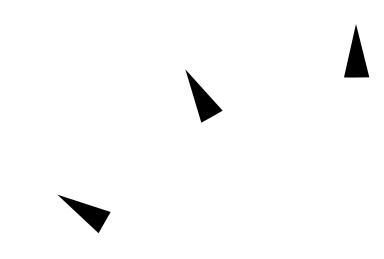
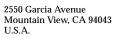
# Porting NEXTSTEP 3.2/3.3 Applications to OpenStep on Solaris











© 1996 Sun Microsystems, Inc.

2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A. All rights reserved.

Portions Copyright 1995 NeXT Computer, Inc. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Portions of this product may be derived from the UNIX® system, licensed from UNIX System Laboratories, Inc., a wholly owned subsidiary of Novell, Inc., and from the Berkeley 4.3 BSD system, licensed from the University of California. Third-party font software, including font technology in this product, is protected by copyright and licensed from Sun's suppliers. This product incorporates technology licensed from Object Design, Inc.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and FAR 52.227-19.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

#### **TRADEMARKS**

Sun, Sun Microsystems, the Sun logo, SunSoft, the SunSoft logo, Solaris, SunOS, and OpenWindows are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. OPEN LOOK is a registered trademark of Novell, Inc. PostScript and Display PostScript are trademarks of Adobe Systems, Inc. Object Design is a trademark and the Object Design logo is a registered trademark of Object Design, Inc. OpenStep, NeXT, the NeXT logo, NEXTSTEP, the NEXTSTEP logo, Application Kit, Foundation Kit, Project Builder, and Workspace Manager are trademarks of NeXT Computer, Inc. Unicode is a trademark of Unicode, Inc. VT100 is a trademark of Digital Equipment Corporation. All other product names mentioned herein are the trademarks of their respective owners.

All SPARC trademarks are trademarks or registered trademarks of SPARC International, Inc. in the United States and other countries. SPARCcenter, SPARCcluster, SPARCCompiler, SPARCdesign, SPARC811, SPARCengine, SPARCprinter, SPARCstation, SPARCstorage, SPARCworks, microSPARC-11, and UltraSPARC are licensed exclusively to Sun Microsystems, Inc. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK® and SunTM Graphical User Interfaces were developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

X Window System is a product of X Consortium, Inc.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

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





# Table of Contents

Pref	ace	vii
1.	Introduction to OpenStep	1
	Components of OpenStep	1
	OpenStep and CORBA	2
2.	Managing Interface Differences	5
	Interface Changes in OpenStep	6
	Application Kit	6
	Foundation Kit	7
	Display PostScript Kit	7
	Distributed Objects System	7
	Objective C Run-Time System	8
	Unsupported NEXTSTEP Interfaces	8
3.	Resolving Operating System Differences	9
	Supporting Virtual Memory	9
	Accommodating Tasks	11

	Managing Threads	12
	Providing Interprocess Communication	15
	Handling Exceptions	17
4.	<b>Handling Environment, Compiler, and Language Differences</b>	19
	Environment Changes	19
	Compiler Differences	20
	Invoking the C++ Compiler	20
	Specifying Include Files and Libraries	20
	Handling Preprocessor Variables	21
	Language Variants	22
	Adjusting for Type Checking	22
	Handling Use of Reserved Words	23
	Creating Prototypes for C++ Functions	24
	Managing Syntax Differences	24
	Handling Structures Defined Inside a Cast	25
	Replacing Unsupported GNU Extension	25
	Accommodating Dynamically Allocated Arrays	26
<b>5</b> .	Supporting Internationalization of Your Application	27
	Specifying a Locale	27
	Creating Locale Directories	28
	Displaying Internationalized Messages	28
6.	Performing the Porting Process	31
	Translating Your Project and Make Files	31
	Converting Mach-Specific Features to Solaris	39

	Converting Objective C to Objective C++	32
	Converting the NEXTSTEP Interface to OpenStep	32
	Working with Conversion Tools	35
	The cvtCC Conversion Tool	35
	The cvtmake Conversion Tool	37
	The cytstand Conversion Tool	37
	The cvtextract Conversion Tool	38
	The cvtquery Conversion Tool	38
	The cytre Conversion Tool	38
	Using the Emacs Editor	39
7.	Porting a Simple Application	43
	Converting Project Files	43
	Converting Header Files	44
	Converting Source Files	47
	Compiling Your Project	52
	Ported Source Code	53

Table of Contents

# Preface

Porting NEXTSTEP 3.2/3.3 Applications to OpenStep on Solaris describes porting-related information, conversion processes, and tools you need to port your NEXTSTEP $^{\text{\tiny M}}$  Release 3.2 and Release 3.3 applications to OpenStep $^{\text{\tiny M}}$  on the Solaris $^{\text{\tiny M}}$  operating system.

The reference information in this white paper complements information presented in the SunSoft<sup>™</sup> training course, "Porting NEXTSTEP Applications." For course registration information, contact SunSoft at (800) SUNSOFT to locate an Authorized Education Center (AEC) offering this class near you.

## Who Should Use This Paper

This paper is intended for developers who are already familiar with developing NEXTSTEP applications. You can use this paper to help you understand environment differences between NEXTSTEP and OpenStep. It also provides a tutorial to help you understand and perform the porting process.

#### Before You Read This Paper

Before you begin this paper, you should already

- Be proficient in Objective C and object-oriented programming.
- Be familiar with NEXTSTEP.
- Be familiar with UNIX.
- Know how to use tools such as compilers and debuggers.

An overview of OpenStep is provided in Chapter 1.

#### How This Paper Is Organized

This paper, *Porting NEXTSTEP 3.2/3.3 Applications to OpenStep on Solaris*, discusses these topics:

**Chapter 1, "Introduction to OpenStep,"** is an overview of OpenStep—its components and relationship to CORBA (Common Object Request Broker Architecture).

**Chapter 2, "Managing Interface Differences,"** explains interface differences between NEXTSTEP and OpenStep.

**Chapter 3, "Resolving Operating System Differences,"** describes operating system differences between NEXTSTEP Release 3.2/3.3 applications on the Mach operating system and OpenStep on the Solaris operating system.

Chapter 4, "Handling Environment, Compiler, and Language Differences," discusses how to handle environment, compiler, and language differences when porting your NEXTSTEP application to OpenStep.

**Chapter 5, "Supporting Internationalization of Your Application,"** explains how to modify your NEXTSTEP application to support the features of OpenStep internationalization.

**Chapter 6, "Performing the Porting Process,"** summarizes NEXTSTEP to OpenStep porting processes and describes the OpenStep tools you use to accomplish these processes.

**Chapter 7, "Porting a Simple Application,"** illustrates the NEXTSTEP to OpenStep porting process on a simple calculator application.

#### **Related Information**

For general information on porting to Solaris, refer to the manual, *Solaris Porting Guide*. The book is written by SunSoft ISV Engineering, Michele Ann Goodman, Manoj Goyal, and Robert A. Massoudi. It is published by SunSoft Press and PTR Prentice Hall. This book is available at local bookstores, or can be ordered directly from Prentice Hall by calling (201) 592-2863. Its ISBN is 0-13-030396-8.

If you have the OpenStep product, you can find on-line troubleshooting hints related to the porting process in

/usr/openstep/Developer/ConversionTools/cvtnotes.doc.

Preface

# Introduction to OpenStep



The OpenStep application environment provides a user interface that includes desktop applications and services, and a development environment that enables you to rapidly prototype, develop, and deploy enterprise applications.

The specification for OpenStep was developed by NeXT Computer and Sun. OpenStep is based on the NEXTSTEP Release 3.2 development environment created by NeXT Computer.

This chapter describes the components and positioning of OpenStep.

#### Components of OpenStep

OpenStep is expected to meet the needs of fortune 1000 companies that require vertical market or "mission critical" distributed applications. The ability to develop custom applications enables corporations to meet specific business goals. For example, many financial services companies compete by using market data that is coupled with their own data and analysis algorithms. OpenStep enables companies to quickly develop and enhance custom applications to remain competitive in a fast-moving industry.

On the desktop, OpenStep provides end-users with a collection of productivity applications such as electronic mail, a word processor, and a terminal emulator. These applications maintain a common appearance and behavior—they also interoperate and coexist with other desktop applications created with CDE (Common Desktop Environment) and SunSoft's OpenWindows $^{\text{\tiny M}}$ .

OpenStep supports an application programming interface (API) and several application services such as spell checking and printing that make it easier for you to design full-featured applications. OpenStep interfaces are primarily an enhanced superset of NEXTSTEP interfaces, although a few NEXTSTEP interfaces are no longer supported.

In addition, OpenStep provides you with an environment for rapid application development that includes an interface and project builder, a compilation suite, team-oriented source code control, and a visual debugger.

#### OpenStep and CORBA

OpenStep is designed to complement OMG (Object Management Group) CORBA (Common Object Request Broker Architecture) technologies and database information systems. Together, these components comprise a three-tier client server architecture. Figure 1-1 shows the relationship of OpenStep to these components.

Database

ORB

CORBA
Business Object

OpenStep

Other environment

Figure 1-1 Position of OpenStep in the Three-Tier Architecture

The top tier consists of a data repository (the model data), the second tier supports your business logic (a controller for the data), and the third tier is your application presentation (a view of the data).

