

QNX 4 Operating System

User's Guide

QNX Software Systems Ltd.

175 Terence Matthews Crescent
Kanata, Ontario K2M 1W8
Canada
613-591-0931

© QNX Software Systems Ltd. 1993
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise without the prior written permission of QNX Software Systems Ltd.

Although every precaution has been taken in the preparation of this book, we assume no responsibility for any errors or omissions, nor do we assume liability for damages resulting from the use of the information contained in this book.

Printing history:

May 1992	First edition
January 1993	Minor changes
August 1993	Minor changes
December 1993	Reprint
April 1994	Reprint
September 1994	Reprint
April 1995	Reprint

QNX and the  symbol are registered trademarks of QNX Software Systems Ltd.
All other trademarks and registered trademarks belong to their respective owners.

UNIX Software Acknowledgments and Copyrights

The following individuals and organizations have placed the utilities listed below in the public domain. We have enhanced some of these utilities and hope you will find them helpful. The full source code for these utilities along with additional notes and credits are available from our online update and conferencing system QUICS:

RCS Utilities

© 1982, 1988, 1989 Walter Tichy

© 1990 Paul Eggert

Distributed under license by the Free Software Foundation Inc.

RCS is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 1, or (at your option) any later version.

The full source code for RCS as well as a copy of the GNU General Public License can be found in electronic form on QUICS. You can also write to:

The Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.

Cover art by Robert Timney

Printed in Canada.

UX14H-07

Acknowledgments

Several members of QNX Software Systems Ltd. have contributed to the design and implementation of QNX 4. This product embodies more than a decade of research, development, and field-tested experience. We are proud to see the QNX operating system enter the standards arena without sacrificing the fundamental qualities of its original vision.

The following people deserve a special word of thanks and acknowledgment for their substantial contribution to QNX 4:

Software Development:

Luc N. Bazinet
Gordon Bell
Andrew C. Boyd
Robin Burgener
Brian Campbell
Grant D. Courville
Dan Dodge
Andrew Edgar
William A. Flowers
Dan Hildebrand
Eric Johnson
Steve McPolin
Rob Oakley
Peter van der Veen
Martin Walter

Documentation:

Kevin Didsbury
Paul N. Leroux
John M. Ostrander

About this Guide

What will I find in this guide?

The QNX operating system *User's Guide* accompanies the QNX operating system (release 4) and is intended for both system administrators and end-users.

The *User's Guide* contains two main parts:

- **Getting Started**—a quick tour through keyboard usage, command basics, and the QNX filesystem
- **System Administration**—a detailed look at installing, configuring, and maintaining your system



Throughout this guide, we assume that as the system administrator, you have *superuser* privileges, which you acquire by logging in using the `root` login account.

Conventions used in this guide

Technical terms

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications. The following table summarizes our conventions.

Reference	Example
command names	ls
data types	<i>long</i>
environment variables	PATH
error numbers	[EINTR]
filenames	/dev/tty
functions	<i>system()</i>
function arguments	<i>arg1</i>
global externals	<i>errno</i>
headers	<sys/stat.h>
options	-c
option arguments	-w <i>width</i>
operands	<i>file_name</i>
parameters	<directory pathname>
symbolic constants	{_POSIX_VDISABLE}

How keys are displayed in print

When this guide instructs you to press keys on your keyboard, the keys you're to type appear between angle brackets. For example, if a passage instructs you to:

press <Enter>

you press the key labeled Enter.

Key chords

When this guide instructs you to press more than one key at the same time, the keys are separated by a hyphen (-). For example, if you see:

press <Ctrl>-<N>

you would:

- 1) press <Ctrl> and hold it down
- 2) press <N>
- 3) quickly release both keys.

Entering text

When this guide instructs you to enter text from your keyboard, the text you should type appears in a **constant-width typeface**. For example, if you see:

Type **pwd**

you would type:

pwd

Important notes

Important text is indicated by a pointing hand (☞). For example:



If you want to move a file from another user's directory, you must first have the appropriate permissions on that directory.

Contents

Part I Getting Started

Chapter 1	Using the Keyboard	3
	Introduction	4
	Entering line-oriented input	4
	Line-editing keys	4
	Maximum length of an input line	5
	Entering long input lines	5
	Recalling commands	5
	Switching virtual consoles	6
	Using multiple consoles	7
	Changing the console fonts	7
	Suspending and resuming output	8
	Killing a process	8
	Invoking the system debugger	8
	Rebooting	9
	International keyboards	9
	The keyboard at a glance	10
Chapter 2	Command Basics	11
	Understanding command syntax	12
	Syntax structure	12
	Key symbols	13
	Grouping options	14
	Quoting special characters	14

Command Basics (cont'd)

- Redirecting input and output 16
- Executing commands on another node or tty 17
- Displaying online usage messages 18
- Using wildcards 19
- Entering multiple commands 20
- A note on shell scripts 20
- Want to know more? 21

Chapter 3 Working in the QNX Filesystem 23

- Introduction 24
- Filenames 24
 - Portable filenames 24
 - Filename suffixes 24
- Pathnames 25
 - Network pathnames 25
 - Current working directory 26
 - Home directory 26
- A typical QNX filesystem 27
- Commonly used utilities 29

Part II System Administration

Chapter 4 Basic Installation 33

- Installing QNX onto a hard disk 34
- Installing additional software 35
- The system initialization file 36
 - What happens when you boot 36
 - `sysinit.node` 37
 - `sysinit` 38
 - `altsysinit` 38

Using the system initialization file	39
Base-level services	39
Optional services	41
Supporting non-USA keyboards	42
Time zones and the realtime clock	42
Establishing the time zone	43
Getting the date and time from the realtime clock	43
Licensing	45
Operating system licensing	45
Application licensing	45
Expanding your license	46

Chapter 5 Network Installation 49

Introduction	50
Logical node IDs	50
Logical network IDs	50
Physical node IDs	50
Setup considerations	51
Network drivers	51
Determining physical node IDs	51
Assigning logical node IDs	52
Boot servers and workstations	52
Planning your network	53
Configuring a boot server	54
Configuring a workstation	58

Chapter 6 Setting up User Accounts 61

Security	62
Access control utilities	62
User and group IDs	64
File permissions	65
The password database	69
Default password files	70

Setting up User Accounts (cont'd)

- Accounting file 71
 - Enabling accounting 71
 - Record format 71
 - Clearing the logfile 72

Chapter 7 Connecting Terminals, Printers, & Modems 75

- Starting device drivers 76
- Parallel devices 76
 - Single parallel port 76
 - Multiple parallel ports 76
 - Output buffers 77
- Serial devices 77
 - Hardware adapters 77
 - The RS-232 serial protocol 79
 - Configuring serial ports 81
 - Connecting serial devices 82
 - Configuring serial lines for terminals and users 86
 - Troubleshooting serial device problems 89

Chapter 8 Setting up a Second Filesystem 91

- Introduction 92
- Partitioning the pathname space 92
 - A hard disk and a floppy 92
 - Two hard disks (same node) 93
 - Two or more QNX partitions 93
 - Local and remote hard disks 93
- Setting up a DOS filesystem 95
 - Invocation modes 95
 - Starting **Dosfsys** 96
 - Dosfsys** name adoption 96
 - DOS devices 97
 - DOS version support 97

DOS text files	98
DOS binary files	98
QNX-to-DOS character and name mapping	98
DOS volume labels	100
DOS/QNX permission mapping	100
File ownership	101
Terminating Dosfsys	101
Error codes returned by Dosfsys	101

Chapter 9 Advanced Network Configuration 103

Introduction	104
Multiple boot servers	104
Broadcast booting with a QNX ROM	104
Running the netboot utility	105
Multiple network links	106
Benefits	106
Communicating across networks	106
Configuring multiple network links	108
Adding network cards	108
Starting network drivers	109
Editing the netmap file	111
Editing the netboot file	111
Examples	112

Chapter 10 Print spooling 117

Introduction	118
Sharing resources on a network	118
Spoolers	118
Using the spool utilities	119
Starting the spooler	119
Submitting spool jobs	119
Querying spool jobs	120
Canceling spool jobs	120
Controlling the spool queues	120
Spooler architecture	121

Print spooling (cont'd)

- The spool setup file 124
 - Syntax definitions 124
 - Global keywords 127
 - Metavariables 128
 - Default behavior 128
- Using setup files 129
 - Queues and targets 129
 - Filters 129
 - Chaining queues 131
 - Accounting information 131
 - Dealing with errors 131
- Example setup files 132
 - Multiple queues feeding a single target 132
 - Multiple queues feeding three targets 133
- Accessing spoolers and queues 135
 - LPSRV and LPDEST 135
 - Examples 136
 - Initialization files 138

Chapter 11 Building a Custom Operating System Image 139

- Introduction 140
- Constructing a build file 140
 - Setting heap sizes 141
 - Setting priorities 141
 - Selecting processes for an image 141
 - Disk images 142
 - Network images 144
 - Embedded images 144

Chapter 12 Using the Revision Control System (RCS) 145

- What is RCS? 146
- Getting started 147
 - Checking in 147
 - Checking out 148
 - Locking files 148
 - Creating an RCS directory 149
 - Protecting your working file 149
 - Assigning revision numbers 150
- Automatic identification 151
 - Want more information? 151
- RCS file format 152
 - Example 155
- RCS commands 155

Chapter 13 Making Backups 157

- Introduction 158
- When to back up 158
- Backup formats 158
 - Archive backups 159
 - Filesystem backups 160
- Backup media 160
 - Floppy 160
 - Tape 161
 - Removable disk 162
 - Fixed disk 162
- Compression 162
- Archive examples 163
 - Compressed floppy archive 163
 - UNIX-compatible floppy archive 163
 - Tape archive 163
 - Cartridge/optical 164

Chapter 14 Disk & File Recovery 165

- Introduction 166
- Overview of QNX 4 disk structure 166
 - Partition components 167
 - Directories 169
 - Links 170
 - Extent blocks 171
 - Files 172
- File maintenance utilities 173
 - fdisk** 173
 - dinit** 173
 - chkfsys** 173
 - zap** 174
 - spatch** 174
- Disk recovery procedures 174
 - Using **chkfsys** 174
 - Recovering deleted files 176
 - Recovering from a bad block in the middle of a file 176
- What to do if your system will no longer boot 177
 - If the disk is unrecoverable 181
 - If the filesystem is intact 181
- Recovering lost files and directories 182
 - The **lost+found** directory 182
 - Using **spatch** 182

Chapter 15 The Quantum Update System 185

- Reporting bugs 186
- Free software 186
- Electronic mail 186
- Conferencing system 187
- About our hardware 187
- How to phone us 188
- Logging in 189
- Using QUICS 189

<i>Appendix A</i>	The QNX Terminal	191
	Introduction	193
	Standard console character set	194
	QNX keyboard codes	196
	Composed character input sequences	198
	Terminal escape sequences	199
<i>Appendix B</i>	Time Zone Rules	203
	Index	209

Part I

Getting Started

In this Part...

Using the Keyboard

Command Basics

Working in the QNX Filesystem

Chapter 1

Using the Keyboard

In this Chapter...

