate industry standards body, such as OMG or X/Open, for adoption as a vendor-independent specification.

1.2 How OpenStep Fits Into the Solaris System

SunSoft is also fully committed to the success of cose and the common desktop environment (CDE). As shown in Figure 1-1, the new object-oriented application environment will co-exist on Solaris systems with the cose/CDE environment. Applications written to the OpenStep specification on Solaris will inter-operate with CDE applications.

paste.tiff ¬

Figure 1-1 OpenStep on Solaris

1.3 For Whom is this Paper Written?

There is no single development environment that is perfect for everybody. The benefit of the OMG architecture is that it allows the easy integration of components developed in different development environments (different languages, different underlying systems, and from different vendors).

This paper discusses some of the issues developers need to consider when using the SunSoft OpenStep Developer Starter Kit to develop new applications for, or port existing applications to, OpenStep on Solaris. The SunSoft OpenStep Developer's Starter Kit is

based on the NEXTSTEP Release 3.2 Developer Kit and provides facilities that are available in NEXTSTEP Release 3.2. However, some of these facilities are either not available in OpenStep on Solaris, or are available with different interfaces in OpenStep on Solaris. This paper highlights some of these differences in functionality.

This paper assumes familiarity with UNIX, Mach, and NEXTSTEP.

CautionÐThis paper is not intended to be a transition guide to the OpenStep development environment. The purpose of this paper is to provide development guidelines that will make it easier for you to port your applications to the OpenStep environment on Solaris when it becomes available. The OpenStep Developer Starter Kit is *not* guaranteed to be upwardly compatible with the OpenStep release. *All* OpenStep application programming interfaces (APIs) and formats are subject to change. Also, some features that are part of the OpenStep Developer Starter Kit may either not be available in OpenStep on Solaris, or will be available with different interfaces.

1.4 How this Paper is Organized

This paper is organized in two sections: ªComparing OpenStep and NEXTSTEP Release 3.2 Functionsº and ªPreparing for OpenStep on Solaris.º

1.5 Recommended Books to Help Make Porting Easier

Solaris Porting Guide by SunSoft ISV Engineering, Michele Ann Goodman, Manoj Goyal, and Robert A. Massoudi (Published by SunSoft Press and PTR Prentice Hall, ISBN 0-13-030396-8. This book is available at local bookstores, or can be ordered

directly from Prentice Hall by calling (201) 592-2863.

224817_NineInchRule.eps ¬

2. Comparing OpenStep and NEXTSTEP Release 3.2

The OpenStep specification is being derived from NEXTSTEP Release 3.2. The specification is still under development but its general outline is clear. From the user's point of view, an OpenStep implementation will be identical to NEXTSTEP Release 3.2; from the developer's point of view, the programming environments will be similar, but some differences will be evident.

The differences between the OpenStep and the NEXTSTEP Release 3.2 APIs fall into two groups:

1. Interfaces that are part of NEXTSTEP Release 3.2 but will be modified for OpenStep.
2. Interfaces that are part of NEXTSTEP Release 3.2, but will not be available to OpenStep developers.

This section describes these differences.

CautionÐAll OpenStep application programming interfaces (APIs) and formats are subject to change.

o 2.1 Interfaces to be Modified for OpenStep

Only minor API differences exist in the software that OpenStep and NEXTSTEP Release 3.2 have in common. The modifications were motivated by a goal to make OpenStep APIs as clear, powerful, and consistent as possible.

- OpenStep defines only the best approach to accomplish a task.
- OpenStep uses operating system-independent APIs.
- A consistency in names for similar methods prevail.

The similarities and differences between the OpenStep and NEXTSTEP Release 3.2 components are summarized in Table 2±1, and discussed in more detail in the following subsections.

Table 2±1 Comparing Components

892825_NineInchRule.eps ¬

OpenStep Component	**Compared to NEXTSTEP Release3.2 Component**
Application Kit	Relatively minor changes throughout: API refinement and regularization
Common Classes	Enhanced functionality
Database Kit	Minor modifications
Display PostScript	Assumes Adobe's Display PostScript™ client library; Includes NEXTSTEP Release 3.2 compositing operators
Distributed Objects System	Minor modifications
Indexing Kit	Storage and B-tree components retained; file system indexing component removed.
Mach Kit	Minor modifications

Objective C Run-time System Minor modifications

434131_NineInchRule.eps ¬

OpenStep will provide tools to help you update your NEXTSTEP source code to these new programming interfaces.

NoteÐNEXTSTEP development tools (such as Interface Builder, Project Builder, the Objective C++ compiler, and the debugger) are not discussed in this paper because they are not part of the OpenStep specification.