For example, the CORBA layer handles the workflow of the business process—it contains business service objects that can respond to queries such as a request for the number of products purchased by a specific region's customers. A CORBA business object can access several local or remote databases to answer a query. Multiple application environments, including OpenStep, can utilize the information provided by a business object.

The rest of this paper describes how to begin using the features of the OpenStep application environment by porting your existing NEXTSTEP 3.2/3.3 applications to OpenStep.



# Managing Interface Differences



The OpenStep programming interface is derived from NEXTSTEP Release 3.2/3.3. From a user's point of view, OpenStep is virtually identical to NEXTSTEP—from a developer's point of view, the programming environments are similar, but contain notable differences.

Differences between OpenStep and NEXTSTEP can be generally grouped into three categories:

- NEXTSTEP interfaces that are modified for OpenStep
- NEXTSTEP interfaces that are not available in OpenStep
- OpenStep interfaces that have been created to provide additional capabilities and enhancements

Information on OpenStep porting tools that you can use to convert NEXTSTEP source code to OpenStep programming interfaces is described in Chapter 6, "Performing the Porting Process" in this paper. Detailed OpenStep interface information is provided in the manual, *OpenStep Programming Reference*. Information on development tools, such as Interface Builder and Project Builder, for creating OpenStep interfaces is described in the manual, *OpenStep Development Tools*.



# Interface Changes in OpenStep

OpenStep and NEXTSTEP Release 3.2/3.3 have several interfaces in common. Most modifications made to OpenStep interfaces are intended as enhancements to the original functionality provided in NEXTSTEP. For example, OpenStep interfaces are operating-system independent.

Where appropriate, interface components such as Mach-specific run-time functions are no longer supported. When necessary, naming conventions for OpenStep methods are modified to ensure uniformity.

Table 2-1 summarizes modifications in the OpenStep application programming interface (API) from its original component in NEXTSTEP Release 3.2/3.3. Additional information is provided throughout the rest of this chapter.

Table 2-1 Changes Between NEXTSTEP and OpenStep Interface Components

NEXTSTEP Release 3.2/3.3 Component	OpenStep Component
Application Kit	Minor API refinements and the removal of obsolete classes.
Foundation Kit	OpenStep Foundation Kit is an enhanced superset of the NEXTSTEP 3.3 Foundation Kit. The OpenStep Kit contains API modifications and additions.
Display PostScript Kit	As in NEXTSTEP 3.2, the OpenStep Display PostScript kit extends the display PostScript language by adding operators for compositing and transparency.
Distributed Objects System	Minor modifications and API refinement.
Objective C Run-Time System	Removal of Mach-specific run-time functions.

## Application Kit

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

- 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 interapplication communication including cut and paste, drag and drop, interapplication services, and object-linking capabilities

The OpenStep Application Kit is similar to the NEXTSTEP Application Kit except for minor API refinements and the removal of obsolete classes, such as Speaker, Listener, and NXJournaler.

#### Foundation Kit

The Foundation Kit provides classes to help make software development easier. The objects it defines are used for accessing structured and stored data, thread control, automatic deallocation, Unicode strings, object distribution, and the program environment. Many of the data access and storage classes available in NEXTSTEP 3.2 have equivalents in OpenStep's Foundation Kit.

NEXTSTEP 3.3 supports a preliminary version of a Foundation Kit; release 3.2 does not support a Foundation Kit. The OpenStep Foundation Kit is an enhanced superset of the NEXTSTEP Foundation Kit.

#### Display PostScript Kit

Adobe's Display PostScript client library extends the PostScript language and adapts it to accommodate the interactive requirements of the screen.

In its Display PostScript Kit, NEXTSTEP extends the Display PostScript language by adding operators for window management, compositing, and transparency. Similarly, OpenStep extends the client library with operators for compositing and transparency. However, it does *not* include support for window management operators—this functionality is supported by OpenStep's Window Manager.

#### Distributed Objects System

The Distributed Objects System enables interapplication communication using the same Objective C messaging that is used by an individual application. This messaging system can also be used between remote applications on a network.



The OpenStep Distributed Objects System is similar to the NEXTSTEP Distributed Objects system except for minor API refinements. You should note that the OpenStep version of the system is wire compatible with the NeXT version.

#### Objective C Run-Time System

The Objective C Run-Time System provides operating system functionality for the Objective C language. These run-time functions implement the language's messaging system and allow dynamic loading and archiving of objects.

The OpenStep Run-Time System is identical to the NEXTSTEP version of the system excluding the Mach-specific run-time functions which are unsupported.

## Unsupported NEXTSTEP Interfaces

OpenStep does not support all of the software kits and libraries that are supported in the NEXTSTEP 3.2 and 3.3 releases. You should avoid using these additional libraries to ensure that your application is OpenStep compliant and portable across future versions of the OpenStep product.

The software kits and libraries that are unsupported in OpenStep include:

- Indexing Kit
- DB Kit
- Mach Kit
- MIDI Driver API
- NetInfo Kit
- Novell NetWare networking
- Sound Kit
- 3D Graphics Kit

**Note** – OpenStep does provide support for the DBTableView class that was originally part of the NEXTSTEP DB Kit as well as a subset of the NEXTSTEP Sound Kit. Refer to the *OpenStep Programming Reference* for detailed interface information.

# Resolving Operating System Differences

**3** 

To port your application from NEXTSTEP to OpenStep, you need to resolve differences in operating systems for which these environments are designed. Existing NEXTSTEP Release 3.2/3.3 applications run on the Mach operating system which is compatible with UNIX 4.3 BSD $^{\text{\tiny M}}$ . OpenStep runs on the Solaris operating system which is an SVR4 UNIX $^{\text{\tiny M}}$  implementation.

For information on converting 4.3 BSD compatible programming interfaces in Mach to SVR4 UNIX interfaces in Solaris, see the manual, *Solaris Porting Guide*.

This chapter describes how to convert pure Mach operating system and library calls to Solaris. Specifically, it identifies important differences between the Mach and Solaris operating systems and provides recommendations for handling them.

**Note** – Operating system porting issues do not apply if your NEXTSTEP application does not contain any Mach operating system functions.

#### Supporting Virtual Memory

Mach operating system virtual memory facilities provide applications with direct access to pages or objects of virtual memory. An application can allocate, deallocate, and protect these pages. Unlike memory allocated with the UNIX malloc function, Mach enables allocation to occur anywhere in the address space of a Mach task. (A task is roughly equivalent to a UNIX process.)

Solaris provides applications with access to virtual memory objects with the mmap(2) system function. This function establishes a mapping between the address space of a process and a virtual memory object.

Both Mach and Solaris allow applications to specify access to virtual memory, to indicate if it is to be shared, and to define how pages are protected. Your application is easier to port to OpenStep if you've adhered to a basic implementation of virtual memory allocation, deallocation, and protection.

Mach provides a map\_fd function that maps a file to virtual memory. To achieve similar functionality in Solaris, you can use the mmap(2) function. The most portable way to provide this functionality is through the NSPosixFileDescriptor class that is part of the OpenStep Foundation Kit.

For detailed information on the NSPosixFileDescriptor class and the OpenStep Foundation Kit, see the manual, *OpenStep Programming Reference*.

The most efficient way to make your application's memory usage portable is to use the OpenStep Foundation Kit's zone-based interfaces. Specifically, you should use the NSZone functions which provide interfaces for memory allocation, deallocation, and duplication. The zones returned by these functions are understood and used by other OpenStep interfaces.

Table 3-1 lists the Mach virtual memory functions and their corresponding functions in Solaris.

Table 3-1 Mach Virtual Memory Functions and Solaris Equivalents

<b>Mach Function</b>	Description	<b>Solaris Function</b>
vm_allocate()	Allocates virtual memory	mmap(2)
<pre>vm_deallocate()</pre>	Deallocates virtual memory	munmap(2)
<pre>vm_protect()</pre>	Specifies protection on a range of virtual memory pages	<pre>mmap(2), mprotect(2)</pre>
<pre>vm_inherit()</pre>	Specifies inheritance characteristics of virtual memory	Not applicable
vm_read()	Reads virtual memory of a specified task	proc(4)
<pre>vm_write()</pre>	Writes virtual memory of a specified task	proc(4)
vm_copy()	Copies virtual memory of a specified task	Not applicable

Table 3-1 Mach Virtual Memory Functions and Solaris Equivalents (Continued)

Mach Function	Description	<b>Solaris Function</b>
vm_region()	Returns description of characteristics of a range of virtual memory	proc(4)
vm_set_policy()	Specifies paging policy for a region of virtual memory	madvise(3)
vm_statistics()	Examines virtual memory statistics	Not applicable



Caution – Using the Mach vm\_protect function for memory protections or using the Mach vm\_set\_policy function to affect the kernel's treatment of memory may make porting your application to OpenStep difficult.

## Accommodating Tasks

In the Mach operating system, an executing program consists of at least one task and some number of threads. This program maintains 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. In Solaris, a UNIX process contains resources and executes using a lightweight process (LWP).

Most Mach task functions have no direct equivalents in Solaris. When porting your application to OpenStep, you should use Solaris task-related mechanisms. The Mach task functions you should avoid when porting to OpenStep include:

- 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()

#### Managing Threads

The Mach operating system provides two programmatic interfaces to its threads—a lower-level interface to Mach kernel threads and a higher-level interface called C-threads. The C-threads interface is an easily ported user-level library. The Solaris threads library provides sophisticated constructs to manipulate threads that closely correspond to the C-threads library.