Entering line-oriented input

Recalling commands

Switching virtual consoles

Using multiple consoles

Changing the console fonts

Suspending and resuming output

Killing a process

Invoking the system debugger

Rebooting

International keyboards

The keyboard at a glance

Introduction

This chapter describes the standard keyboard conventions in QNX. Remember that some keys may behave differently from how they're described here, depending on how your system has been configured.

Entering line-oriented input

Line-editing keys

Many applications run in *edited* mode. If an application runs in this mode, you can use the following keys for entering line-oriented input:

	If you want to:	press this key:
Moving the cursor	move the cursor left one position	← (left arrow key)
	move the cursor right one position	→ (right arrow key)
	move the cursor to the beginning of the line	<Home>
	move the cursor to the end of the line	<End>
Deleting characters	delete the character to the left of the current cursor position*	← (gray arrow key)
	delete the character at the current cursor position	
	delete all characters on the current line	<Ctrl><U>
Inserting/overwriting characters	toggle between insert mode and typeover mode (default is insert)**	<Ins>
Submitting a line	submit a line of input	<Enter>

* The gray arrow key may be labeled <Rubout> or <Backspace> on your keyboard. Pressing this key generates a 7F hex (ASCII Rubout), not a 08 hex.

** If you're in typeover mode and you submit a line, you'll be returned to insert mode.



The QNX Shell has additional input editing commands. For more information, see **sh** in the *Utilities Reference*.



Your keyboard may not behave as indicated if:

- you're working with an application that has complex requirements for user interaction—the application may take control over how the keyboard works
- you're working at an attached terminal—the terminal may have keyboard limitations

Maximum length of an input line

Entering long input lines

The maximum length of an input line is 256 characters. Application programs can impose lower limits.

If you enter a single line of input that's too long for your screen to display as a single line, the line may be displayed as several screen lines. The line still behaves as a single input line, however, when you use the standard line-editing keys.

For more information on edited mode, see *System Architecture*, Chapter 6, The Device Manager.

Recalling commands

The Shell lets you recall commands that you've previously entered, then re-execute them. These commands are maintained by the Shell in a buffer.

If you want to move:	press this key:
backward through the buffer	↑ (up arrow key)
forward through the buffer	↓ (down arrow key)

When you recall a command, you're free to alter it using the standard line-editing keys.

Switching virtual consoles

The display adapter and the screen, plus the system keyboard, are collectively referred to as the *console*. To let you interact with several applications all at once, QNX permits multiple sessions to be run concurrently on consoles by means of *virtual consoles*. These virtual consoles are usually named `/dev/con1`, `/dev/con2`, etc.

Each virtual console can be running a different foreground application that uses the entire screen. The keyboard is attached to the virtual console that's currently visible. You can switch from one virtual console to another—and thus from one application to another—by entering the following key chords.

If you want to see:	press:
the next active console	<code><Ctrl>-<Alt>-<Enter></code> or <code><Ctrl>-<Alt>-<+></code>
the previous active console	<code><Ctrl>-<Alt>-<-></code>
<code><+></code> is the gray plus key <code><-></code> is the gray minus key	

You can also “jump” to a specific console using the `<Ctrl>-<Alt>-<n>` key chord, where *n* is a numeric digit that represents the console number of a virtual console.

If you want to see:	press:
<code>/dev/con1</code>	<code><Ctrl>-<Alt>-<1></code>
<code>/dev/con2</code> (if available)	<code><Ctrl>-<Alt>-<2></code>
<code>/dev/con8</code> (if available)	<code><Ctrl>-<Alt>-<8></code>

You can disable keyboard console switching with the `stty +noswitch` command.

For more information on the QNX console, see *System Architecture*, Chapter 6, The Device Manager.

Using multiple consoles

The system administrator can specify how many virtual consoles are supported on your machine by specifying an argument to the console driver process (**Dev.con**) when it is started.

The administrator can also specify the program—if any—that is initially launched on each console. By default, the terminal initialization utility (**tinit**) launches a **login** on the first console only, but will be “armed” to launch a **login** on any other console on which you press a key. This means that while console 1 is always available, any other given console won’t be used unless you specifically switch to that console and press a key.

To start a **login** on an unused console, follow these steps:

- 1) Switch to the unused console via <Ctrl>-<Alt>-<n>.
- 2) Press any key. A **login** will be launched.

You can now access the console via any of the cyclical console-switching key chords (e.g. <Ctrl>-<Alt>-<+>) described in “Switching virtual consoles.”

When you terminate the session by typing **logout** or **exit**, or by pressing <Ctrl>-<D>, the console will once again be idle. It won’t appear when you use any of the cyclical console-switching key chords. The exception is console 1, on which the system usually restarts a **login**.

Changing the console fonts

Depending on your video hardware, the console driver (**Dev.con**) may support a variety of screen fonts. The available fonts are numbered from 0 to *n*. When QNX boots, it defines font 0 as a 25×80 text font. If you’re using an EGA or VGA video adapter, QNX also defines font 1 as a 43×80 text font (EGA) or a 50×80 text font (VGA).

To inform **Dev.con** about new fonts or to redefine existing fonts, you use the **cfont** utility. You can use this utility to provide fonts of different sizes or fonts that contain alternate character sets.

To change the font associated with the current console, you use the following key chords:

If you want to choose:	press:
the next font (up to <i>n</i>)	<Ctrl><Alt>-< > >
the previous font (down to 0)	<Ctrl><Alt>-< < >
< > > is the "greater than" key	
< < > is the "less than" key	

QNX keeps track of the font being used by each console. All consoles initially display font 0. You can disable keyboard font changing with the **stty +noresize** command.

Suspending and resuming output

If you want to:	press:
suspend the display of output	<Ctrl><S>
resume the display of output	<Ctrl><Q>

Killing a process

If you need to kill the process currently running on the console, press <Ctrl><C> or <Ctrl><Break>. The system will attempt to kill the process.

Invoking the system debugger

QNX comes with a low-level system debugger that lets you set breakpoints in programs, display and edit memory, disassemble code, and examine I/O ports. If it has been built into your **OS** image, you can invoke this debugger with the following key chord:

<Ctrl><Alt>-<Esc>

You can disable this key chord with the `stty +nodebug` command.



Don't use this debugger in a multiuser environment, because it disables interrupts and freezes the entire system—it's intended only for low-level debugging. For more information on this debugger, see **Debugger** in the *Utilities Reference*. For information on general debugging, see the WATCOM VIDEO *User's Guide*.

Rebooting

To reboot your computer, you use this key chord:

`<Ctrl>-<Alt>-<Shift>-<Del>`

You can disable this key chord with the `stty +noboot` command.



Before entering this command, make sure that no applications or utilities are running on your computer. If there are, files may be left open. Moreover, if you reboot when a critical update is in progress, it's possible that the filesystem would require maintenance (see the section on "Filesystem robustness" in *System Architecture* and see also Chapter 14, *Disk & File Recovery*, of this guide).

International keyboards

Some keyboard layouts—the French and German layouts, for example—use accent keys which, by themselves, don't generate a character. QNX treats these keys as "dead" keys. Pressing a dead key, followed by a second key, modifies the second key, creating an accented character. For example, to create the ù character, you press `<">`, followed by `<Shift>-<U>`. This dead key processing provides typists with a familiar method of composing characters.

Note that you can also generate composed characters by:

- 1) pressing and releasing `<Alt>`
- 2) typing two characters

For more information, see Appendix A.

The keyboard at a glance

	If you want to:	press:
Line editing (see also <code>ah</code> utility)	move the cursor to the left	←
	move the cursor to the right	→
	move the cursor to the start of a line	<Home>
	move the cursor to the end of a line	<End>
	delete the character left of the cursor	← (gray arrow key)
	delete the character at the cursor	
	delete all characters on a line	<Ctrl>-<U>
	toggle between insert/typeover modes	<Ins>
	submit a line	<Enter>
Recalling commands	recall a command	↑ or ↓
Switching consoles	switch to the next virtual console	<Ctrl>-<Alt>-<Enter> or <Ctrl>-<Alt>-<+>
	switch to the previous virtual console	<Ctrl>-<Alt>-<->
	switch to a specific console	<Ctrl>-<Alt>-<n>
Changing console fonts	choose the next font	<Ctrl>-<Alt>-< > >
	choose the previous font	<Ctrl>-<Alt>-< < >
Controlling output	suspend display of output	<Ctrl>-<S>
	resume display of output	<Ctrl>-<Q>
Killing a process	attempt to kill a process	<Ctrl>-<C> or <Ctrl>-<Break>
EOF	indicate end of input (EOF)	<Ctrl>-<D>
Debugging	invoke the system debugger	<Ctrl>-<Alt>-<Esc>
Rebooting	reboot your computer	<Ctrl>-<Alt>-<Shift>-

Chapter 2

Command Basics

In this Chapter...

Understanding command syntax

Redirecting input and output

Executing commands on another node or tty

Displaying online usage messages

Using wildcards

Entering multiple commands

A note on shell scripts

Want to know more?

Understanding command syntax

Syntax structure

Whenever you look up a command in the *Utilities Reference*, you'll see a *syntax statement* that summarizes how the command can be used. For most commands, this statement consists of three main parts:

command_name options operands

command_name

The name of the command to be executed. This may be the name of an executable program, such as a utility, or it may be the name of a command built into the Shell.

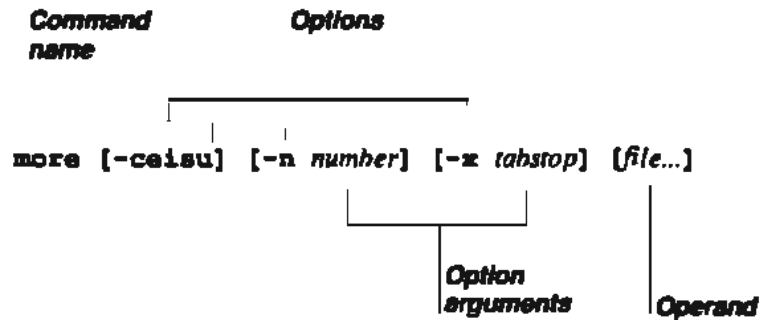
options

These let you specify alternate behavior for the command. Options typically consist of an alphanumeric character preceded by a dash (e.g. "-c"). Some options take an *argument* (e.g. "-n *number*"). If you specify an option that takes an argument, you must include its argument as well.

operands

These let you specify the data the command requires (typically a filename). If a command lets you enter multiple operands, they are usually processed in the order you list them. Unlike options, operands aren't preceded by a hyphen (e.g. "*file...*").

The following illustration shows a typical syntax statement:



Note that the element:

`[-celsu]`

is simply an economical way of indicating that the following options are available:

`-c -e -l -s -u`

Key symbols

When reading a syntax statement, you should be aware of these symbols:

... An ellipsis following an option argument or an operand indicates that more than one can be specified. For example, in the above statement for the `more` utility, the operand:

`file...`

indicates that you can specify more than one file on the command line (e.g. "`more myfile1 myfile2`").