Application Kit

The Application Kit defines the essential components, structure, and functionality of a NEXTSTEP application. The objects it defines are:

- Manage event-handling, drawing, and printing
- Provide standard user-interface components (such as Windows, Buttons, Font panels, and Text objects)
- Provide text-editing functionality with Text object, spell-checker, and font panel
- Coordinate inter-application communication: cut-and-paste, drag-and-drop, inter-application services, and object-linking capabilities

In OpenStep

The OpenStep Application Kit is very similar to the NEXTSTEP Application Kit except for the following changes:

- Minor API refinements throughout
- Removal of obsolete classes such as Listener, Speaker, and NXJournaler

Header Location

/NextDeveloper/Headers/appkit

Common Classes

The Common Classes define objects for managing data. Primary classes in this group include HashTable, List, and Storage.

In OpenStep

The OpenStep version of the Common Classes extends the functionality of the NEXTSTEP Common Classes.

Header Location

/NextDeveloper/Headers/objc

Database Kit

The Database Kit is a set of object-oriented resources you use to create applications that will work with industry-standard database servers such as those provided by Oracle Corporation and Sybase, Inc. The Database Kit allows you to design front-end applications that are easy to build and maintain, that can communicate with other applications, and that draw upon the standard interface features used by all NEXTSTEP applications.

In OpenStep

The OpenStep version of the Database Kit is similar to the NEXTSTEP version of the Database Kit except for minor refinements in the API.

Header Location

/NextDeveloper/Headers/dbkit

Display PostScript

The Display PostScript client library, developed by NeXT Computer and Adobe Systems Incorporated, extends the PostScript language and adapts it to the interactive requirements of the screen. The NEXTSTEP implementation extends the Display PostScript language further by adding operators for window management, compositing, and transparency.

In OpenStep

The OpenStep specification assumes the existence of Adobe's Display PostScript client library. OpenStep adds the NEXTSTEP compositing and transparency operators to this API. OpenStep does not include the NEXTSTEP window management operators, since the Application Kit provides access to this functionality at a more appropriate level.

Header Location

/NextDeveloper/Headers/dpsclient

Distributed Objects System

The Distributed Objects System provides a straightforward way for applications to communicate with one another using the same Objective C messaging that is used within a single application. The applications can reside on different machines across a network.

In OpenStep

The API of the Distributed Objects System is similar in OpenStep and NEXTSTEP; however, implementations differ because of the different underlying operating systems.

Header Location

/NextDeveloper/Headers/remote

Indexing Kit

At the lowest level, the Indexing Kit provides storage and retrieval capabilities (based on a B-tree algorithm) for untyped data. Built upon this layer are facilities for indexing and searching files in the file system.

In OpenStep

OpenStep includes the storage and B-tree components of the Indexing Kit; it does not include the file system searching facility (as declared in the header file /`NextDeveloper/Headers/indexing`).

Header Location

/NextDeveloper/Headers/btree

/NextDeveloper/Headers/store

Mach Kit

The Mach Kit provides an object-oriented interface to some of the features of the underlying operating system. It is used primarily in conjunction with the Distributed Objects System to enable the passage of messages between processes.

In OpenStep

The OpenStep and NEXTSTEP versions of the Mach Kit API are similar, with the exception that the Mach-specific API has been removed. OpenStep will make greater use of this kit to ensure application portability.

Header Location

/NextDeveloper/Headers/machkit

Objective C Run-Time System

The Objective C run-time functions are a kind of operating system for the Objective C language, implementing its messaging system and enabling dynamic loading and archiving of objects.

In OpenStep

The OpenStep and NEXTSTEP versions of the Objective C run-time system have identical functionality; however, the Mach-specific run-time functions have been removed from the OpenStep version.

Header Location

/NextDeveloper/Headers/objc

o 2.2 Interfaces Not Available to OpenStep Developers

Several of the NEXTSTEP kits and software libraries that are part of NEXTSTEP Release 3.2 will *not* be available to OpenStep developers. These kits either are too new to include in a standard for an object-oriented application layer, or they address features specific to hardware supplied by NeXT Computer. Table 2±2 lists these kits and software libraries.

Table 2±2 Kits and Software Libraries Not Available in OpenStep

113450_NineInchRule.eps ¬

Name of Kit	Name of Software Library
NetInfo Kit	MIDI Driver API
Driver Kit	Novell NetWare networking
3D Graphics Kit	Sound Kit and Sound functions
	Interface Builder API
	Preferences API

312899_NineInchRule.eps ¬

NoteÐThis list is subject to change.