The most efficient way to provide your applications with thread and synchronization abstractions is to use the interface for the OpenStep Foundation Kit. Specifically, the NSThread and NSLock classes provide interfaces to the Solaris threading facilities. The NSThread object creates a new thread of control within your application, and the NSLock object performs standard synchronization primitives necessary for running multithreaded applications.

The low-level Mach thread functions you should avoid when porting to OpenStep include:

Porting NEXTSTEP 3.2/3.3 Applications to OpenStep on Solaris—February 1996

```
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_set_exception_port()
thread_set_exception_port()
thread_set_exception_port()
```

- thread\_set\_state()
- thread\_info()
- thread\_policy()
- thread\_priority()
- thread\_max\_priority()
- thread\_resume()
- thread\_suspend()
- thread\_switch()
- thread\_terminate()

Note – If your application requires sophisticated threading mechanisms, you might prefer to use the Solaris  ${\tt thr\_self}$  function which corresponds to the Mach  ${\tt thread\_self}$  function.

Table 3-2 lists the Mach C-threads interfaces and the corresponding Solaris threads library functions.

Table 3-2 Mach C-Threads and Solaris Equivalents

<b>Mach C-Threads Function</b>	Description	<b>Solaris Function</b>
condition_alloc()	Allocate condition variable	Not applicable
condition_broadcast()	Broadcast a condition	cond_broadcast()
<pre>condition_clear()</pre>	Clear a condition	cond_destroy()
condition_init()	Initialize a condition	cond_init()
<pre>condition_free()</pre>	Free condition variable	Not applicable
condition_name()	Get name of a name	Not applicable
<pre>condition_set_name()</pre>	Set name of a condition	Not applicable
condition_signal()	Signal a condition	cond_signal()
condition_wait()	Wait on a condition	<pre>cond_wait() cond_timedwait()</pre>
cthread_abort()	Interrupt a thread	thr_kill()
<pre>cthread_count()</pre>	Get number of threads in this task	Not applicable
cthread_data()	Get thread-specific data	thr_getspecific()
cthread_set_data()	Set thread-specific data	<pre>thr_keycreate() thr_setspecific()</pre>



Table 3-2 Mach C-Threads and Solaris Equivalents (Continued)

Mach C-Threads Function	Description	<b>Solaris Function</b>
cthread_detach()	Detach a thread	thr_create()
cthread_errno()	Get errno for current thread	errno variable is valid for each thread
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	Not applicable
cthread_set_name()	Set the name of this thread	Not applicable
cthread_priority()	Set base priority of this thread	thr_setprio()
cthread_max_priority()	Set maximum priority of this thread	Not applicable
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 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	<pre>mutex_destroy()</pre>
mutex_free()	Free memory associated with a mutex variable	Not applicable
mutex_init()	Initialize mutex variable	<pre>mutex_init()</pre>
mutex_lock()	Lock a mutex	<pre>mutex_lock()</pre>
mutex_name()	Get the name of a mutex	Not applicable

Table 3-2 Mach C-Threads and Solaris Equivalents (Continued)

Mach C-Threads Function	Description	<b>Solaris Function</b>
mutex_set_name()	Set the name of a mutex	Not applicable
<pre>mutex_try_lock()</pre>	Try a mutex lock	<pre>mutex_trylock()</pre>
<pre>mutex_unlock()</pre>	Unlock a mutex	<pre>mutex_unlock()</pre>

In addition to these thread-related functions, Solaris provides you with functions for implementing semaphores in threads and functions for implementing multiple read/single writer locks. These functions include:

- sema\_init()
- sema\_destroy()
- sema wait()
- sema\_trywait()
- sema\_post()
- rwlock\_init()
- rwlock\_destroy()
- rw\_rdlock()
- rw\_wrlock()
- rw unlock()
- rw\_trydlock()
- rw\_trywrlock()

#### **Providing Interprocess Communication**

Mach is a message-based operating system. All communication between two tasks, or between a task and the kernel, occurs through Mach messaging. To achieve messaging, Mach uses a port as a communication channel.

The Mach messaging interface for ports represents a network transparent form of interprocess communication (IPC). You can convert Mach messaging in your NEXTSTEP application to either native Solaris IPC mechanisms or to OpenStep Distributed Object classes.

Basic IPC mechanisms in Solaris are pipes, named pipes, sockets, signals, messages, semaphores, and shared memory. For distributed applications, Solaris provides remote procedure call (RPC) facilities. Solaris currently supports, RPC, transport-independent RPC, and the ToolTalk™ service for standard messaging on the common desktop environment (CDE).



The most efficient way to provide your applications with an object-oriented, network transparent message facility is to use Distributed Objects classes in OpenStep's Distributed Objects System. Specifically, you should use the NSConnection, NSProxy, and NSDistantObject classes.

The Mach messaging interface for ports includes:

```
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()
```

# Handling Exceptions

The Mach operating system supports a message-based model for exception handling. Mach sends exceptions to a thread on an exception port. Solaris exception handling is based on SVR4 signals. When porting to OpenStep, you should use the OpenStep interface for exception handling provided by the NSException class.



# Handling Environment, Compiler, and Language Differences



This chapter describes how to handle environment, compiler, and language differences when porting your NEXTSTEP application to OpenStep. Specifically, this chapter

- Describes basic environment changes.
- Outlines primary differences between the NeXT GNU-based Objective C compiler and the Sun SPARCworks C++ compiler.
- Specifies variations between the NEXTSTEP usage of Objective C and the OpenStep usage of Objective C++.

# **Environment Changes**

To locate the OpenStep compiler, debugger, and utilities, set your standard path or path variable to include the following in your .cshrc (or equivalent) file:



# Compiler Differences

When porting your application to OpenStep, you need to handle effects that the C++ compiler has on your Objective C code. This section describes how to invoke the C++ compiler to accommodate your Objective C code and to specify include files and OpenStep libraries; it also provides a comparison of compiler preprocessor variables.

#### *Invoking the C++ Compiler*

You invoke SunSoft's SPARCworks C++ compiler as CC; as such, it is a C++ compiler. But when you invoke CC to compile a source file with a suffix of .m or .M or with the -objc flag, it accepts Objective C syntax as well as C++ syntax. For example:

```
CC -c foo.m
CC -objc bar.C
```

To link a group of .o files, it is necessary to use the -objc flag to indicate to the compiler that you are linking an Objective C program. For example:

```
CC foo.o bar.o -objc -o myprogram
```

**Note** – Although NeXT supports an Objective C++ compiler, this compiler is not widely used at this time.

#### Specifying Include Files and Libraries

To compile Objective C programs using OpenStep programming interfaces requires the following  $-\mathtt{I}$  include path names:

```
CC -I/usr/openstep/include foo.m -c -mt
    -I/usr/openwin/include -I/usr/openwin/include/X11
```

During linking, OpenStep requires the following options to create an Objective C program:

```
CC -objc -mt foo.o bar.o -o myprogram
   -L/usr/openstep/lib -R/usr/openstep/lib
   -L/usr/openwin/lib-R/usr/openwin/lib
   -lAppKit -ldps -lFoundation
```

The -L and -R and the linker (1d) options indicate to the Solaris linker and run-time loader the location of OpenStep libraries. These libraries include: -lAppKit, -ldps, and -lFoundation.

These options are automatically inserted into Project Builder generated make files.

#### Handling Preprocessor Variables

The NeXT compiler predefines a number of preprocessor variables. Table 4-1 provides a comparison of NeXT and Sun preprocessor variables.

Table 4-1 Correspondences Between NeXT and Sun Preprocessor Variables

NeXT Preprocessor Variable	Sun Preprocessor Variable
-DOBJC	Same when compiling under Objective C
-Dsparc	Same
-Dsun	Same
-Dunix	Same
-Dsparc	Same
-Dsun	Same
-Dunix	Same
-Dsun	usessun
-Dunix	usesunix
-Dsparc	usessparc
-Dsvr4	usesSVR4
-DBIG_ENDIAN	Add to make file
-DNeXT_PDO	Not applicable
-DNeXT_PDO	Not applicable
-DNeXT_PDO	Not applicable
-DNX_COMPILER_RELEASE_3_0	Not applicable
-DNX_COMPILER_RELEASE_3_0	Not applicable
-DNX_COMPILER_RELEASE_3_0	Not applicable



Table 4-1 Correspondences Between NeXT and Sun Preprocessor Variables (Continued)

NeXT Preprocessor Variable	Sun Preprocessor Variable
-DGNUC= <some value=""></some>	Not applicable
-DGNUC_MINOR= <some value=""></some>	Not applicable
-DGCC_NEW_VARARGS	Not applicable

### Language Variants

Objective C consists of a set of object-oriented language extensions on a base language. In NEXTSTEP, the standard C language is used as the base language. In OpenStep, C++ is used as the base language.

When porting your NEXTSTEP application to OpenStep, you should be aware of the following C to C++ language-related issues:

- Stronger type conversion rules for assignments and initializations
- Use of C++ reserved words as identifiers such as class and new
- Necessity to declare prototypes for functions before they are used
- Differences between C++ and ANSI-C syntax for enumerations
- Handing structures that are declared inside a cast
- Unsupported GNU extensions such as the typeof operator and dynamically allocated arrays

The rest of this section provides additional information on language differences.