[] Anything enclosed in brackets ([]) is optional. Options are always enclosed within brackets (e.g. "`[-c]`"). If an *operand* is enclosed in brackets, its use isn't mandatory (e.g. "`[file...]`" in the above illustration).

| The OR bar is used between alternatives, indicating that only one can be selected (e.g. "`-a | -f`").

Grouping options

The following examples of **more** command lines show many of the ways you can group options and option arguments (each of these command lines would let you display a file called **myfile**):

- a) You can list each option separately on the command line:

```
more -c -e myfile
```

- b) You can run several options together using a single dash (-):

```
more -ce myfile
```

- c) If you run options together with a dash (-), remember that only the last option can take an argument. So you *can* enter this:

```
more -cen10 myfile
```

but you *can't* enter this:

```
more -n10ce myfile
```

- d) You're free to specify an option argument immediately after the option or separate the two by one or more spaces:

```
more -ce -n10 -x 4 myfile
```

- e) This example shows all the ways you can group options and their arguments:

```
more -c -e -sn10 -x 4 myfile
```

Quoting special characters

Certain characters may have special meaning to the Shell, depending on their context. If any of the special characters recognized by the Shell are to be included in a command line, then you may have to *quote* these characters. Quoting refers to a method of forcing these characters to be treated as simple characters instead of being interpreted in a special way by the Shell.

You *must* quote the following characters to avoid their special interpretation:

	\$	<space>
&	\	<tab>
;	\	<newline>
(	"	
)	'	

You *might* need to quote the following characters, depending on their context within a shell command:

* ? [# ~ = &

There are three methods of quoting characters:

In order to quote:	you can:
a single character	precede the character with a single backslash (\) character
all special characters within a string of characters	enclose the whole string in <i>single</i> quotes
all special characters within a string, except for \$, ', and \	enclose the whole string in <i>double</i> quotes

Examples: Any of the following commands:

```
grep QNX\ 4 chapter1.txt
grep 'QNX 4' chapter1.txt
grep "QNX 4" chapter1.txt
```

would search for all occurrences of the string "QNX 4" within the `chapter1.txt` file. However, note that:

```
grep QNX 4 chapter1.txt
```

wouldn't do what you expect, as it would attempt to find the string "QNX" in the files `4` and `chapter1.txt`.

In addition, this command:

```
grep 'QNX\ 4' chapter1.txt
```

wouldn't search for "QNX 4" either, because the single quotes have caused the Shell to ignore the special meaning of the backslash character. Hence the above command would attempt to find the string:

```
QNX\ 4
```

within the `chapter1.txt` file.

Redirecting input and output

Most commands read their input from the standard input file (`stdin`), which is normally assigned to your keyboard, and write their output to the standard output file (`stdout`), which is normally assigned to your display screen. There are occasions, however, when you want commands to:

- read their input from a file or a device other than the keyboard
- redirect `stdout` to a file or to another command

If you want a process to:	use this symbol:
read from a file, or another device	< (<i>input redirection symbol</i>)
redirect <code>stdout</code> to a file	> (<i>output redirection symbol</i>)
redirect <code>stdout</code> to a file, appending to the file's contents	>> (<i>output append symbol</i>)
pipe <code>stdout</code> directly to another command	(<i>pipe symbol</i>)

For example, let's look at the `ls` command, which lists the files in a directory. If you want the list of files that `ls` outputs to be redirected to a file called `filelist`, you would enter:

```
ls >filelist
```

Executing commands on another node or tty

With QNX, you can execute commands on a machine other than your own if both machines are on the same network—this is known as *remote execution*. Each machine in a network is called a *node*. When a command is invoked on another node, the command's standard input, standard output, and standard error output are displayed on your console screen (or terminal) unless you explicitly redirect them to another device.

To invoke a command on another node, you use the **onnode** command; to redirect (or “attach”) a command's standard input and output to a different tty, you use the **ontty** command.

Using onnode Run the **sin** command on node 4 (**sin**'s output is displayed on the invoking screen, i.e. your own screen).

```
onnode 4 sin
```

The following is equivalent to the above **onnode** command, but uses the *//node command* shorthand notation:

```
//4 sin
```

Load the **sin** command from **/bin** on node 2, and execute it on node 4 (**sin**'s output is still displayed on the invoking screen):

```
onnode 4 //2/bin/sin
```

Run the **sin** command on node 4 and redirect its output to console 1 on node 2 (**sin**'s input and error output streams are still attached to the invoking screen):

```
onnode 4 sin >//2/dev/con1
```

Using ontty Run the **ls** command with its input and output connected to console 3:

```
ontty /dev/con3 ls
```

Combining **onnode** and **ontty**

If you combine **onnode** with **ontty** you can both invoke a command on another node as well as attach the command's input and output to a tty on that node. No virtual circuits will be established back to the node that the command was started from. For example, the following command runs the **nameloc** server on node 4 and attaches *all* of its I/O to **/dev/con1** on node 4:

```
onnode 4 ontty //4/dev/con1 nameloc &
```

Note that even though this program is being run on node 4 and has all its I/O attached to a tty on node 4, it's still a child of the invoking shell. For example, if this command were started from the login shell and the login shell were subsequently exited, the command would have a **SIGHUP** set on it (i.e. it would be terminated). In order to fully detach the command from the invoking shell, you would have to include the **nohup** command. For example:

```
onnode 4 ontty //4/dev/con1 nohup nameloc &
```



For more information on the above commands, see **nohup** and **sh** in the *Utilities Reference*.

For an alternative to using **ontty** and **onnode**, see the **on** utility, also in the *Utilities Reference*.

Displaying online usage messages

Each QNX utility provides an online usage message. You display a utility's usage message with the **use** command. For example, to display the message for **more**, you would type:

```
use more
```

If you request usage for a command and the command either doesn't have an executable in the current path or doesn't contain usage message records, an error message is displayed. For more information on the **use** command, see the *Utilities Reference*.

Using wildcards

Instead of using a command to work on just one file at a time, you can use *wildcards* (also known as *pattern-matching characters*) to perform operations on several files all at once. You can use the following characters:

- * matches zero or more characters
- ? matches any single character
- [] matches any characters (or range of characters separated by a dash) specified within the brackets
- ! excludes characters specified within brackets



Note that directories, as well as files, are matched by these wildcards.

Examples

The following examples show you how you can use wildcards with the **cp** utility to copy groups of files to a directory named **/tmp** (for more information on using **cp**, see the *Utilities Reference*).

If you enter:	cp does this:
cp f* /tmp	copies all files starting with f (e.g. frd.c , flnt)
cp *d /tmp	copies all files ending with d (e.g. frd , fld)
cp fred? /tmp	copies all files beginning with fred and ending with one other character (e.g. freda , fred3)
cp fred[123] /tmp	copies all files beginning with fred and ending with 1, 2, or 3 (i.e. fred1 , fred2 , fred3)
cp fred[1-3] /tmp	same as above
cp *. [ch] /tmp	copies all files ending with .c or .h (e.g. frd.c , barn.h)
cp *. [!o] /tmp	copies all files that <i>don't</i> end with .o

Entering multiple commands

You can enter more than one command at a time by separating your commands with a semicolon (;). For example, if you want to determine your currently working directory and see what it contains, you could enter:

```
pwd;ls
```

Or let's say you want to make a copy of a file in your current working directory, then see the directory's updated contents. You could enter something like:

```
cp myfile myfile2;ls
```

A note on shell scripts

You can enter any number of shell commands into a text file and then invoke the commands in batch mode simply by executing (or “shelling”) the file. A file containing shell commands is commonly known as a *shell script*. You can execute a shell script in either of two ways:

- Another shell can be invoked with the name of your shell script as an argument:

```
sh myscript
```

- The shell script itself can be made executable and will then be treated as a new command:

```
chmod 744 myscript  
myscript
```

In this instance your shell automatically invokes a new shell to execute the shell script.

Want to know more?

Some of the topics covered in this chapter are covered in greater detail in the `sh` man page of the *Utilities Reference*.

To learn more about:	see this section in <code>sh</code> :
quoting special characters	Quoting
redirecting input and output	Redirection
executing commands on another node or tty	Remote execution <i>and</i> Using Shell commands
using wildcards	Expansion
entering multiple commands	Command lists
shell scripts	Shell scripts

Chapter 3

Working in the QNX Filesystem

In this Chapter...

Introduction

Filenames

Pathnames

A typical QNX filesystem

Commonly used commands

Introduction

QNX comes with a rich set of utilities and shell commands that let you perform a wide range of operations on files and directories. To help you work with these commands, this chapter provides basic information on filename and pathname conventions in QNX; it also describes the directory structure of a typical QNX filesystem.



For more information on the QNX filesystem, see *System Architecture*, Chapter 5, The Filesystem Manager.

Filenames

A *filename* is the symbolic name that lets you identify and access a file. In QNX, a filename may be up to 48 characters in length.

You can create filenames with all the characters in the computer's character set except for the following (all values in hex):

- 0x00 to 0x1F (all control characters)
- 0x2F (slash: /)
- 0x7F (rubout)
- 0xFF

Portable filenames

If you anticipate moving your programs and data to other systems, you can make your files more portable by naming them only with those characters defined by POSIX for filenames:

```

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789_.-

```



You can't use the dash (-) as the first character of a portable filename.

Filename suffixes

Filename suffixes (e.g. ".txt" in `myfile.txt`) have no special meaning. The "." character is treated the same as any other character in a filename. Applications often use filename suffix conventions, however. For example, the `make` utility understands `.c`, `.o`, `.y`, etc.

Pathnames

QNX lets you group related files into directories. Directories can also contain other directories, so that you can implement a hierarchical filesystem.

Each filesystem has its own *root directory*. This root is at the top of the directory hierarchy, and it is where QNX starts looking for other directories and files. The typical name for the root directory of the default filesystem on a hard disk is “/”.

If a hard disk has more than one filesystem, the choice of names for filesystems is up to the system administrator. These may include names such as `/hd2`, `/hd3`, etc., or perhaps `/home2`, `/home3`, etc.

To locate a file, you specify the *path* from the root directory to the file. Other directories in the path to a file are also separated by slashes. Thus, to get to the file `sysinit.1` in the `/etc/config` directory, you would specify this path:

`/etc/config/sysinit.1`

Network pathnames

You sometimes need to specify which computer (or “node”) on the network contains the file you want. To specify a node, you use a leading double slash (`//`), followed by the node number of the node. So, if you wanted to access `/home/sam/file` on node 24 of a network, you would enter:

`//24/home/sam/file`

If you don’t explicitly specify a node, QNX uses the current default network root. (For more information, see *System Architecture*.)

Current working directory

To save you from constantly specifying complete pathnames to access a file, QNX provides a shortcut—the *current working directory*.

The advantage of having a current working directory is that you don't have to specify a fully qualified path to access files in or below it. For example, if you're positioned in the directory `/home/gbbell` and you want to access the file `progl.c`, you *wouldn't* need to specify:

```
/home/gbbell/progl.c
```

Instead, you could specify only the filename itself:

```
progl.c
```

Home directory

When you first log in to QNX, your current working directory is set to a directory that has been assigned to you by your system administrator. This directory is known as your *home directory* and is your initial current working directory.