387538_NineInchRule.eps ¬

3. Preparing for OpenStep on Solaris

3.1 Mach and UNIX Compatibility and Differences

Mach is the operating system of all systems that run the NEXTSTEP product. Although implemented differently than UNIX4.3BSD, it provides UNIX4.3BSD compatibility. Mach system calls are upwardly compatible with UNIX4.3BSD system calls; and Mach also supports UNIX4.3BSD commands. However, while Mach retains UNIX4.3BSD functionality, it departs from current UNIX design. Both Mach's compatibility with UNIX4.3BSD and its differences from the UNIX4.3BSD are important factors to consider when preparing applications for use in the OpenStep environment.

3.2 Developing New OpenStep Applications Using NEXTSTEP Release 3.2

If you are developing new applications using NEXTSTEP Release 3.2, use the BSD 4.3 functions provided as part of Mach whenever possible. If you must use Mach functions, try to isolate them into modules. Isolating Mach functions in this manner will make it easier for you to identify and replace this code when you port to OpenStep on Solaris.

See the "Preparing Existing NeXT Applications with Mach Functions" section for

information about some of the porting issues you will need to resolve when porting your application to Solaris if your application includes Mach functions.

NoteÐTo assure that your application ports easily to OpenStep on Solaris, we strongly recommend that you do *not* use Mach functions. *No* guarantee is made that you will not have to rewrite the sections of your code that contain Mach functions. We recommend the "Solaris Porting Guide" for any porting issues that are related to porting UNIX 4.3BSD to Solaris.

3.3 Preparing Existing NeXT Applications without Mach Functions

If your existing applications are written using NEXTSTEP Release 3.2 but do not contain any Mach functions, there are no special porting issues relating to the NEXTSTEP functionality to consider at this time. The OpenStep specification will address how these applications are to be ported.

3.4 Preparing Existing NeXT Applications with Mach Functions

If your existing applications are written using NEXTSTEP Release 3.2 *and* contain any Mach functions, there are a number of special porting issues relating to the Mach functionality that you will need to consider. This section includes a list of some of the

Mach functions that will make porting your application to OpenStep on Solaris difficult, and provides guidelines to help you port this code from NEXTSTEP to OpenStep on Solaris.

3.5 Mach Functions for Which a Similar UNIX Facility Exists

This section includes a list of some of the Mach functions for which a similar UNIX facility exists.

NoteÐThe Solaris recommendation appears in *italics* for emphasis.

Mach Virtual Memory

The Mach virtual memory facilities give a program direct access to pages or objects of virtual memory. A program can allocate, deallocate and protect these pages. Unlike memory allocated with the Unix `malloc()` call, allocation may occur anywhere in the address space of a Mach task. In fact, Mach allows a program to allocate virtual memory in the address space of another task. *Solaris gives a program access to virtual memory objects with the `mmap(2)` system call, which establishes a mapping between a process's address space and a virtual memory object.*

Both Mach and Solaris allow the program to specify how virtual memory is to be accessed, whether or not it is to be shared, and how the pages are to be protected. If you adhere to the basics of virtual memory allocation, deallocation and protection, your code will port with a minimum of effort.

Mach also provides a function `map_fd()` which maps a file into virtual memory. *Use the Solaris `mmap(2)` call to achieve similar functionality in Solaris.*

Table 3±1 lists the Mach virtual memory functions and the names of the corresponding functions in Solaris.

Table 3±1 Allocating, De-allocation, and Accessing Virtual Memory
516645_NineInchRule.eps ¬

Mach Function	Description	Similar Solaris Function
vm_allocate()	Allocate virtual memory	mmap(2)
vm_deallocate()	Deallocate virtual memory	munmap(2)
vm_protect()	Specify protection on a range of virtual memory pages	mmap(2), mprotect(2)
vm_inherit()	Specify inheritance characteristics of virtual memory	not applicable
vm_read()	Read virtual memory of the specified task	not applicable
vm_write()	Write virtual memory of the specified task	not applicable
vm_copy()	Copy virtual memory of specified task	not applicable
vm_region()	Return description of characteristics of a range of virtual memory	not applicable
vm_set_policy()	Specify paging policy for a region of virtual memory	vm_advise(3)
vm_statistics()	Examine virtual memory statistics	none

337092_NineInchRule.eps ¬

CautionÐUsing Mach-specific memory protections via `vm_protect()`, or using the

Mach function `vm_set_policy()` to affect how the kernel treats the memory may make porting to Solaris difficult.

Mach Tasks

In Mach, an executing program consists of at least one task and some number of threads. There is a clear distinction between a task and its threads. The task contains resources associated with the program but is not an executing entity. Mach threads are the entities associated with execution. However, a UNIX process contains resources and it also executes. A single Mach task with one thread is the approximate equivalent of a single UNIX process. (In fact, the `fork()` system call available under Mach does both create a task and start a new thread in that task.)