## Adjusting for Type Checking

Unlike ANSI-C, C++ changes the semantics of (void \*) such that values of type (void \*) may not be assigned to stronger typed pointers. When compiling Objective C++ code, the SPARCworks C++ compiler treats this as a warning rather than an error. Because this compiler feature may not be supported in future releases, you should insert an appropriate cast.

In addition, you also need to resolve differences between formal and actual arguments for a function call. For this, you must also insert the appropriate cast.

#### Compiler Diagnostic

on anachronisms.

```
"filename.m", line 2: Warning (Anachronism): Assigning
            void* to objc_object*.
"filename.m", line 2: Note: Type "CC -migration" for more
```

#### Sample Code

```
id objects;
objects = malloc(count*sizeof(id));  // Warning
objects = (id )malloc(count*sizeof(id));// Corrected
```

#### Compiler Diagnostic

```
"filename.m", line 4: Warning (Anachronism): Formal argument charString of type const unsigned char** in call to function(const unsigned char**, unsigned, unsigned short*) is being passed unsigned char**.
```

### Sample Code

```
unsigned char *bufptr;
void function(const unsigned char **, int);
function(&bufptr, numChars);
function((const unsigned char **)&bufptr, numChars);
```

## Handling Use of Reserved Words

Because Objective C uses C++ reserved words as identifiers, it is necessary for you to rename these variables.

#### Compiler Diagnostic

```
"filename.m", line 2: Error: class may not be a type name.
```

#### Sample Code

```
+ allocWithZone:(NSZone *)zone {
   Class class = self;//'class' is a reserved word.
   Class cls = self;//Corrected
   ....
}
```

## Creating Prototypes for C++ Functions

In C++, all functions must have prototypes before they are used.

#### Compiler Diagnostic

```
"filename.m", line 1: Error: The function access must have a prototype.
```

#### Sample Code

```
access("foo", R_OK);
```

#### Managing Syntax Differences

C++ does not allow a comma after the last member of an enumeration declaration.

#### Compiler Diagnostic

```
"filename.m", line 4: Warning (Anachronism): Identifier expected instead of "}".
```

#### Sample Code

```
typedef enum {
   isLocal = 0,
   isRemote = 1,
   isOther = 2,
   isInvalid = 3,//Trailing ','
}
```

#### Handling Structures Defined Inside a Cast

The NeXT Objective C compiler allows you to declare a structure inside a cast. When porting to OpenStep, it is necessary for you to pull the type definition outside of the cast and make it a global declaration.

#### Compiler Diagnostic

```
"filename.m", line 2: Error: Defining struct* within a cast is illegal.
```

#### Sample Code

```
//Incorrect way
#define SELFPTR(X) ((struct
{@defs(NSConcreteMutableCharacterSet);} *)X)
...
otherBits = SELFPTR(otherSet)->bits;

//Correct way
typedef struct {
    @defs(NSConcreteMutableCharacterSet);
} *NSConcreteMutableCharacterSet_p;
#define SELFPTR(X) ((NSConcreteMutableCharacterSet_p *)X)
...
otherBits = SELFPTR(otherSet)->bits;
```

# Replacing Unsupported GNU Extension

The typeof operator is a GNU extension. It is not unsupported by the SPARCworks C++ compiler. It is necessary for you to replace typeof with a real type name.

#### Compiler Diagnostic

```
"filename.m", line 1: Error: Type name expected instead of "(".
```

#### Sample Code

```
[coder encodeValueOfObjCType:@encode(typeof(int)) at:&low];
[coder encodeValueOfObjCType:@encode(int) at:&low]; //Corrected
```

#### Accommodating Dynamically Allocated Arrays

The NeXT Objective C compiler allows dynamically allocated arrays. To achieve the same effect in SPARCWorks C++, use the alloc function.

#### Compiler Diagnostic

```
"filename.m", line 260: Error: An integer constant expression is required within the array subscript operator.
```

#### Sample Code

```
//Incorrect way
void foo(int formatLen) {
  char formatChars[formatLen];
}

//Correct way
#import <alloca.h>
void foo(int formatLen) {
        char *formatChars;
        formatChars = (char *)alloc(formatLen);
}
```

# Supporting Internationalization of Your Application

**5** 

The purpose of software internationalization is to enable applications to easily convert to and operate in multiple cultures or locales. A locale is typically defined by a particular area or region, a language, and a character set. To support the OpenStep international environment, you need to modify your existing NEXTSTEP applications.

To reduce the cost of internationalization, OpenStep on the Solaris operating system separates locale specific data from source code. This allows you to accommodate several locales using a single binary which operates appropriately in multiple locales.

This chapter describes how to modify your NEXTSTEP application to support the features of OpenStep internationalization.

## Specifying a Locale

To specify a particular locale, the Solaris operating system uses the POSIX/ANSI C function setlocale. As in NEXTSTEP, OpenStep specifies a locale in response to user input on a language choice in the Preferences application. This language choice sets the NSLanguages user default.

When porting to OpenStep, you need to support the Solaris setlocale function to access Solaris international libraries. You also need to make sure that your application is internally synchronized with the NSLanguages user default.

If your application is a subclass of NSApplication, locale specification is handled by the initialization code of NSApplication. However, if your application is not a subclass, you need to call the setlocale function.

## Creating Locale Directories

NEXTSTEP specifies locale data in language-specific subdirectories as <NameOfLanguage>.lproj. For example, locale data for the French language is French.lproj.

When porting to OpenStep, you need to support Solaris language naming conventions. For example, instead of French.lproj, your directory should be fr.lproj. For detailed information on Solaris language names, refer to the manual, Solaris International Developer's Guide, Appendix A.

To support Solaris locale data, OpenStep requires you to create additional language directories at the same level as your \*.lproj directories. You specify these directories as the names of the languages you support. For example, for the French language, your directory should be fr in accordance with Solaris language naming conventions. For a sample directory structure, see the next section "Displaying Internationalized Messages."

# Displaying Internationalized Messages

To handle translated text, Solaris provides gettext functions. As in NEXTSTEP, OpenStep provides functions for displaying internationalized messages. These messages comprise user-visible text that can be translated into various languages such as help text, menu items, and errors messages. Display functions for these messages include NSLocalizedString,

NSLocalizedStringFromTable, and NSLocalizedStringFromTableInBundle.

For information on gettext functions, see the manual, *Solaris International Developer's Guide*. For detailed information on international message display functions, see the manual, *OpenStep Programming Reference*.

OpenStep subsumes Solaris gettext functionality in its implementation so it is not necessary for you to change the international message code that you port. The primary difference between OpenStep and NEXTSTEP message functions is the generation of portable object (\*.po) files and message object (\*.mo) files to replace NEXTSTEP \*.string files.

To use the OpenStep on Solaris messaging scheme, you need to:

- 1. Wrap your internationalized messages with NSLocalizedString functions.
- 2. Run the genstrings program on your source code to extract your messages and create your portable object (\*.po) files.

  In OpenStep, genstrings supports the Solaris gettext functions. (To display the contents of your files, you can run the cat program.)

```
% cat myprog.m
#include <Foundation/NSBundle.h>
main()
{
   NSString * myString1 = NSLocalizedString(@"How are you!",
                                     @"How are you comment");
   NSString * myString2= NSLocalizedStringFromTable(@"Hello
                      World!", @"MyTable", @"Hello comment");
}
% genstrings myprog.m
                            // Creates a MyTable.po and a
                                Localizable.po file.
% cat MyTable.po
# Hello comment
msgid "Hello World!"
msgstr
% cat Localizable.po
# How are you comment
msgid "How are you!"
msgstr
```

3. Edit your .po files to include your translated message.

To display the contents of your edited files, you can re-run the cat program.



4. Run the msgfmt (message format) program on your \*.po files to create message object (\*.mo) files.

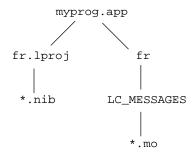
```
% cat MyTable.po
# Hello comment
msgid "Hello World!"
msgstr "<translated "Hello World!>"

% cat Localizable.po
# How are you comment
msgid "How are you!"
msgstr "<translated "How are you!>"

% msgfmt MyTable.po
% msgfmt Localizable.po
```

5. Place the \*.mo files in the <lang>/LC\_MESSAGES directory.

Sample OpenStep Directory Structure



**Note** – If the current locale is "C," which indicates the default locale in the Solaris naming conventions, gettext simply returns the message string that was passed.

# Performing the Porting Process



To convert your existing NEXTSTEP applications to OpenStep, you need to perform several conversion processes. Specifically, you need to:

- Translate your PB.project and make files into a format acceptable to OpenStep.
- Convert the Mach operating system features to Solaris.
- Convert the Objective C source language to Objective C++.
- Convert the NEXTSTEP interface to OpenStep.

This chapter describes how to accomplish these porting processes.

# Translating Your Project and Make Files

When porting to OpenStep, you need to convert your application's PB.project and make files to an OpenStep compatible format. To accomplish this, run the ProjectBuilder on the project to be converted. You should perform this on a new copy of the project.