No matter where you are, you can always return to your home directory by entering the following command:

```
cd
```

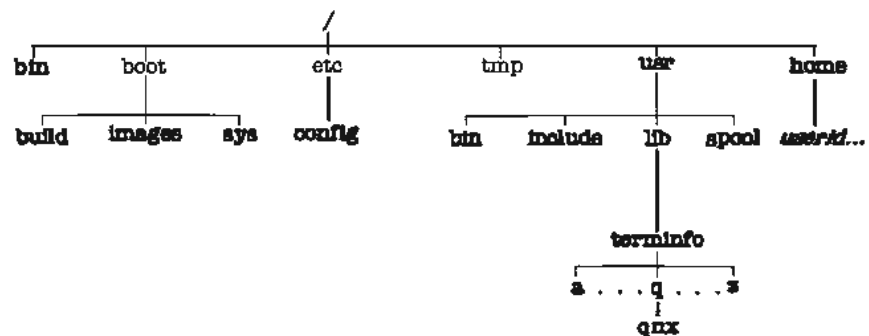


For more information on the pathname space in QNX, see *System Architecture*, Chapter 4, I/O Namespace.

A typical QNX filesystem

QNX 4 follows most of the UNIX conventions for storing standard files. For example, system executables are stored under `/bin` and temporary files are found under `/tmp`.

The following diagram gives an overview of the general directory structure in a typical QNX installation.



The following table summarizes the types of files you'll find under the main directories:

Directory:	Description:
<code>/bin</code>	executables
<code>/boot</code>	OS image Makefile
<code>/boot/build</code>	build files to make images (the make utility reads these)
<code>/boot/images</code>	OS image files
<code>/boot/sys</code>	system processes needed at boot time
<code>/etc</code>	initialization and other files
<code>/etc/config</code>	sysinit and configuration files
<code>/etc/readme</code>	software release information files

A typical QNX filesystem

Directory:	Description:
/etc/readme/technotes	technical notes
/tmp	default location for temporary files
/usr/bin	executables
/usr/include	header (.h) files for the C compiler
/usr/lib	libraries for the C compiler
/usr/lib/terminfo	terminal description files
/usr/lib/application	installed QNX applications
/usr/spool/lp	work files for system spoolers*
/home/userid	user home directories*
* This is configurable by the system administrator.	

Commonly used utilities

The following table lists the utilities you use to perform common operations. For more information, see the *Utilities Reference*.

	If you want to:	use this utility:
Working with directories	change your current working directory	cd
	create directories	mkdir
	remove directories	rmdir
	display your current working directory	pwd
	list the contents of a directory	ls
Working with files	compare two files	diff
	concatenate files	cat
	copy files	cp
	count the number of words, lines, or bytes in a file	wc
	display the contents of a file	more, less, cat
	print the contents of a file	lp
	rename or move files	mv
	remove files	rm
	search for string patterns in files	grep
	sort or merge lines in a file	sort
Other commands	display who is on the system	who
	display processes that are currently running	ps, sin
	find files	find

Part II

System Administration

In this Part...

Basic Installation

Network Installation

Setting up User Accounts

Connecting Terminals, Printers, & Modems

Setting up a Second Filesystem

Advanced Network Configuration

Print spooling

Building a Custom Operating System Image

Using the Revision Control System (RCS)

Making Backups

Disk & File Recovery

The Quantum Update System

Chapter 4

Basic Installation

In this Chapter...

Installing QNX onto a hard disk

Installing additional software

The system initialization file

Using the system initialization file

Time zones and the realtime clock

Licensing

Installing QNX onto a hard disk

You use an automated `install` program to install QNX onto a hard disk.

To install QNX, follow these steps:

Step 1 **Boot from floppy**

Place the QNX *boot diskette* into the floppy drive and reboot the computer. You should see a spinning arrow in the top left corner of your screen, followed by the QNX logo, a welcome message, and a shell prompt.

Step 2 **Run the `install` program**

At the shell prompt (`#`), type:

```
install
```

Simply follow the instructions on your screen to set up your hard disk so it will boot QNX.

Step 3 **Reboot from hard disk**

Once installation is complete, you should remove any floppy diskettes and reboot your computer from hard disk. QNX should now be up and running. At this point, you'll need to log in as `root`.

Step 4 **Read *System Architecture***

It's a good idea to familiarize yourself with the basic philosophy and operation of QNX as described in *System Architecture* and Part I of this *User's Guide*. For the remainder of this guide, we'll assume you have a basic level of familiarity with QNX.



Now that you've installed QNX onto a hard disk, you may want to customize your installation. If so, go to the next section, "The System Initialization File." If you have a network, you should also refer to Chapter 5, "Network Installation".

Installing additional software

All additional software packages supplied by Quantum for QNX 4 come in a standard compressed format. You can install these packages with the `/etc/install` script supplied with the operating system—just follow the instructions below. You might also be able to use this method to install packages from third-party vendors, but always check the instructions that come with each package before you proceed.



Before installing your software, you must be logged in as the superuser (**root**) and the floppy driver must be running. To check if the driver is running, type:

```
sin ver
```

If “**Floppy**” doesn’t appear in the **NAME** column, type:

```
fsys.floppy &
```

To install your software, follow these steps:

- 1 Insert the application’s install disk (normally disk 1) in the floppy drive.
- 2 At the shell prompt, type:

```
/etc/install [drive]
```

where *drive* is the device from which the software is being installed. The default is the local floppy drive (`/dev/fd0`).

Simply follow the instructions on your screen; the software will be installed onto your hard disk.

The system initialization file

What happens when you boot

When you boot QNX, you are booting an *image* composed of several processes. The first process in the image is always the Process Manager (which contains the Microkernel). The Process Manager performs processor initialization, and then schedules each process included within the image for execution. The last process in the image is the `sinit` utility. The `sinit` utility initiates the second phase of your system initialization by starting a shell that executes commands from a file. This file—the system initialization file—contains commands that set up services for the machine. It's a standard shell file that runs just like any other shell file except that breaks are disabled.

Being able to start system services after boot is one of the benefits of QNX's modular architecture. The image that is booted typically contains only those few essential services needed to start all other desired services.



While it is seldom required, you may need to configure a special boot image for some installations (e.g. embedded systems). Once you're familiar with normal system initialization, you can refer to Chapter 11, *Building a Custom Operating System Image*, for details.

When `sinit` runs, it first determines if the image it's part of was booted from disk or from over the network. If the image was booted from disk, `sinit` checks to see if a normal boot or an alternate boot occurred. (You can optionally select an alternate boot by pressing <Esc> when prompted by the boot loader during powerup.) Based upon this information, `sinit` attempts to open one of two system initialization files:

If you choose:	sinit tries to find and open:
a normal boot (from disk or network)	/etc/config/sysinit.node If this fails, it tries: /etc/config/sysinit
an alternate boot (from disk only)	/etc/config/altsysinit If this fails, it tries: /etc/config/sysinit

The **sinit** utility will attempt to open **sysinit** only if it can't open the first file. If it can open neither file, **sinit** terminates and no system initialization will be performed. If the open succeeds, **sinit** replaces itself with the Shell (**/bin/sh**) and passes it the name of the file that **sinit** opened.

Each of the system initialization files that may be executed by **sinit** serves a particular purpose:

- **sysinit.node**
- **sysinit**
- **altsysinit**

sysinit.node

If you wish to customize your installation, this is the file you modify. It contains the custom commands needed to set up the environment and services for your machine, allowing every node in the network to have its own setup. The *node* suffix, which is used by **sinit**, is the logical node ID of your machine. This is a value between 1 and the number of nodes in your network.

A **sysinit.1** file is always created by the **install** program. The file's initial contents reflect the installation parameters determined by **install**.

Customizing workstations

Nodes that boot over the network from a server (i.e. workstations) initially use the default **sysinit** file for booting. To customize a workstation's setup, you have to create a **sysinit.node** file on the boot server for that workstation. You can start with the

sysinit.node file for another node with a similar configuration, or you can simply make a copy of the default **sysinit** file, which you may then modify:

```
cp /etc/config/sysinit /etc/config/sysinit.node
```

Remember that the *node* suffix must be the logical node ID of the workstation you're customizing.

sysinit

This file is executed if a **sysinit.node** file isn't present. It should contain services that will work on any machine. Quantum ships a generic file that should be able to boot any machine. We recommend that you make few or no modifications to this file.

The **sysinit** file is automatically put on your system during the installation.



If you change the logical node ID of a machine, the machine will now look for a different **sysinit.node** file.

altsysinit

This file serves as a safety net in case a modification to your **sysinit.node** file leaves the system in a state where you can't log in. The **altsysinit** file is executed only if you specify an alternate boot when booting from local disk (i.e. you press <Esc> when the boot loader prompts you for an alternate boot).

The **altsysinit** file should always contain a backup of the last working copy of the **sysinit.node** file for a machine that boots from a local hard disk. So, before you make any changes to a working **sysinit.node** file that could prevent the machine from booting from hard disk, you should copy the **sysinit.node** file to **altsysinit**:

```
cp /etc/config/sysinit.node /etc/config/altsysinit
```

We recommend that you update the **altsysinit** file after all successful changes to your **sysinit.node** file.

Using the system initialization file

Base-level services

The contents of each machine's system initialization file reflects the hardware on that machine and the services it is to provide. The following describes a base set of services that will be in most initialization files.



If several machines will be using a common set of commands, you can place the commands in a separate shell file and have the file executed via the `.` (dot) shell built-in. For example, let's say you create a file called **techies** containing commands to be used by all the machines in your Technical Department. The file could be executed from within the **sysinit.node** file of each machine in that department with the following command:

```
. /etc/config/techies
```

You would add node-specific commands after this "dot" line. For more information on the dot command, see **sh** in the *Utilities Reference*.

Establishing the time zone

The following lines define the time zone (EST in this case) and get the time from the realtime clock. These two lines should be the first in the file for machines that boot from hard disk. They are optional in a machine that boots over the network, since the machine inherits this information from the node it boots from.

```
export TZ=EST5DST4,M4.1.0/3,M10.5.0/3  
rtc hw
```

For more information, see the following section on "Time zones and the realtime clock."

Starting device drivers

The following lines start the Device Manager and the console driver with eight virtual consoles, then instruct the Shell to reopen its standard I/O through the new console device.

```
Dev &  
Dev.con -n 8 &  
reopen //0/dev/con1
```

The following lines start up any other device drivers you may need. They start both a serial driver, which will look for COM1 and COM2, and a parallel driver. These drivers will terminate if they can't find the necessary hardware.

```
Dev.ser &  
Dev.par &
```

If you started **Dev.ser**, you might need to use the **stty** utility to change the default line configuration. For example, the following lines change the baud rate to 19200:

```
stty baud=19200 </dev/ser1  
stty baud=19200 </dev/ser2  
etc...
```

Starting the floating-point emulator

If your programs use floating point and you don't have an 80x87 chip, you need to start the floating-point emulator:

```
emu87 &
```

Loading node mappings

When booting a node on a network, you must run the **netmap** utility, which informs the Network Manager of node ID mappings. You should place the following **netmap** command in the **sysinit.node** file, even if the node isn't currently running on a network (the command has no effect on a non-networked machine):

```
netmap -f
```

Note that this command is included in the standard **sysinit** files shipped by Quantum.