To make your Mach code more portable to Solaris, *use the process related calls whenever possible*. If your application needs to execute multiple threads, see the "Mach Kernel Threads and C-threads" section of this document.

For portability, you should avoid the Mach task functions listed in Table 3±2 whenever possible.

Table 3±2 Mach Task Functions

HalfRule.eps ¬

task_assign()

task_assign_default()

task_by_unix_pid()

task_create()

task_get_assignment()

task_get_special_port()

task_set_special_port()

task_self()

task_notify()

task_get_notify_port()

task_set_notify_port()

task_get_exception_port()

task_set_exception_port()

task_get_bootstrap_port()

task_set_bootstrap_port()

task_info()

task_priority()

task_resume()

task_suspend()

task_terminate()

636863_HalfRule.eps ¬

NoteÐThe listed functions, `task_self()` and `task_notify()` are actually references to special communications ports in Mach and are frequently used in Mach functions. If you use many Mach functions, you will probably not be able to avoid their usage; however, *`task_self()` is roughly analogous to using `getpid()` under Solaris*.

Mach Kernel Threads and C-threads

As mentioned in the "Mach Tasks" section, a Mach program consists of some number of tasks and threads. The Mach thread is the basic unit of execution that is scheduled to run in the kernel. *Solaris provides a similar concept with its lightweight processes (LWPs) as the scheduled unit of execution. The Solaris model also provides userlevel threads as lightweight abstractions*. The Solaris threads library implements its threads via LWPs. The library automatically maps threads onto some number of LWPs. In addition, it gives the program facilities to alter the mapping of threads on LWPs if this is desired.
Mach provides two programmatic interfaces to its threads, the lower level API to Mach kernel threads and a higher level interface known as C-threads. The C-threads API is a more portable C interface which implements such concepts as:

- Forking and joining of threads
- Critical region locks
- Thread synchronization primitives

The Solaris threads library provides constructs to manipulate threads that correspond closely to the Cthreads library. If you write an application that uses threads on Mach, use the C-threads API instead of the Mach kernel thread interface whenever possible.

Table 3±3 lists the lowlevel Mach thread functions that you should avoid to make your code more portable.

Table 3±3 Mach Kernel Thread Functions
412132_HalfRule.eps ¬
thread_abort()

thread_assign()

thread_assign_default()

thread_create()

thread_get_assignment()

thread_get_special_port()

thread_set_special_port()

thread_self()

thread_reply()

thread_get_reply_port()

thread_set_reply_port()

thread_get_exception_port()

thread_set_exception_port()

thread_get_state()

thread_set_state()

thread_info()

thread_policy()

thread_priority()

thread_max_priority()

thread_resume()

thread_suspend()

thread_switch()

thread_terminate()

458868_HalfRule.eps ¬

NoteÐThe listed functions, `thread_self()` and `thread_reply()` are actually references to special communications ports in Mach and are frequently used in Mach functions. If you use many Mach functions, you will probably not be able to avoid their usage; however, *`thread_self()` is analogous to `thr_self()` under Solaris*.

Table 3±4 lists the Mach C-threads API and the corresponding threads library functions available under Solaris.

Table 3±4 Mach C-threads and Solaris Thread Functions
277072_NineInchRule.eps ¬

Mach C-threads Function	Description	Solaris Threads Functions
condition_alloc()	Allocate condition variable	not applicable
condition_broadcast()	Broadcast a condition	cond_broadcast()
condition_clear()	Clear a condition	cond_destroy()
condition_init()	Initialize a condition	cond_init()
condition_free()	Free condition variable	not applicable
condition_name()	Get name of a name	none
condition_set_name()	Set name of a condition	none
condition_signal()	Signal a condition	cond_signal()
condition_wait()	Wait on a condition	cond_wait() cond_timedwait()
cthread_abort()	Interrupt a thread	thr_kill()

cthread_count()	Number of threads in the task	none
cthread_data()	Get thread specific data	thr_getspecific()
cthread_set_data()	Set thread specific data	thr_keycreate() thr_setspecific()
cthread_detach()	Detach a thread	thr_create()
cthread_errno() for each thread	Get errno for current thread	errno variable is valid
cthread_exit()	Exit a thread	thr_exit()
cthread_fork()	Fork a new thread	thr_create()
cthread_join()	Join threads	thr_join()
cthread_limit()	Get maximum number of threads in this task	thr_getconcurrency()
cthread_set_limit()	Set maximum number of threads in this task	thr_setconcurrency()
cthread_name()	Get the name of this thread	none
cthread_set_name()	Set the name of the thread	none
cthread_priority()	Set base priority of a thread	thr_setprio()
cthread_max_priority()	Set maximum priority of a thread	none
cthread_self()	Return callers thread identifier	thr_self()
cthread_set_errno_self()	Set errno for current thread	errno
cthread_thread()	Return mach thread identifier for this c-thread	thr_self()
cthread_yield()	Yield to another thread	thr_yield()