ProjectBuilder recognizes a NEXTSTEP project and asks you to confirm that you want to convert it. Reply "yes" to allow ProjectBuilder to convert all the PB.project and make files in your project to the new format. ProjectBuilder however, does not convert Makefile.preamble and Makefile.postamble files. If your project contains any of these files, you must manually convert them according to the guidelines contained in

/usr/openstep/Developer/Makefiles/Makefile.preamble



For detailed information on the ProjectBuilder, refer to the manual, *OpenStep Development Tools*.

## Converting Mach-Specific Features to Solaris

You will get compilation error messages when porting to OpenStep if your project contains Mach operating system specific features. To resolve these errors, you need to manually convert these features to Solaris operating system equivalents.

For information on resolving Mach and Solaris operating system differences, see Chapter 3, "Resolving Operating System Differences" in this paper.

# Converting Objective C to Objective C++

You need to handle compilation errors that occur as a result of language differences between NEXTSTEP's use of Objective C and OpenStep's use of Objective C++.

The most common errors are the use of C++ keywords such as new and class and type mismatches caused by missing casts. Type mismatches occur because C++ is a more strongly typed language than C.

For information on environment, compiler, and language differences between Objective C and Objective C++, see Chapter 4, "Handling Environment, Compiler, and Language Differences" in this paper.

## Converting the NEXTSTEP Interface to OpenStep

You need to convert NEXTSTEP interface elements to OpenStep equivalents. Elements you need to address are:

- Include file names
- Macro names
- Type names such as type definition, structure, union, and enumeration names
- Global function names
- Global variable names

- Class names
- Method names
- Instance variable names

For most interface elements, there is a simple one-to-one mapping of an NX (NEXTSTEP) name to a corresponding NS (OpenStep) name. In other cases, some amount of recoding is necessary.

To assist you in interface conversion, OpenStep provides you with tools that are located in /usr/openstep/Developer/ConversionTools/bin. Put this directory name in your PATH environment variable.

These tools use a conversion database that resides in the directory /usr/openstep/Developer/ConversionTools/appkitdb. This is a normal ASCII file that contains OpenStep equivalents for NEXTSTEP interfaces. If you discover a missing conversion, you can add a conversion to a copy of appkitdb to make it available for future use. To accomplish this, read the notes contained in appkitdb.

Detailed information on conversion tools is provided in "Working with Conversion Tools" in this chapter. For information on NEXTSTEP and OpenStep interface differences, see Chapter 2, "Managing Interface Differences" in this paper.

In addition to the primary interface elements, you may need to convert NeXT Interface Builder (nib) files. Any nib files from releases prior to NEXTSTEP 3.x are unsupported by the OpenStep Interface Builder so you need to convert these nib files to NEXTSTEP 3.x or 4.0 format. However, nib files from NEXTSTEP 3.x are compatible with Sun's Openstep Application Kit and Interface Builder.

Note – NEXTSTEP 3.3 and 4.0 releases use the system font Helvetica12—OpenStep uses the system font LucidaSans12. As a result, your 3.3 and 4.0 nib files that contain buttons or other views may have clipped labels in OpenStep. To correct this, you can use Interface Builder to adjust the labels on your objects.

OpenStep's Application Kit maps NEXTSTEP 3.x classes to appropriate 4.0 classes when it loads them. When the OpenStep Interface Builder loads a 3.x nib file, it prompts the user to save the nib under a different name to allow the user to prevent conversion to a 4.0 format.

Nib files from NeXT's prerelease 4.0 product may or may not be compatible with the OpenStep Application Kit and Interface Builder. Prior to NeXT's P2, nib files created in NeXT's 4.0 Interface Builder do not load in the OpenStep Interface Builder. Other isolated incompatibilities may also occur in later prereleases.

Nib files you created using NEXTSTEP 4.0 that contain instances of classes defined in OpenStep, are compatible with the OpenStep Interface Builder and Application Kit. In addition, the OpenStep Application Kit and Interface Builder support the NSTableView class and its associated classes, NSFormatter and its subclasses, and the SoundView, SoundMeter, and Sound classes. There is also partial support for the NSImageView class.

The builtin classes offered on the nib file window Classes Tab in the OpenStep Interface Builder are part of the Openstep specification or are compatibly exchangeable with NEXTSTEP 4.0. You should not encounter any nib incompatibilities if you confine creation of additional objects in OpenStep's Interface Builder to

- the Openstep objects in the Classes tab.
- objects of classes you define in Interface Builder and supply code for.
- objects of headers that you parse in and supply code from libraries for.

Your code and libraries are available on NEXTSTEP for those classes you define or parse in the OpenStep Interface Builder.

Use of the IDL extension to create CORBA objects in the OpenStep Interface Builder results in incompatible objects being added to nib files. If you are concerned about compatibility, you should avoid the IDL extension and the use of sound in your OpenStep Interface Builder interfaces.

If you want to use NSImageView objects, you should drag an NSCustomView from the Views palette in Interface Builder and use the attributes inspector to customize it to NSImageView. The NSImageView objects in nib files that you create in NEXTSTEP 4.0 appear in the OpenStep Interface Builder. However, you can only resize and reposition these objects—there is no inspector available to change their attributes.

Openstep implements the NSText object as the NSCStringText subclass. In NEXTSTEP 4.0, the NSTextView subclass of NSText implements the NSText object. For simple uses of text objects, you should program to the NSText superclass interface which functions the same on both platforms. For more

sophisticated uses of text, you can program to the appropriate subclass interface by using a runtime check to determine which class implements the text object.

## Working with Conversion Tools

OpenStep provides you with several tools to assist you in the conversion process. These tools perform many simple conversions such as changing a NEXTSTEP construct to its OpenStep equivalent.

The conversion tools don't perform complicated conversions. For example, after a tool completes its conversion process, you may still have several remaining compilation errors. The remaining conversions can be performed at a rate of approximately 1,000 to 3,000 lines per day. For example, a 10,000 line application may take approximately three to ten days to produce a clean compile.

OpenStep conversion tools include:

- cvtCC, which is an error correcting compiler
- cvtmake, which performs a standard make using cvtCC to compile .m files
- cvtstand, which performs conversions on NEXTSTEP usages that do not cause compiler errors
- cvtextract, which extracts messages from cvtCC and cvtstand log files
- cvtquery, which is an interactive program for querying the conversion database
- cvtre, which is a file that performs sed-like regular expression replacements

The conversion tools use the appkitdb database by default. You can override this by using the -db option. You can also specify one or more auxiliary databases using +db options.

## The cvtCC Conversion Tool

The <code>cvtCC</code> conversion tool is essentially an error-correcting compiler. You call it with the same options that you specify for <code>CC</code>, except that you can only specify the name of a single file instead of a list of files to compile. The <code>cvtCC</code> conversion tool takes these options:

cvtCC [-db database] [+db aux-database] [-v] [-CC ccName] CC-options

The cvtCC compiles a file using compiler CC which it acquires from your PATH environment variable. You can use the -CC option to override this compiler.

After compilation is complete, <code>cvtCC</code> examines the error listing for include files that cannot be opened. For each file, <code>cvtCC</code> looks up the include file name in the conversion database. If located, <code>cvtCC</code> converts the source code to use the <code>NS</code> equivalent <code>include</code> file. It then repeats compilation and reexamines the error listing. If, during compilation, there are remaining <code>include</code> file discrepancies, <code>cvtCC</code> prints a message to standard out and quits.

When a compilation contains no missing include files, <code>cvtCC</code> examines the error listing and, using the conversion database, converts all NX names that the compiler targets to NS names. It recompiles the file and examines the error listing until all conversions are complete.

If cvtCC needs to fix a read-only file, it attempts to edit it using sccs. If this fails, the file is not modified. Instead, a modified version of the file is produced in a file with the same name and a .new suffix.

The <code>cvtCC</code> conversion tool produces output on standard out that contains the conversion it used to fix each error message and the conversion(s) that apply to each error message that it could not fix.

The <code>cvtCC</code> can correct errors that involve a reference to an unknown name that is actually an old <code>NX</code> name. This type of error occurs because the compiler uses new <code>NS</code> include files. Old <code>NX</code> names aren't defined in these files. The <code>cvtCC</code> cannot correct any syntax errors or type mismatch errors.

When cvtCC identifies a naming problem, it might be unable to resolve it due to these circumstances:

- The database doesn't specify a corresponding NS name. This might occur because the database is incomplete, or because the conversion for the NX name only contains a comment that indicates some manual conversion is necessary.
- The database contains multiple conversions of the NX name and cvtCC isn't able to determine which to use.

• The database contains a unique conversion of the NX name but cvtCC is unable to apply it. This typically happens with method names in which the NS method name contains a different number of formal parameters than the NX method name.

## The cytmake Conversion Tool

The cvtmake conversion tool is a script that performs a standard make but uses cvtCC to compile .m files. It accomplishes this by:

```
setenv CCC=/user/openstep/Developer/ConversionTools/bin/cvtCC
setenv OBJCC=/usr/openstep/Developer/ConversionTools/bin/cvtCC
make -e
```

The cytmake conversion tool takes these options:

```
cvtmake [-db dbname] [+db aux-database] [-CC CCname] make-options
```

## The cytstand Conversion Tool

The cvtCC conversion tool only performs conversions on NX names that the compiler has flagged with errors and warning messages. The cvtstand conversion tool converts NX names that do not cause compiler error and warning messages. Currently, cvtstand converts:

- Class names in @class declarations
- Function names declared in extern function declarations
- Method names in @selector statements

The cytstand conversion tool takes these options:

```
cvtstand [-db database] [+db aux-database] [-Idirname ...] fn1,
    fn2, ...
```

The cvtstand conversion tool does not convert method declarations although it does output notes regarding manual conversions that you need to perform. This tool can output more succinct messages about method conversions if it knows the inheritance of the class to which a method belongs. It is aware of the inheritance of all the Application Kit and Foundation Kit classes.