Starting a name server

At least one machine in the network must run the global name server. You could start the name server on this machine if it isn't rebooted frequently:

```
nameloc &
```

You may also run **nameloc** on other nodes (up to a total of 10) in order to provide redundancy in case one node running **nameloc** fails.

Starting a terminal
daemon

The following line starts a login on the first console and arms all other consoles:

```
tinit -T /dev/con* -t /dev/con1 &
```

While it isn't mandatory, almost all installations use the **tinit** utility. Hence system initialization files provided by Quantum contain **tinit**.



When it is included, **tinit** must be the last command executed in the system initialization file.

Optional services

You can add many other services to your system initialization file. You should add these services just before the line containing the **tinit** command. The following examples show commonly used optional services. Note that these utilities typically support command-line options to modify their behavior—these options are explained in the documentation for each utility.

Defining environment
variables

Define any environment variables you wish inherited by all processes started in the system initialization file:

```
export var=value
```

The **export** command is described in the **sh** man page of the *Utilities Reference*.

Starting a floppy driver

Start a local floppy driver (if the QNX filesystem is running locally, *don't* include the first line):

```
Fsys &  
Fsys.floppy &
```

If a floppy driver is running, you can use the **vol** and **pax** utilities to access the block special file (e.g. **/dev/fd0**). If you want to access it as a QNX filesystem, you must mount it as such (we recommend you use **pax** to transport files between QNX filesystems):

```
mount /dev/fd0 /fd0
```

Starting a DOS filesystem

Start a DOS filesystem to access DOS floppies and partitions. This requires that the QNX filesystem be running with a driver for the floppy and/or hard disk:

```
Dosfsys &
```

Selecting a `cron` server

If you think this machine will rarely be rebooted, you may also select it as a `cron` server:

```
cron &
```

Starting a clock

Start a clock that will always display the time in the upper right corner of the screen:

```
clock -f green "+%X" &
```

Supporting non-USA keyboards

By default, the QNX console driver assumes a 101-key USA keyboard layout. However, QNX provides a variety of configurations corresponding to the keyboards most commonly used throughout the world. If you need to select an alternate keyboard configuration, you use the `kbd` utility.

If you select a non-USA keyboard when installing QNX on a hard disk machine, the appropriate `kbd` command will be added to the machine's `sysinit.node` file. If you subsequently set up workstations, you'll have to change their `sysinit.node` files to include this command.

Note that you can also generate custom keyboard layouts with the `kedit` utility.

Time zones and the realtime clock

It's important that the correct date, time, and time zone information be established early during initialization. This should be the first step in your system initialization file. The `install` program assumes your hardware clock has a valid date and time and asks you for the time zone information.

Internally, QNX uses Coordinated Universal Time (UTC), which is often referred to as Greenwich Mean Time. Applications and utilities convert to local time by using information about your time zone.



If the time zone isn't set, QNX will assume that local time is the same as UTC time. This would present a problem only if you wish to transfer files to another system in a different time zone. In this case, the dates on the file will appear to be shifted by the difference between the two time zones. If you plan to use Quantum's update system to download new software, the time zone should be set correctly.

Establishing the time zone

The time zone information should be established before the current date and time is set. If the realtime clock in your computer has been set to local time, QNX needs the time zone information in order to establish UTC.

In the following example, the time zone as well as the time change rules are set for Eastern Standard Time in North America:

```
export TZ=EST5DST4,M4.1.0/3,M10.5.0/3
```

where:

export	Shell command to set an environment variable
TZ	Name of variable
EST	Eastern Standard Time
5	5 hours before Greenwich Mean Time
DST	Daylight savings time
4	4 hours before Greenwich Mean Time
M4.1.0/3	First Sunday of April at 3am
M10.5.0/3	Last Sunday of October at 3am



For more information on the time zone rules, see Appendix B.

Getting the date and time from the realtime clock

If you're booting from disk, you should follow the first line in your system initialization file with the `rtc` utility to establish the current date and time from the realtime clock. The following two lines would accomplish this:

```
export TZ=EST5DST4,M4.1.0/3,M10.5.0/3  
rtc hw
```

This will set the date and time on an IBM AT-compatible computer.

Note that there are two possible approaches to take when setting the realtime clock in your machine:

- setting the realtime clock to UTC
- setting the realtime clock to local time

We recommend that you set the time in the realtime clock to UTC. But if you're also running operating systems that assume the realtime clock is set to local time (e.g. DOS), you'll want to use the `rtc` utility with the `-l` ("`el`") option:

```
rtc -l hw
```

This invocation of `rtc` assumes that the realtime clock was set to local time. Note that when you use local time in the realtime clock, you'll have to manually change the value in the realtime clock when you switch to and from daylight savings time in locales where it's used.



If the time in your hardware clock is incorrect (perhaps the battery has been replaced), you should set the system time using the `date` utility, then set the realtime clock using the `rtc` utility with the `-s` option. For details, see the documentation for these utilities in the *Utilities Reference*.

If you're booting over the network...

If you boot over the network, the booting machine will inherit the UTC time and time zone environment variable from the node it boots from. Therefore, you don't need to put this information in your system initialization file.

Licensing

Operating system licensing

QNX must be licensed for each computer it runs on. This is true whether you're using 10 standalone machines or 10 networked machines.

In a network, each machine is referred to as a “node” and each node is assigned a “logical” node ID. Logical node IDs range from 1 to the number of nodes in the network. For example, a network licensed for three nodes would support logical node IDs 1, 2, and 3. You set the size of the network by running the `license` utility (see the section “Expanding your license”).

To verify licensing, you can use the `ls` command. The following would indicate a single-node QNX license:

```
ls /etc/licenses
qnx0000178n001
  |             |
  |             |Number of nodes
  |             |
  |             |Serial number
  |             |
  |             |Product
```

If you've installed other QNX applications (e.g. Watcom C), their licenses will also be displayed.

Application licensing

Per-node licensing

Applications are usually licensed per node *or* per concurrent use.

Most applications are licensed per node in a QNX network in the same way as the operating system. If you have an application licensed for, say, three nodes, then you would be restricted to running it on only three nodes of your network. (Your network license size may exceed the license size for an application.)

You may run multiple invocations of the application on each licensed node. When you run an application licensed per node, the invoking node will consume one of the licenses. The license will remain with that node until it is rebooted.

By convention, serial numbers that end in “*nddd*” are licensed per node.

Concurrent-use licensing

Applications may also be licensed for the number of copies in concurrent use. For example, if an application is licensed for three concurrent copies, then up to three copies may be run anywhere in the network at one time. Note that on a non-networked single node machine, you may still run only three concurrent invocations.

By convention, serial numbers that end in “*udda*” are licensed per use.

Expanding your license

You expand your license size for the operating system or an application using the **license** utility. This utility installs license serial numbers from floppies to your hard disk. Each QNX boot diskette and application diskette contains a serial number license for one node (or for one use).

The following example transfers the license from floppy drive 0 (“drive A:” in DOS terminology) to your hard disk. Note that the floppy driver (**Fsys . floppy**) must be running:

```
license
```



After adding new serial numbers to your set of licenses, you must activate them. To do so, you can run **license -r**, which reads licenses from **/etc/licenses** on the hard disk and places them in the machine’s in-memory license database.

On a network, you can run **license -R node** to read up-to-date licenses from another node’s current in-memory license database.

Any machines that boot over the network will inherit the serial numbers from the machine they boot from. If you wish to boot networked machines from their local hard disk, you must use the **license** utility to add serial numbers to *each* hard disk you wish to boot from. Once you have done this on a single hard disk, you can use **license** to copy the serial numbers across the network from one hard disk to another and thus avoid having to re-insert floppies. Note that license files cannot be copied by **cp**.

For more information on the **license** utility, see the *Utilities Reference*.



For licensing to work across a network, the boot server (the machine with the licenses) must be running the `nameloc` utility. If a machine has a local hard disk (with licenses) and a network card, but isn't on a network—a portable, for example—it must also run `nameloc` for licensing to work.

Chapter 5

Network Installation

In this Chapter...

Introduction

Setup considerations

Configuring a boot server

Configuring a workstation

Introduction

QNX is a network-distributed operating system. Each computer QNX runs on is called a *node*. A single computer can be considered a one-node network.

Logical node IDs

Each node is assigned a unique logical node ID represented by a positive integer. The operating system uses this logical node ID to communicate with other nodes connected by the network.

You should assign logical node IDs in an uninterrupted sequence starting at 1 (e.g. 1, 2, 3, 4, 5, ...). There are two reasons for this:

- Licensing for QNX is based upon the total number of logical nodes in the network. A five-node network license allows nodes 1 through 5 to communicate. A node 6 would be isolated and ignored, even if one of nodes 1 through 5 were omitted.
- A few important utilities attempt to communicate with each node of the network in turn. They do this by sending messages starting at node 1 and continuing up to node n , where n is the size of the network license. The `nameloc`, `sin net`, and `who -a` utilities all behave this way.

Logical network IDs

Nodes are connected via one or more networks. Each network in an installation is a separate communications link, and hence data transmitted on one network won't be seen on another network. Networks, like nodes, are assigned logical IDs that must be unique in an installation. A single network (the most common case) will default to logical network ID 1.

Physical node IDs

Although QNX processes deal with logical node IDs, the network cards themselves communicate with physical node IDs. This physical ID is typically contained within the network card itself. The format of the physical ID depends on the network type (e.g. Ethernet, Arcnet, Token Ring) and is invisible to most applications. Physical IDs must be unique within a network, but they need not be unique across all networks in an installation.

Setup considerations

Network drivers

Setting up a network typically involves installing a network card in your computer and connecting the card to other computers with a network cable. Ethernet, Arcnet, and Token Ring are all popular network technologies. To use a particular network, you need both an appropriate network adapter card and a QNX network driver that supports that card. In most cases, cards from different vendors for the same network media (e.g. Ethernet) require their own special drivers. Sometimes, however, a driver may support cards from more than one company if the cards provide a similar hardware interface. The network drivers are documented in the *Utilities Reference* and all start with “Net .”



The hardware installation of a network card and the physical connection between cards should be described in the manual provided with the card.

Determining physical node IDs

Ethernet & Token Ring

Determining the physical node ID of a network card depends on the type of card you're using.

Every Ethernet or Token Ring card is shipped with its own physical node ID built into the card. This ID, which is unique worldwide, is 48 bits long in order to conform to the IEEE 802 standard. On Ethernet cards, you should find the ID labeled somewhere on the card or on the box. The address format of the ID may vary. For example, the following are all identical:

```
0000c0 4a9330
0000c04a9330
00 00 c0 4a 93 30
00:00:c0:4a:93:30
0000 c04a 9330
```

Arcnet

Every Arcnet card requires that you program a physical node ID into the card. Depending on the manufacturer, this is done by a dip switch or in a nonvolatile RAM via a menu interface at boot time. The physical ID is 8 bits in length. You cannot use physical ID 0, which is reserved. Since you get to choose the physical node ID, we

recommend that for a single Arcnet network you make the physical node ID the same as the logical node ID. This helps keep things simple.