mutex_alloc()	Allocate a mutex variable	not applicable
mutex_clear()	Clear a mutex variable	mutex_destroy()
mutex_free()	Free memory associated with a mutex variable	not applicable
mutex_init()	Initialize mutex variable	mutex_init()
mutex_lock()	Lock a mutex	mutex_lock()
mutex_name()	Get the name of a mutex	none
mutex_set_name()	Set the name of a mutex	none
mutex_try_lock()	Try a mutex lock	mutex_trylock()
mutex_unlock()	Unlock a mutex	mutex_unlock()

531515_NineInchRule.eps ¬

In addition to these thread-related calls, Solaris provides a set of functions for manipulating semaphores in threads and another set for implementing multiple reader, single writer locks. These functions are shown in Table 3±5.

Table 3±5 Other Solaris Thread Functions

433062_HalfRule.eps ¬

sema_init()

sema_destroy()

sema_wait()

sema_trywait()

sema_post()

rwlock_init()

rwlock_destroy()

rw_rdlock()

rw_wrlock()

rw_unlock()

rw_trydlock()

rw_trywrlock()

746188_HalfRule.eps ¬

Mach InterProcess Communication (IPC)

Mach is a message-based operating system. All communication between two tasks, or between a task and the kernel, is via Mach messaging. Mach introduces the concept of a port or communication channel to achieve messaging. All messaging is accomplished by sending a message to a port.

Mach extends the local IPC concept to seamlessly message not only to another task on the same system but to a task on another host across a network. The fact that the receiver may be remote is transparent to the sender. In addition, Mach integrates the virtual memory system with its concept of IPC. Mach tasks can send large amounts of message data to another task efficiently because the data is merely mapped into the receiver's address space, no data is physically copied.

For portability, use either BSD IPC facilities available under Mach or the NEXTSTEP Distributed Object interface instead of using the Mach IPC directly. *The basic IPC mechanisms available under Solaris are: pipes, named pipes, sockets, signals, messages, semaphores, and shared memory*.

Solaris also provides remote procedure call facilities for distributed application: currently, RPC and Transport Independent RPC are available, as well as the ToolTalk® service, the standard messaging facility for CDE.

3.6 Mach Functions that Do Not Map Directly to UNIX Facilities

This section includes a list of the Mach functions that do not map directly to UNIX facilities. If you use the Mach functions listed in this section, it will make porting your application to OpenStep on Solaris difficult.

NoteÐOpenStep on Solaris developers are aware of these issues and working to find a solution for them. However, *no* guarantee is made that you will not have to rewrite your code if your application contains these Mach functions.

Mach Port Functions

As noted in the "Mach InterProcess Communication (IPC)" section, the Mach port is the basic communication object. If you program directly to the Mach IPC functions, you will not be able to avoid some use of the Mach port functions. It is important to note that heavy dependence on the port concepts may make your code difficult to port.

To make your code easier to port, you should only use simple port allocation and deallocation to send a message whenever possible. The Mach port functions are detailed in Table 3±6.

Table 3±6 Mach Port Functions

144270_HalfRule.eps ¬

port_allocate()

port_deallocate()

port_extract_receive()

port_extract_send()

port_insert_receive()

port_insert_send()

port_names()

port_rename()

port_set_add()

port_set_allocate()

port_set_backlog()

port_set_backup()

port_set_deallocate()

port_set_remove()

port_set_status()

port_status()

port_type()

bootstrap_check_in()

bootstrap_create_service()

bootstrap_info()

bootstrap_look_up()

bootstrap_look_up_array()

bootstrap_register()

bootstrap_status()

bootstrap_subset()

netname_check_in()

netname_check_out()

netname_look_up()

544575_HalfRule.eps ¬

Mach Exception Handling

Mach provides a model for exception handling that is message-based. Exceptions are sent to a thread on a special exception port. This exception handling is unlike Solaris exception handling, which is based on SVR/4 signals. Avoid the Mach exception handling model whenever possible.

467318_NineInchRule.eps ¬

SunSoft_on-line_back_address.tiff ¬

895143_NeXT_on-line_address.tiff ¬