To locate the inheritance of user classes, cvtstand looks at the @interface declarations in all the .h files in the current directory. If your application uses other .h files, then you can name their directories with -I options so that cvtstand can locate them. For example, if all your .h files are in

gus/include, you should run cvtstand \*.h in that directory location. If all your .m files are in gus/src, you should run cvtstand -I.../include \*.m in that directory location.

The cvtstand conversion tool handles read-only files the same way as that cvtCC does. The cvtstand produces messages that describe the conversions that could and could not be performed on stdout.

## The cytextract Conversion Tool

You can use the cytextract conversion tool to extract messages from the log files produced by cytcc and cytstand. The cytcc and cytstand log files contain messages that describe the conversions that could and could not be performed on standard out.

The  ${\tt cvtextract}$  conversion tool reads the log file from standard in and writes to standard out.

If you prefer, you can simply view cvtCC and cvtstand messages in standard out.

# The cvtquery Conversion Tool

The cvtquery conversion tool is an interactive program that you can use to query the conversion database. You enter an NX name and, if applicable, it responds with its corresponding NS name.

The cytquery conversion tool takes these options:

```
cvtquery [-v] [-dump] [-help] [-db fileName] [+db aux-database]
```

By default, cvtquery displays conversions in an abbreviated form. Pressing CR without entering a name repeats the previous query in verbose mode which provides more detailed information.

## The cytre Conversion Tool

The cytre conversion tool performs sed-like regular expression replacements. It performs these replacements on a list of file name options:

```
cvtre [-dryrun] [-i] -redb regexps-file fn1, fn2, ...
```

You can use the cytre to perform conversions that cannot be performed by cytcc. For example, you must convert the message:

```
[ gus getCellSize: &fred];
to:
fred = [ gus getCellSize];
```

Several conversions are located for you in the regexp file in /usr/openstep/Developer/ConversionTools/redb.

Alternatively, if you are an emacs user, you can use the emacs functions defined in

/usr/openstep/Developer/ConversionTools/emacs/openstep-port.el

Although the cvtre cannot perform conversions that span line boundaries, emacs functions do. However, you can run cvtre on a list of files and, if necessary, it automatically performs sccs edits on these files. Using emacs, it is necessary for you to manually invoke functions on one file at a time.

## Using the Emacs Editor

The OpenStep conversion tools produce messages in a format that is compatible with the emacs compilation facility. For example, you can run cvtstand by specifying a cvtstand command to M-x compile. This places the messages from cvtstand into the compilation buffer which allows for easy browsing.

When using xemacs, you can also colorize the error logs produced by CC and the conversion tools. To accomplish this, add this to your .emacs file:

```
load-file "/usr/openstep/Developer/ConversionTools/emacs/cvt-font-
lock.el"
```

If you also use your .emacs file for emacs18, you should add this line to an emacs19 only section. This automatically turns on colorization in the compilation mode buffer. It also makes the same colorization available in shell buffers.

You explicitly enable colorization to appear by using M-x font-lock-mode or by selecting the Options.Syntax Highlighting.Colors menu item.

You can also use the compilation mode error browsing capabilities in shell buffers and regular files. For example, you can use a shell command and save output in a log file. To accomplish this line, add this to your <code>.emacs</code> file:

```
autoload 'compilation-mode "compile" "error browsing" t
```

Next, use M-x compilation-mode with the cursor in the desired buffer.

**Note** – Due to an implementation error, the error browser ignores the first two lines in the buffer, so be sure to insert two blank lines at the beginning of the buffer. In addition, once you switch a shell buffer to compilation mode, you can no longer enter shell commands in it. To reinstate it as a shell buffer, use M-x shell-mode.

If you don't like the default colors that appear with colorization, you can change them by:

- Redoing the assignment of regular expressions to different font-lock faces. See compilation-font-lock-keywords in the <code>cvt-font-lock.el</code> file.
- Assigning different colors to the font-lock faces themselves. For example, see the function use-my-colors in <code>cvt-font-lock.el</code>. These colors could be enabled by adding to your .emacs file: add-hook 'font-lock-mode-hook' use-my-colors. This changes the colors used in syntax highlighting in all the buffers, not just in the compilation error logs.

#### Emacs users can also use the functions in:

/usr/openstep/Developer/ConversionTools/emacs/openstep-port.el

to perform many regexp type conversions that are unsupported by the OpenStep conversion tools.



# Porting a Simple Application



This chapter provides detailed instructions on porting a simple calculator application from NEXTSTEP 3.3 to OpenStep using the porting tools available with the OpenStep product.

This lab requires that you have access to a machine running NEXTSTEP with the developer examples. Before you begin, you will need to copy the folder /NextDeveloper/Examples/Appkit/CalculatorLab from a NEXTSTEP machine to your machine running OpenStep on Solaris.

# Converting Project Files

When porting to OpenStep, you need to convert your application's PB.project and make files to an OpenStep compatible format.

- 1. Convert the PB.project and make files.
  - Before you attempt to compile this project, you need to convert your project and make files. When you open the project for the first time, ProjectBuilder performs this conversion for you.
  - a. Launch ProjectBuilder.
  - b. Open the CalculatorLab PB.project file.

    ProjectBuilder displays a panel that asks if you would like to save backup copies of the old PB.project and make files. You should do this especially if you are converting a large project file.



## 2. Remove the MinusPanel class from the project.

The MinusPanel class is no longer needed due to modifications in OpenStep over the NEXTSTEP product. Prior to porting any application, you should remove any classes that you know are obsolete.

#### 3. Save the project.

#### 4. Convert the nib file.

It is not necessary to convert your NEXTSTEP nib files because they are supported in OpenStep. However, performing this process ensures a cleaner conversion.

- a. Delete the StringTable object from the nib file.
  - NXStringTable has no equivalent in the OpenStep classes. Localized strings are now created with calls to the
  - NSLocalizedStringFromTable function. Since this application did not make use of internationalization, you can simply replace calls to the NXStringTable with the strings they are requesting. For information on supporting internationalization, see Chapter 5, "Supporting Internationalization of Your Application" in this paper.
- b. Change the class of the panel from MinusPanel to NSPanel. Since you deleted the MinusPanel class from the project, you need to make sure the nib file doesn't reference it.
- c. Save these changes to the nib file.

# Converting Header Files

The first part of the source code conversion process is converting the interfaces of the objects in your project. You use the cvtstand conversion tool to locate discrepancies between your code and the OpenStep interfaces. This tool does not actually fix errors for you. When cvtstand finds discrepancies, you need to make the changes to both your .h and .m files. This is a good time to make changes to the actual source code if the parameter or return types for the method have changed.

#### 1. Launch the xemacs editor.

While using the conversion tools, you can use xemacs as your editor. This editor has an interface to the OpenStep conversion tools. When the conversion tools locate complex conversions that they cannot perform automatically, you can use xemacs to help you to locate the lines of code that require a manual change.

- a. Launch a shell window.
- b. Type xemacs & at the prompt.
  This editor launches with an empty buffer.

#### 2. Run the cytstand conversion tool.

- a. Click the Compile button in the xemacs tool bar. This editor brings up a panel that asks if you want to edit the command it uses to compile.
- b. Click on the Edit command button on the panel
- c. Type cvtstand \*.h in the minibuffer at the bottom of the window.
- d. Press the return key.

This editor splits the window into two sections—one window displays compilation errors and the other allows you to work on source code. You should see something like:

```
*** Fix the following method declarations/@selectors by hand
"SimpleCalc.h", line 83: - appDidInit:sender;
Method belongs to class: SimpleCalc:NSObject
-- from> file=appkit/Application.h, source= '- appDidInit:sender;'
-- to> file=NSApplication.h, source= '-
(void)applicationDidFinishLaunching:(NSNotification *)notification;

"SimpleCalc.h", line 88: - windowWillClose:sender;
Method belongs to class: SimpleCalc:NSObject
-- *** Method names are the same; check param/return types
-- from> file=appkit/Window.h, source= '- windowWillClose:sender;'
-- to> file=AppKit/NSWindow.h, source= '-
(void)windowWillClose:(NSNotification *)notification;'
```



#### 3. Fix the errors that cytstand locates.

The cvtstand conversion tool finds two necessary conversions for the delegate methods that are used by the SimpleCalc class. All delegate methods in OpenStep work using the notification mechanism. The delegate automatically receives all of the notification methods that it implements.

a. Convert the appDidInit method to the

applicationDidFinishLaunching method.

The applicationDidFinishLaunching method takes an NSNotification object as the parameter and returns void.

- i. Edit SimpleCalc.h to change the method name, parameter, and return type.
- ii. Edit SimpleCalc.m to reflect the changes you made in SimpleCalc.h.
- iii. Change the applicationDidFinishLaunching method so that it does not return a value.
- b. Convert the windowWillClose method's parameter and return types.
  - i. Change the parameter and return value types.
  - ii. Change the source code to fix a looping bug.