Assigning logical node IDs

Since QNX processes use logical node IDs, the Network Manager must map logical IDs to the physical IDs used by the hardware. This mapping is defined in the `/etc/config/netmap` file. Each line in this file defines a single-node mapping with the logical node and network IDs followed by a physical node ID. For example, the following line maps logical node 8 into a 48-bit physical ID of 0000c04a9330:

```
8      1      0000c0 4a9330
                        └──────────┘
                        Physical node ID (base 16)
Logical network ID
Logical node ID
```

You can separate the logical and physical IDs by `<space>`s or `<tab>`s. The logical IDs are in base 10, while the physical IDs are in base 16, unless preceded by a “t”, in which case they’re also in base 10. For example, in the case of Arcnet, it’s more convenient to express the physical node ID in decimal:

```
15     1     t15
```

The physical ID cannot exceed 48 bits. The number of bits needed is determined by the network hardware.

Boot servers and workstations

Setting up a QNX network is done in two phases:

- 1) Configuring one or more *boot servers*
- 2) Configuring *workstations*

A computer can be configured to boot QNX from a hard disk or, if it’s a node on a network, it can be optionally configured to boot from the network. In order for nodes to boot from the network, you must first set up one node—a boot server—from which the other nodes may boot. Usually, a node is chosen to be a boot server because it’s rarely rebooted.

The nodes that boot from the network are known as workstations. Since workstations boot from the network, some workstations don't have a hard disk. These are called *diskless workstations*.

When a workstation boots from the network, an operating system image is loaded into its memory; the image inherits the following from the boot server:

- the time
- its list of name locators (see `nameloc` utility in *Utilities Reference*)
- its licensing capabilities

Planning your network

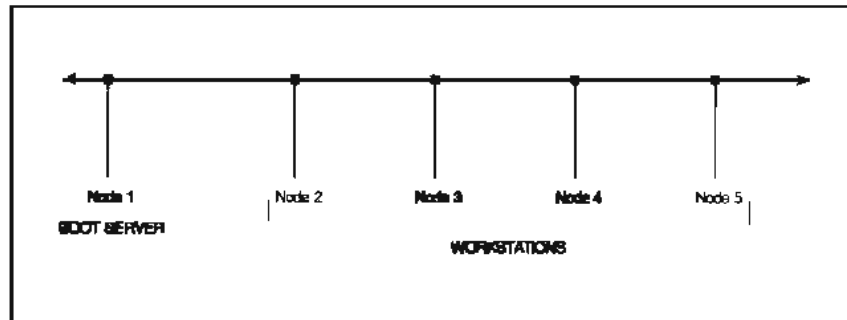
If you're setting up a small network, it will probably need only a single boot server, typically node 1.

If, however, you're setting up a larger network that will span several departments, you should set up a boot server for each department. The users of each department will still be able to access files (subject to file permissions) in other departments. For example, let's assume you have three departments: R&D, Marketing, and Operations. You may elect to have each department boot from its own boot server, which could also act as the department's major file server. This setup creates a fault-tolerant environment—a failure in one department won't stop another department from booting. For large numbers of nodes, this practice should also reduce the potential for a boot bottleneck when people in the office start work first thing in the morning.

This chapter shows you how to set up a boot server; once you've set up a single server, refer to Chapter 9, *Advanced Network Configuration*, if you want to set up additional boot servers.

Preparing a diagram

You should prepare a diagram for your network, similar to the one below. This example diagram shows a network consisting of five nodes. One of the nodes is the boot server and four are workstations that will boot from and obtain their files from the server. We've assigned a logical node ID to each node; the boot server is node 1.



A five-node network consisting of a boot server and four workstations.

Configuring a boot server



Before you start, you must have a computer booting QNX from a hard disk. For more information, see Chapter 4, Basic Installation.

To turn a computer that boots QNX into a *network boot server*, follow these steps:

Step 1 Install your network card



Quantum has provided technical notes for installing network cards. These notes, which you'll find in `/etc/readme/technotes`, have filenames beginning with `Net` (e.g. `Net.ethernet8003`). You should read the technical note for the type of card you're using before installing the card.

Turn off the computer, then install your network card according to the instructions provided with the card. If you can, note the physical node ID of the card. When you're finished, reboot the computer.

Step 2 Install your network licenses

You'll need a network license for each node in your network. When you installed QNX onto your hard disk, a license was created for at least a single node. You can verify this license using the `ls` command. The following would indicate a single-node QNX license:

```
ls /etc/licenses
qnx0000178n001
```

		Number of nodes
	Serial number	
Product		

If you've installed other QNX applications (e.g. Watcom C), their licenses will also be displayed.

QNX ships additional licenses for QNX and applications on diskettes labeled "Network Expansion." To add the licenses from a network expansion diskette, insert the diskette, then run the **license** utility without any arguments. Repeat this operation for each of your expansion diskettes. To activate the licenses, you must run the **license -r** command.

For more information, see the section on "Licensing" in Chapter 4 as well as the **license** utility in the *Utilities Reference*.

Step 3 Modify the **sysinit.node** file

If you specified to **install** that this node was to be a boot server, **install** automatically took care of this step. You can skip to Step 4.



If you didn't tell **install** that this machine was to be a boot server, but have subsequently decided it should become one, then you must:

- 1) Start the commands described in this step
- 2) Add the commands to this node's **sysinit.node** file

Before you add any commands to this node's **sysinit.node** file, copy the file to **altsysinit**:

```
cp /etc/config/sysinit.node /etc/config/altsysinit
```

Starting the Network
Manager & network
driver

The Network Manager (**Net**) and the network driver for your card both must be running for network access to occur.

Configuring a boot server

The names for all network drivers are of the form `Net.xxx`, where the `xxx` represents the appropriate driver suffix. For example, if you have an 8003 series Ethernet card, you could enter:

```
Net &  
Net.ether8003 &
```

To find out which network drivers are available on your system, enter the `ls /bin/Net.*` command.

Starting `nameloc`

The `nameloc` utility must be running on the network—even if it's on just a single node—for licensed products to work:

```
nameloc &
```

Starting `netboot`

When a node wants to boot, it sends a boot request to its boot server. When the Network Manager on the boot server receives a boot request message, it forwards this request to the `netboot` utility, which also must be running on the boot server:

```
netboot &
```

Once you've started all the above commands, you can use your node as a boot server. Once you've added these commands to the `sysinit.node` file, they will be started automatically when the node is rebooted.



You may now wish to further customize the environment and services for your boot server. For more information, see the section on “The system initialization file” in Chapter 4.

Step 4 Set up a logical-to-physical node mapping

If you're installing a boot server on a single Arcnet network (where the physical node ID is the same as the logical node ID), you're done! Otherwise, you must choose a logical node ID for the boot server.

If this is your first (primary) server, we suggest you choose logical ID 1.

To map the logical node ID you've chosen to the physical node ID of the network card, you need to add a line to the **/etc/config/netmap** file. For example, to define mappings for nodes 1, 2, 3, and 4, you would add lines similar to these to **netmap**:

```
1      1      physical_node_ID
2      1      physical_node_ID
3      1      physical_node_ID
4      1      physical_node_ID
```

Note that for an initial network, the logical network ID is always 1.



If you didn't note the physical node ID when installing the card, or the ID wasn't on the card, running **netmap** without any options will print out your physical node ID.

For more information on the **netmap** file, see the **netmap** utility in the *Utilities Reference*.

Step 5 Modify the **/etc/config/netboot** file

If you specified to **install** that this node was to be a boot server, or if you're installing a boot server on a single Arcnet network where the physical node ID is the same as the logical node ID, you're done. Otherwise, you must complete this step, in which you select the proper build file for your network media.

When responding to a boot request, **netboot** uses the **/etc/config/netboot** file to determine which build file is to be used to generate the OS for the requesting node. We've provided generic build files suitable for building workstation images for different media such as Arcnet (e.g. **/boot/build/ws.arcnet**) and Ethernet (e.g. **/boot/build/ws.ether8003**). By default, the **/etc/config/netboot** file contains the line:

```
* f=build/ws.arcnet
```

which is for a single Arcnet network.

If you're using another network type, you'll need to modify the `f=` entry to state which build file to use. For example, for an Ethernet network using 8003 series network cards, your entry would look like this:

```
* f=build/ws.ether8003
```

Configuring a workstation

Step 1 Install your network card



Quantum has provided technical notes for installing network cards. These notes, which you'll find in `/etc/readme/technotes`, have filenames beginning with `Net` (e.g. `Net.ether8003`). You should read the technical note for the type of card you're using before installing the card.

Install your network card in your computer according to the instructions provided with the card. Remember to note the physical node ID of the card. (This is the same procedure you followed in Step 1 of Configuring a boot server.)



If your card needs a boot ROM, make sure the ROM is installed. The Quantum Arcnet card comes with a boot ROM already installed. When booting, the QNX boot ROM will print out the physical node ID.

Step 2 Set up a logical-to-physical node mapping

We recommend you assign workstations with logical node IDs 2, 3, 4, and so on. For more information, see Step 4 of Configuring a boot server.

Step 3 Boot the workstation

The workstation should now boot over the network. If you have any problems, specify `-v` to the Network Manager in the workstation build file (e.g. `/boot/build/ws.arcnet`). This will display network errors on the console.

In addition, you can run the **netinfo** utility, which displays recent network events. You should find this utility helpful whenever you need to perform network diagnostics. For more information on **netinfo**, see the *Utilities Reference*.



If you haven't yet customized the environment and services for your workstations, see the section on "The system initialization file" in Chapter 4.

Chapter 6

Setting up User Accounts

In this Chapter...

Security

Accounting file

Security

QNX provides mechanisms to control access to resources and critical system functions. These mechanisms are based on the ability of the system to identify a particular user.

Access control utilities

The QNX access control utilities are:

login	sign on to the system
su	temporarily become another user
passwd	maintain the user accounts/change passwords
newgrp	switch to a different group

The **login** utility

The **login** utility should be started by **tinit** on all tty devices. The **login** utility demands a username and password that are verified in the password database. If the user fails to enter a correct username/password combination within three attempts, the utility will exit, updating appropriate accounting information. If the user enters a correct combination, **login** will start a login shell and will load the user's environment.

The **su** utility

The **su** utility lets a user temporarily have the privileges of another user. If the user is not the superuser (**root**), **su** will demand the appropriate password. If a valid combination is entered, **su** will do an **exec()** to create a shell for the user with the effective group ID and user ID of the desired user. Exiting from the shell will return the user to where they were before running **su**.

The **passwd** utility

The **passwd** utility may be used to change a password or to add a new user account to the system.

When a user invokes **passwd** to change the password of an existing account, **passwd** asks the user for the old password, the new password, then the new password again (to verify it was entered correctly). If the user is the superuser, the old password isn't requested.

Only the superuser (**root**) may add a new account. The **passwd** utility uses the file **/etc/default/passwd** to determine local policies such as stringency of passwords, user ID ranges, and so on. For example, to create a new account with the username **carol**, you would enter:

```
passwd carol
```

By default, **passwd** will then prompt for the following information:

- user ID
- group ID
- user's real name
- home directory
- login shell
- password

By convention, user accounts have a user ID greater than or equal to 100. User IDs below 100 are often used by system processes.

If the group doesn't exist, a reminder to update the groups file will be displayed. If the group *does* exist, this user will be added to it.

The home directory, login shell, and password items are optional:

If you don't specify a:	then:
home directory	/home/username will be used (will be created if it doesn't exist)
login shell	the system default shell (/bin/sh) will be used
password	the user account will be created without one

If you specify a new password, you'll be asked to enter it twice to verify it was entered correctly.

The **passwd** utility doesn't provide any mechanism for deleting accounts. This function may be performed using a text editor on the **/etc/passwd**, **/etc/group**, and **/etc/shadow** files.

The **newgrp** utility

The **newgrp** utility changes the current process's real and effective group ID. If invoked without arguments, **newgrp** changes the group identification to the group identified in the password database for the current user. The groups that a particular user may change to are controlled by the password database.

User and group IDs

Upon logging in, a user is assigned two pieces of information:

- a user ID
- a group ID

These are known as the user's *real* user ID and *real* group ID.

The user ID should be unique—no two users should share the same user ID. This rule can be enforced by the `passwd` utility, but the superuser can override the rule by editing the password file directly.

The group ID allows several users to be associated with a group. This group mechanism lets a team of users share resources without making those resources available to the rest of the world.

Effective UID and GID

Processes have two classes of user and group IDs: *real* and *effective*. When a process is created, it automatically inherits these four IDs from its parent (which is typically the login shell):

- real user ID (uid)
- real group ID (gid)
- effective user ID (euid)
- effective group ID (egid)

The effective user and group IDs are used for permission checking. A process can change its effective user or group ID, or both, so that they differ from their real counterparts. Typically, this is done to gain access to resources unavailable with the real user and group IDs.

To modify effective group and user IDs:	you use the:
from within C programs	<code>seteuid()</code> and <code>setegid()</code> functions
from within the Shell	<code>setuid</code> and <code>setgid</code> file permission bits

For more information on `setuid` and `setgid`, see the next section on “File permissions.”

File permissions

In QNX, each file has an associated set of permissions called *mode bits*. These mode bits, in conjunction with the file's *owner* and *group*, control access to the file.

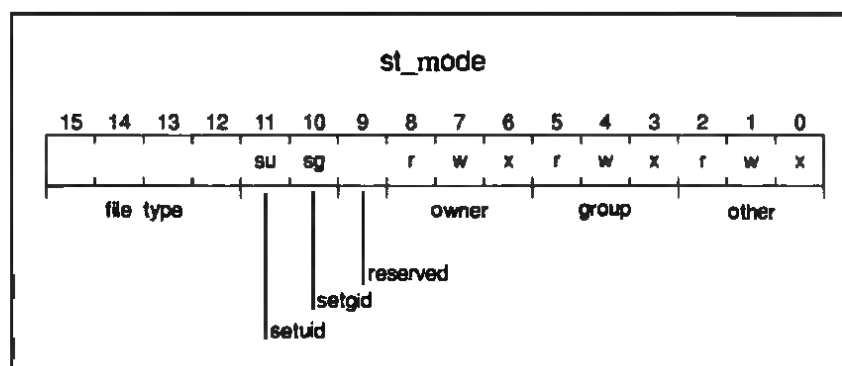
When a process creates a file, the process's effective user ID and effective group ID become the file's owner and group. A set of mode bits is also assigned to the file (this is described in "How mode bits are assigned"). There are three classes of mode bits: owner, group, and other. When a process attempts to access the file, the appropriate class of mode bits is honored by the Filesystem Manager. The Filesystem Manager determines which mode bits to honor by performing a few comparisons, in a specific order:

- 1) The effective user ID of the requesting process is compared to the owner of the file. If they match, the owner mode bits are honored.
- 2) If the owner comparison fails, the requesting process's effective group ID is compared to the file's group ID. If these group IDs match, the group mode bits are honored.
- 3) If both the owner and group comparisons fail, the other mode bits are honored.



As **root** (uid 0), you can access any file for read/write regardless of the permissions, as well as execute any file that has at least one mode bit permitting execute access.

The file mode bits are stored in the 16-bit **st_mode** field of the directory entry for the file; access is granted if a mode bit is set.



As the following table illustrates, permissions affect regular files and directories differently:

Permission:	Type of access allowed for filetype:	
	Regular file	Directory
r	Reading the file	Examining the names of files in the directory (via ls utility, for example)
w	Writing to the file	Creating, removing, and renaming files within the directory, including subdirectories
x*	Executing the file	Searching the directory. This means you can change your current working directory to this directory. Also, you may include this directory within pathnames (e.g. in <i>open()</i> , <i>stat()</i> , ...).

* Execute permission has meaning only for directories and regular files; it doesn't apply to other filetypes.

Viewing permissions

You can use the **ls -l** command to see a file's permissions, owner, and group. The following **ls -l** ("el") output for the file **alpha.c** shows that the owner and anyone in the **techie** group may read and write to **alpha.c**, while other users may only read it:

```
-rw-rw-r-- 1 brianc  techies  8475 May 17 1991 alpha.c
```

In the following example, the **ls -ld** output for the directory **/bin** shows that the owner and group members may read, write, and search the directory, while other users don't have write permissions and therefore can't add any new files to the directory:

```
drwxrwxr-x 2 root  techies  2048 Sep 19 1990 bin
```

Note, however, that other users may still list or execute files in **/bin**.

It's possible—though unusual—to have read permission on a directory but not search permission. This would let **ls** work, for example (depending on the options you give it), but wouldn't let you access any file in the directory or do a **cd** to the directory. This can be confusing. You may end up thinking the system is behaving

improperly, since it's normal to assume that if you can see the presence of a file you should be able to access it, and that if you can look in a directory you can do a `cd` to it.

How mode bits are assigned

The permissions of a file are derived by combining the current *umask* of the creating process with that process's requested *open()/creat()* mode.

The *umask* is a permission bit mask that indicates which permission bits are to be turned off if a process specifies default permission when creating a file, directory, or FIFO (see *umask* in the *Utilities Reference*). This mask is usually stated in octal form (i.e. 0666 in binary would be 110110110, meaning these permissions: *rw-rw-rw-*). Here's an example of using the *umask*:

Requested mode:	110110110	(0666) =	<i>rw-rw-rw-</i>
Current umask:	000100100	(0022) =	<i>---w--w-</i>
Result:	110100100	(0644) =	<i>rw-r--r--</i>

You can change the permissions of a file with the following:

In order to change:	use:
owner and group	<code>chown</code> and <code>chgrp</code> utilities <code>chown()</code> function (C programs)
access permissions, and setuid and setgid mode bits	<code>chmod</code> utility <code>chmod()</code> function (C programs)



To modify the owner or access permissions of a file, your effective user ID must match that of file's owner. If you change the owner of the file to be other than yourself, you'll no longer be able to change the permissions and ownership of the file. The superuser can modify the mode bits of any file, regardless of who owns the file.

Setuid and setgid

Sometimes, in order to perform certain functions, a user must run a command at a privilege level higher than their own. For example, a user may need to run the `passwd` utility to modify their own entry in the password database, even though they would normally be denied write access to that database.

Two file mode bits allow a user to run a command at a higher privilege level: set-user-ID (setuid) and set-group-ID (setgid). With these bits set, an executable file will run with the privileges of its owner and group rather than with the privileges of the process that invokes the file. These mode bits apply only to executable regular files.

When a file is loaded for execution and the setuid mode bit is set, the effective user ID of the new process becomes that of the file's owner. Similarly, if the setgid mode bit is set, the effective group ID becomes that of the file's group.

To close a potential security hole, the setuid and setgid bits are cleared any time the file ownership (owner or group) is changed. This prevents users from writing an executable file to disk, setting the setuid bit, and then changing the ownership of the file to `root`, thereby gaining unlimited system access.

The `passwd`, `login`, `su`, and `newgrp` utilities are all "setuid" to `root`; these programs therefore run with the permissions of the superuser.



By convention, `root` is the only user with user ID zero, which yields superuser status. With respect to access control, you must ensure that only programs that can be trusted—and absolutely need to be trusted—are "setuid" to `root`. (No special privileges are bestowed on a program when it is "setgid" to `root`.)

Since these "setuid" programs will inherit superuser capabilities, you should make sure they do *not* have general write permissions so that only the superuser will be able to modify the programs.

The password database

Three files collectively form the *password database*:

- `/etc/passwd`
- `/etc/shadow`
- `/etc/group`

The access to these files should be as follows:

File	Owner	Group	Permissions
<code>/etc/passwd</code>	root	root	<code>rw-r--r--</code>
<code>/etc/group</code>	root	root	<code>rw-r--r--</code>
<code>/etc/shadow</code>	root	root	<code>rw-----</code>

`/etc/passwd`

The `/etc/passwd` file contains a set of lines in the following format:

```
username:haspw:userid:group:comment:homedir:shell
```

where:

<i>username</i>	Login name of user
<i>haspw</i>	If empty, user has no password; otherwise a password exists in the <code>/etc/shadow</code> file
<i>userid</i>	Numeric user ID
<i>group</i>	Numeric group ID
<i>comment</i>	Free-form comment field; must not contain “:”
<i>homedir</i>	Home directory of this user (default is <code>/</code>)
<i>shell</i>	Initial shell (and arguments) to start from <code>login</code> (default is <code>/bin/sh</code>)

`/etc/group`

The `/etc/group` file contains lines in the following format:

```
groupname:reserved:group:member[,member]*
```

where:

<i>groupname</i>	The name of the group
<i>reserved</i>	Reserved for later use
<i>group</i>	Numeric ID of the group
<i>member</i>	A list of usernames that belong to this group

/etc/shadow The **/etc/shadow** file contains lines in the following format:

userid:password:reserved:reserved:reserved

where:

<i>userid</i>	The name of this user
<i>password</i>	The encrypted password of this user
<i>reserved</i>	The first three numeric fields are reserved for “password aging”; for now, they are always 0.

You may notice from the above permission list that **/etc/passwd** is readable by anyone. This is to provide standard utilities with a simple mechanism to find information about users. Since this file is readable, the encrypted password isn’t stored within it. The encrypted password is stored in the **/etc/shadow** file, which is readable only by the superuser. This is to inhibit unauthorized attempts by malevolent users to *decrypt* the passwords. As a system administrator, you should ensure that these permissions are maintained.

QNX is shipped with a default password database that includes **/etc/passwd** and **/etc/group**. The **/etc/shadow** file isn’t shipped, because the accounts initially don’t have passwords associated with them.

Default password files

The default **/etc/passwd** file that was shipped with your QNX system contains the following:

root::0:0:::/bin/sh

The default **/etc/group** file contains the following:

root::0:root

Accounting file

Accounting information is logged to the `/etc/acclog` file. If this file doesn't exist, all accounting information will be discarded. This is the normal mode of operation after QNX has been installed.