The windowWillClose method now compiles correctly, but the current version causes an infinite loop when the user either closes the window or quits the application. Closing the window calls the NSApplication terminate method, which attempts to close the window again which calls the terminate method, and so on. You can fix this bug by creating a static variable inside the method body to track whether the terminate method has been called and to prevent calling this method after it is first invoked. Your code should appear similar to this:

```
- (void)windowWillClose:(NSNotification *)notification
{
   static int closing = 0;
   if (!closing)
   {
      closing = 1;
      [NSApp terminate:self];
   }
}
```

4. Save the changes you made to SimpleCalc.h and SimpleCalc.m.

## Converting Source Files

Now that you've completed the major changes to the class interfaces, you can begin to convert the source code. The cvtmake conversion tool assists you with this process. It performs simple conversions on the source code and flags the more complex changes that you need to make by hand. Using cvtmake is an iterative process. After you run cvtmake, you fix the errors it is unable to correct, and then you run cvtmake again to locate more errors. Since cvtmake makes changes to the source files, it is that you important reload files you have open after cvtmake runs.

Note – The OpenStep conversion tools produce messages in a format that is compatible with the emacs compilation facility. Although this tutorial describes how to use emacs to simplify the conversion process, you can perform these changes manually. For example, when this tutorial instructs you to "Click the Compile button in the xemacs tool bar to run cvtmake," you can simply run cvtmake, make the necessary changes by hand, and re-run cvtmake until you have no remaining errors.

- 1. Run the cytmake conversion tool.
  - a. Click the Compile button in the xemacs tool bar. This editor brings up a panel that asks you if you want to edit the command that it uses to compile.
  - b. Click on the Edit command button on the panel.

- c. Type cvtmake in the minibuffer at the bottom of the window.
- d. Press the return key.

The compilation buffer displays the results of performing the cvtmake. It displays the conversions that cvtmake was able to make, followed by the conversions you must make by hand. For the parts you need to fix, you should see something like:

2. Reload the SimpleCalc.m by selecting Reload from the xemacs File menu.

You do this to see the changes performed by cvtmake.



Warning – If you do not reload your files after you run cvtmake, the file you are working on will be out of synchronization with the file that cvtmake modified on your storage device. This makes it necessary for you to reload the source file each time you are finished running cvtmake.

- 3. Fix the error messages by deleting the imports for
  - <sys\_s/NXStringTable.h> and <appkit/publicWraps.h>.
    You can safely delete the imported NXStringTable class because it does
    not exist under OpenStep. The publicWraps.h file is imported for the
    NXBeep function. Under OpenStep, NSBeep is part of the Application Kit.
- 4. Save SimpleCalc.m.
- 5. Click the Compile button on the xemacs tool bar to run cvtmake again.

6. Convert the name of the SimpleCalc operator instance variable.

OpenStep uses an Objective C++ compiler and as such is sensitive to C++ keywords. The operator keyword makes the compiler expect that you are creating an overloaded operator. You can fix this error by changing the variable's name. To accomplish this, you need to change the name of the variable in the header file and in SimpleCalc.m.

**Note** – You can perform a search-and-replace operation in emacs by pressing the Escape key and then the "%" key. Emacs prompts you for the string to search for and the string to replace it with.

- 7. Save SimpleCalc.h and SimpleCalc.m.
- 8. Click the Compile button on the xemacs tool bar to run cvtmake again.
- 9. Fix the error:

```
"SimpleCalc.h", line 83: Error: Type name expected instead of "NSNotification".
```

This application was written before NeXT supported precompiled headers so it imports individual headers for each class it needs. You can fix this error by importing <appKit/AppKit.h> in SimpleCalc.h. You can then delete the individual import lines for class header files in SimpleCalc.m.

10. Click the Compile button on the xemacs tool bar to run cytmake again.

#### 11. Fix the error:

```
"SimpleCalc.m", line 51: Warning (Anachronism): Formal argument astring of type NSString* in call to NSActionCell::- [setStringValue:](NSString*) is being passed char*.
```

In this error, a constant character pointer (const char \*) is being passed to the setStringValue method of an NSTextField. You can fix this by using the precompiler shortcut to create an NSString by replacing "" with @"". This creates an autoreleased NSString object that contains the text between the quotes—in this case, a string with no characters.

#### 12. Fix the error:

"SimpleCalc.m", line 70: Warning: Cannot find instance method valueForStringKey: for class id.

Since this application does not use the internationalization features of the NXStringTable, you can safely substitute calls to it with NSStrings that contain the correct text. In this case, replace [NXStringTable valueForKey: "decimalPointString"] with @".". You need to convert this method, since you are now passing an NSString to the appendToString method. For information on supporting internationalization, see Chapter 5, "Supporting Internationalization of Your Application" in this paper.

- 13. Convert the appendToString: method's parameter type from const char
   \* to NSString \*.
  - a. Delete the copyOfDisplay variable.
  - b. Change the next to last line in the method from:

```
[display setStringValue:strcat(copyOfDisplay, theDigit)];
    to:
[display setStringValue:[[display stringValue]
stringByAppendingString:theDigit]]];
```

This fixes the two errors.

#### 14. Fix the errors:

```
"SimpleCalc.m", line 156: Warning: Cannot find instance method valueForStringKey: for class id.
```

### and

"SimpleCalc.m", line 163: Warning: Cannot find instance method valueForStringKey: for class id.

You can replace both of these calls to NXStringTable with @"0".

#### 15. Fix the error:

"SimpleCalc.m", line 207: Error: Formal argument charCode of type NSString\* in call to NSButton::-[setKeyEquivalent:](NSString\*) is being passed int.

The setKeyEquivalent method now takes an NSString object as the key equivalent. You can use @"\003" to specify the Enter key.

#### 16. Fix the errors:

"SimpleCalc.m", line 224: Warning: Cannot find instance method loadNibSection:owner: for class id.

#### and

"SimpleCalc.m", line 233: Warning: Cannot find instance method loadNibSection:owner: for class id.

In OpenStep, the job of loading nib files is performed by the NSBundle class. Replace the two calls to the NXApp's loadNibSection:owner with a call to NSBundle's class method loadNibNamed:owner.

- 17. Save the changes to SimpleCalc.h and SimpleCalc.m.
- **18.** Click the Compile button on the xemacs tool bar to run cvtmake again. You should have no compile errors this time.
- 19. Complete the deep port by replacing character strings with NSString objects.

Replace the use of character strings in the numberKeys method.

20. Fix SimpleCalc.m so that the numeric keypad can be used.

The buttons on the calculator have keyEquivalents set in the SimpleCalc.nib file to the keys on the numeric keypad. In OpenStep, keyEquivalents (by default) need to have the command key pressed simultaneously in order to be recognized.

To fix this, set the default modifier key mask to 0 for each button. This can be accomplished in the <code>applicationDidFinishLaunching</code> method. Note that OpenStep changes the <code>keyEquivalent</code> for the Enter key to be " $\r$ " for the Sun keyboard.

# Compiling Your Project

Now that you've completed the conversion processes, you are ready to compile and debug the application. With more complex programs, you will probably have many remaining compiler and run-time errors. At this point, the calculator application should compile and run cleanly.

**♦** Build and run the application from ProjectBuilder.

## Ported Source Code

## SimpleCalc.h

```
SimpleCalc -- Randy Nelson, NeXT Computer, Inc.
   Ported by Paradigm Research, Inc.
 * A general class that directly supports a calculator interface
   Created 8-8-90
  You may freely copy, distribute and reuse the code in this
   example.
 * Paradigm Research, Inc. disclaims any warranty of any kind,
   expressed or implied, as to its appropriateness for any
   particular use.
#import <AppKit/AppKit.h>
@interface SimpleCalc:Object
id infoPanel;
id helpPanel;
id display; /* a text object to show the output and input */
id enterKey; /* outlet to set the key equivalents */
id stringSet; /* the string external to the source */
int theOperator;/* represents the current operator or zero */
double accumulator;/* a buffer to hold the first number entered */
BOOL numberHasADecimal,
    startingSecondNumber,
    treatingOperationKeyLikeEqualKey,
    noFirstNumber;/* flags that describe the calculator's state */
}
```



```
- doInit;
/* initialize an instance of the class and its flags for start-up
* also used by the clear all key
- numberKeys:sender;
/* sent by any number key or the decimal point key in the interface
(0-9, .)
* appends the character to display using appendToDisplay:
* erases the previous number if startingSecondNumber
- numberDirectFromDisplay:sender;
/* sent as the action of displayer, allows direct entry of numbers
* startingSecondNumber gets YES
* /
- equalsKey:sender;
/* sent by the equals key in the interface (=)
* at this point, accumulator holds the first number
* operator holds an integer defined to an operation
* displayer hold the second number
\mbox{\scriptsize *} performs the operation and leaves the result in the dsplayer
* /
- operationKeys:sender;
/* sent by the operation keys in the interface (+, -, * and /)
* sets the value of operation
* acts like equals to chain a calculation when
* treatingOperationKeyLikeEqualKey
- clearKeys:sender;
/* sent by either clear key in the interface(clear, clear all)
* clear-zeros the display-allows re-entering a number
* clear all resets by calling init
* /
- decimal;
/* called by numberKeys when it finds the number is a decimal point
* checks first if numberHasADecimal already
 * if not -- appendToDisplay: a decimal point
```

```
- appendToDisplay:(NSString *)theDigit;
/* sent by objects wanting to append a digit to the number in the
* displayer removes the leading zeros a zero value displayer has
* unless there is a decimal point
- (void)applicationDidFinishLaunching:(NSNotification
*)notification;
/* handles some key equivalent setting and orders the window front
* should be application's delegate to receive
- (void)windowWillClose:(NSNotification *)notification;
/* quits application when window is closed
* for example, application's quit menu item can send performClose:
 * to window should be Window's delegate to receive
- infoPanel:sender;
- helpPanel:sender;
/* sent by the menu in interface
* creates the panels as they are needed
* /
@end
```

## SimpleCalc.m

```
* SimpleCalc -- Randy Nelson

* Ported by Paradigm Research, Inc.

* A general class that directly supports a calculator interface

* Created 8-8-90

* You may freely copy, distribute and reuse the code in this

* example.

* Paradigm Research, Inc. disclaims any warranty of any kind,

* expressed or implied, as to its appropriateness for any

* particular use.

* */
```

```
#import "SimpleCalc.h"
#define DECIMALPOINT 10
#define DIVIDE 20
#define MULTIPLY 21
#define SUBTRACT 22
#define ADD 23
#define CLEAR 31
#define CLEARALL 30
@implementation SimpleCalc
- init
{
   if (self = [super init])
       [self doInit];
   return self;
}
- doInit
   accumulator = 0;
   theOperator = 0;
   numberHasADecimal = NO;
   startingSecondNumber = NO;
   treatingOperationKeyLikeEqualKey = NO;
   noFirstNumber = YES;
   return self;
}
- numberKeys:sender
   int digit = [sender selectedTag];
   noFirstNumber = NO;
    /* clear the display in order to begin the second number */
   if(startingSecondNumber == YES)
       [display setStringValue:@""];
       startingSecondNumber = NO;
    }
```

```
if(digit == DECIMALPOINT)
       [self decimal];
       return self;
   [self appendToDisplay:[NSString stringWithFormat:@"%d", digit]];
   return self;
}
- decimal
    if(numberHasADecimal == YES)
       NSBeep();
       return self;
    [self appendToDisplay:@"."];
   numberHasADecimal = YES;
   return self;
}
- operationKeys:sender
    //do nothing if noFirstNumber
   if(noFirstNumber == YES)
       NSBeep();
       return self;
   /* if there's an operation and a first and second number
    * function as though the equal key has been pressed
   if((treatingOperationKeyLikeEqualKey == YES) &&
      (startingSecondNumber == NO))
       [self equalsKey:self];
    }
```

```
theOperator = [sender selectedTag];
   switch(theOperator)
       case DIVIDE: case MULTIPLY: case SUBTRACT: case ADD:
           accumulator = [display doubleValue];
           numberHasADecimal = NO;
           startingSecondNumber = YES;
           treatingOperationKeyLikeEqualKey = YES;
           break;
       default:
       return self;
    }
   return self;
- equalsKey:sender
 /* do nothing if noFirstNumber */
   if(noFirstNumber == YES){
       NSBeep();
       return self;
   }
   switch(theOperator){
   case DIVIDE:{
       if([display doubleValue] == 0){
       //can't divide by zero
       NSBeep();
       return self;
       [display setDoubleValue:(accumulator / [display
       doubleValue])];
       break;
   case MULTIPLY:
       [display setDoubleValue:(accumulator * [display
       doubleValue])];
       break;
```

```
case SUBTRACT:
       [display setDoubleValue:(accumulator - [display
       doubleValue])];
       break;
   case ADD:
       [display setDoubleValue:(accumulator + [display
       doubleValue])];
       break;
   default:{
       return self;
   /* just like doInit
   * but startingSecondNumber gets YES
   * and noFirstNumber continues to be NO (there is a first number)
   theOperator = 0;
   accumulator = 0;
   numberHasADecimal = NO;
   startingSecondNumber = YES;
   treatingOperationKeyLikeEqualKey = NO;
   return self;
}
- clearKeys:sender
   switch([sender selectedTag])
       case CLEAR:
           [display setStringValue:@"0"];
           numberHasADecimal = NO;
           startingSecondNumber = YES;
           break;
       case CLEARALL:
           [display setStringValue:@"0"];
           [self doInit];
            break;
       }
```

```
default:
           break;
    }
    return self;
}
- appendToDisplay:(NSString *)theDigit
{
    /* if the display's current value is zero */
    if([display doubleValue] == 0){
    /* and the zero doesn't have a decimal point */
   if(numberHasADecimal == NO){
        /* replace the display with the digit passed */
       [display setStringValue:theDigit];
       return self;
   }
   }
   /* otherwise, append the digit passed
    {}^{\star} to the digits already in the display
    * /
    [display setStringValue:[[display stringValue]
stringByAppendingString:theDigit]];
    return self;
}
- numberDirectFromDisplay:sender
   noFirstNumber = NO;
   startingSecondNumber = YES;
   return self;
- (void)applicationDidFinishLaunching:(NSNotification
*)notification
    /* enter key can't be set as key equivalent from Interface
    * Builder */
    [enterKey setKeyEquivalent:@"\003"];
    [[display window] makeKeyAndOrderFront:self];
}
```

```
- (void)windowWillClose:(NSNotification *)notification
   static int closing = 0;
   if (!closing)
       closing = 1;
       [NSApp terminate:self];
}
- infoPanel:sender
   if(infoPanel == nil){
   [NSBundle loadNibNamed:@"Info.nib" owner:self];
    [infoPanel orderFront:sender];
   return self;
}
- helpPanel:sender
    if(helpPanel == nil){
   [NSBundle loadNibNamed:@"Help.nib" owner:self];
   [helpPanel orderFront:sender];
   return self;
}
@end
```

Copyright 1996 Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, Californie 94043-1100 U.S.A.

Tous droits réservés. Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, et la décompliation. Aucune partie de ce produit ou de sa documentation associée ne peuvent Être reproduits sous aucune forme, par quelque moyen que ce soit sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il en a.

Des parties de ce produit pourront etre derivees du système UNIX®, licencié par UNIX System Laboratories Inc., filiale entierement detenue par Novell, Inc. ainsi que par le système 4.3. de Berkeley, licencié par l'Université de Californie. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fourmisseurs de Sun.

LEGENDE RELATIVE AUX DROITS RESTREINTS: l'utilisation, la duplication ou la divulgation par l'administration americaine sont soumises aux restrictions visées a l'alinéa (c)(1)(ii) de la clause relative aux droits des données techniques et aux logiciels informatiques du DFARS 252.227-7013 et FAR 52.227-19. Le produit décrit dans ce manuel peut Être protege par un ou plusieurs brevet(s) americain(s), etranger(s) ou par des demandes en cours d'enregistrement.

#### **MARQUES**

Sun, Sun Microsystems, le logo Sun, Solaris sont des marques deposées ou enregistrées par Sun Microsystems, Inc. aux Etats-Unis et dans certains autres pays. UNIX est une marque enregistrée aux Etats-Unis et dans d'autres pays, et exclusivement licenciée par X/Open Company Ltd. OPEN LOOK est une marque enregistrée de Novell, Inc. PostScript et Display PostScript sont des marques d'Adobe Systems, Inc.

Toutes les marques SPARC sont des marques deposées ou enregitrées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. SPARCcenter, SPARCcluster, SPARCompiler, SPARCdesign, SPARC811, SPARCengine, SPARCprinter, SPARCserver, SPARStation, SPARCstorage, SPARCworks, microSPARC, microSPARC-II, et UltraSPARC sont exclusivement licenciées a Sun Microsystems, Inc. Les produits portant les marques sont basés sur une architecture développée par Sun Microsystems, Inc.

Les utilisateurs d'interfaces graphiques OPEN LOOK® et Sun™ ont été développés par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnait les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique, cette licence couvrant aussi les licenciés de Sun qui mettent en place OPEN LOOK GUIs et qui en outre se conforment aux licences écrites de Sun.

Le système X Window est un produit du X Consortium, Inc.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" SANS GARANTIE D'AUCUNE SORTE, NI EXPRESSE NI IMPLICITE, Y COMPRIS, ET SANS QUE CETTE LISTE NE SOIT LIMITATIVE, DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DES PRODUITS A REPONDRE A UNE UTILISATION PARTICULIERE OU LE FAIT QU'ILS NE SOIENT PAS CONTREFAISANTS DE PRODUITS DE TIERS.

CETTE PUBLICATION PEUT CONTENIR DES MENTIONS TECHNIQUES ERRONEES OU DES ERREURS TYPOGRAPHIQUES. DES CHANGEMENTS SONT PERIODIQUEMENT APPORTES AUX INFORMATIONS CONTENUES AUX PRESENTES. CES CHANGEMENTS SERONT INCORPORES AUX NOUVELLES EDITIONS DE LA PUBLICATION. SUN MICROSYSTEMS INC. PEUT REALISER DES AMELIORATIONS ET/OU DES CHANGEMENTS DANS LE(S) PRODUIT(S) ET/OU LE(S) PROGRAMME(S) DECRITS DANS DETTE PUBLICATION A TOUS MOMENTS.