For most realtime systems, this default of discarding accounting information is recommended. If you have a dial-up line to a computer or if you run QNX on a network of many users, you may wish to change this default by creating an empty `/etc/acclog` file.

Enabling accounting

To enable accounting, you create an empty `/etc/acclog` file. You can do this using the `touch` utility:

```
touch /etc/acclog
chmod g=,o= /etc/acclog
```

Once this file is created, accounting information will be logged here.

Note that only the superuser (user ID `root`) should create and modify this file.

Record format

Each record in this text file is of the form:

```
tttttttt cc data...
```

where *tttttttt* is the time in seconds since 1970 (in decimal). This is always followed by a single space. The time is followed by a two-character code *cc*. This code is then followed by a space and *data* specific to each code and terminated by a `<newline>`. The following utilities write these accounting records:

Utility	Purpose	Record
<code>login</code>	user logged in	<i>tttttttt LO device uid gid uname</i>
<code>login</code>	login failed	<i>tttttttt LF device uname</i>
<code>modem</code>	modem connect	<i>tttttttt MO device baud</i>
<code>su</code>	switch user	<i>tttttttt SU device uid gid uname</i>
<code>tinit</code>	start a command	<i>tttttttt TS device command</i>
<code>tinit</code>	arm a device	<i>tttttttt TA device</i>

A typical logfile might look like this:

```
670464500 TS //1/dev/ser1 modem -b 19200 -L
670464545 MO //1/dev/ser1 2400
670464550 LO //1/dev/ser1 100 101 steve
670465824 TS //1/dev/ser1 modem -b 19200 -L
```

This record shows that `tinit` started a `modem` program to wait for calls. A call was received and answered at 2400 baud, and user ID `steve` logged in. Note that the log doesn't show a logout. The logout is inferred, because in the final entry `tinit` starts another `modem` program.

The total connect time for the user (from successful login) can be calculated thus:

$$670465824 - 670464550 = 1274 \text{ seconds}$$

On a busy system, records from many devices will be interspersed throughout the logfile. In order to match events keyed to each device, you'll find a node number associated with each device, letting you track accounting records for all devices throughout a network in a single logfile.

Here are several common event sequences:

TS → LO → TS	A login and logout on a dedicated line.
TS → MO → LO → TS	A login and logout on a dial-up line.
TA → TS → LO → TA	A login and logout on a dedicated line armed by a keystroke.
TS → TS	An unsuccessful login on a dedicated line.
TS → MO → TS	An unsuccessful login on a dial-up line.

Clearing the logfile

Once you create `/etc/acctlog`, the file will start to grow as records are appended to it. If left unmanaged, this file may grow to consume considerable disk space, so you should print or archive the information in this file on a regular basis. You may even want to automate this housekeeping job using the `cron` utility (see the *Utilities Reference*).

In the following example, the logfile is moved to a file named by year and month, and a new empty log is created:

```
mv /etc/acclog /etc/acclogs/9106
touch /etc/acclog
chmod g=,o= /etc/acclog
```



Since the data in this file is very regular, you may use the **freeze** compression utility, which will achieve very high rates of compression on the file. This can significantly reduce disk space requirements if you keep the saved logs online or save them to a floppy. Remember to move the file *before* compressing it. Never compress **/etc/acclog** directly.

Here's an example of the recommended compression procedure:

```
mv /etc/acclog /etc/logs/9106
touch /etc/acclog
chmod g=,o= /etc/acclog
freeze /etc/logs/9106
```

Note also that other utilities (possibly third-party) may add their own accounting records to the **/etc/acclog** file. Utilities may also log problems or unexpected events to the **/etc/syslog** text file.

Chapter 7

Connecting Terminals, Printers, & Modems

In this Chapter...

Starting device drivers

Parallel devices

Serial devices

Starting device drivers

A QNX system will usually contain one or more *terminal devices*. All such devices are managed by the **Dev** process. This process must be started first, before any device drivers are started.

```
/bin/Dev &           Device Manager
```

Once **Dev** has been started, one or more of the following device drivers may be started:

```
/bin/Dev.con &      console device driver
/bin/Dev.par &      parallel printer device driver
/bin/Dev.ser &      serial device driver
/bin/Dev.ptty &     pseudo tty driver
```

Each of these drivers is described in more detail in the *Utilities Reference*.

Parallel devices

Parallel ports are normally used to communicate with parallel printers. Apart from starting up the driver, there's little work for you to do other than connect the printer to the machine.

Single parallel port

If only one parallel port is available on a machine, then no parameters are required:

```
Dev.par &
```

Started this way, the parallel driver will create a device called **/dev/par**, which corresponds to the first parallel port found by the BIOS (LPT1).

Multiple parallel ports

If your machine has more than one parallel port, then you'll need to start an additional **Dev.par** for each extra port. You must provide a unique name for the extra devices. For example:

```
Dev.par &
Dev.par -b 2 -N laser &
```

These commands will create a device called `/dev/par` on LPT1, and a second device called `/dev/laser` on LPT2.

Output buffers

If you have the memory, you might find that specifying large output buffers significantly improves turnaround time when sending data to your printer. Here's an example of a parallel device created with a 30K output buffer:

```
Dev.par -O 30000 &
```

Serial devices

Hardware adapters

I/O addresses

The `Dev.ser` driver can support one or more serial ports. The hardware interface to the computer consists of a UART (*U*niversal *A*synchronous *R*eceiver/*T*ransmitter) for each serial port. The driver will support any or all of the 8250, 16450, or 16550 family of serial controllers.

Each UART uses eight consecutive addresses in the I/O address space of the computer. The `Dev.ser` driver is informed of the I/O address range for each UART by command-line arguments when it is started.

Hardware interrupt

Just as important as the I/O address is the hardware interrupt generated by each UART. Most microcomputers provide several hardware interrupt signals on the bus, labeled IRQ2 through IRQ15 (except for interrupts 0, 1, 8, 9, and 13, which are used internally by the system motherboard).

These interrupt signals are active-high TTL logic signals on ISA buses, which means that you can connect only *one* adapter card to any one interrupt signal. Microchannel architectures (MCA bus) don't have this limitation.

Serial adapter cards come in various configurations. Adapter cards with only one serial port typically offer only a limited set of choices for I/O address and hardware interrupt. The following table shows some commonly found combinations, but we recommend that you read the hardware documentation carefully to discover the choices available on a given manufacturer's adapter card:

Name	Address	Interrupt
COM1	3F8	IRQ4
COM2	2F8	IRQ3
COM3	3E8	varies
COM4	2E8	varies

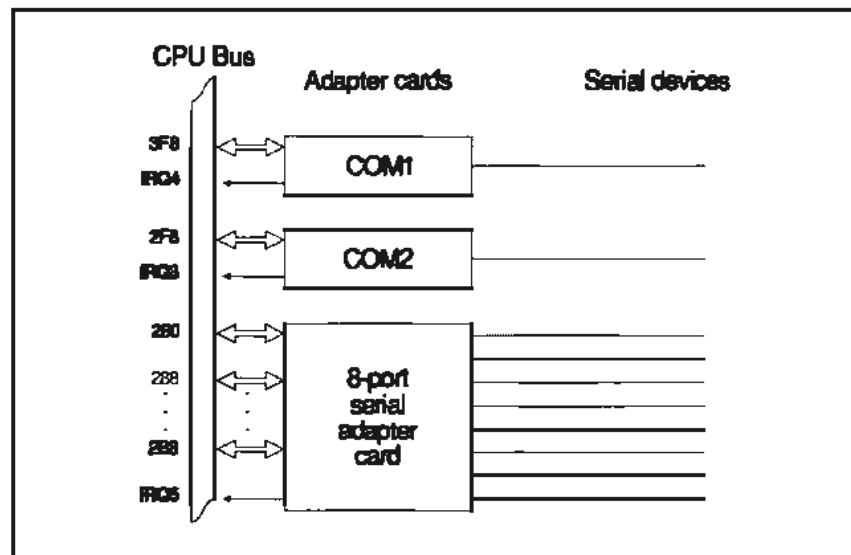
Multi-port serial adapters

You can usually configure multi-port serial adapters to respond to a wide range of I/O addresses. The adapters may also give you considerable flexibility in selecting hardware interrupts. A good choice for I/O addresses is often the 280-through-2BF range.

Because of the limited number of hardware interrupts available, these cards will often “OR” the interrupt lines from the individual UARTs into a single interrupt connected to the bus. QNX allows for many serial ports to share the same interrupt, since `Dev.scr` will check every UART that shares an interrupt.

Typical hardware installation

The following diagram shows a sophisticated configuration of serial adapter cards in a QNX system.



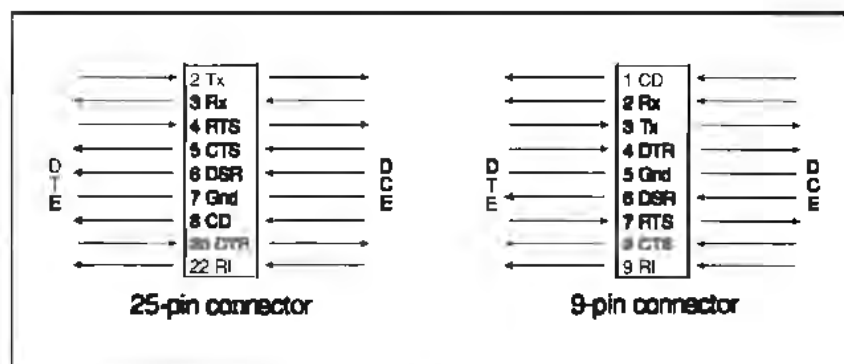
For proper operation, each serial channel must have a unique I/O address, and each adapter card must use a unique hardware interrupt.

The RS-232C serial protocol

Electrical interface

The RS-232C asynchronous communications protocol defines the electrical and physical interface between Data Terminal Equipment (DTE or *terminals*) and Data Communications Equipment (DCE or *modems*).

The following figure shows the cabling assignments of an RS-232 connection.

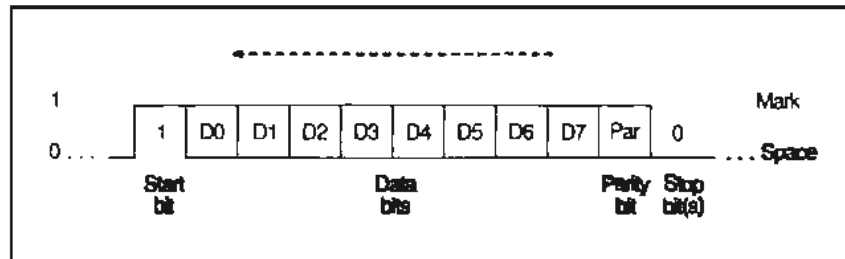


The host computer is usually configured as a DTE, acting as a terminal device. We assume that the computer will be connected to a modem.

The RS-232 signals have the following names:

Tx	transmit data
Rx	receive data
RTS	request to send
CTS	clear to send
DSR	data set ready
DTR	data terminal ready
CD	carrier detect
RI	ring indicator

Serial protocol Data is transmitted asynchronously using a bit protocol as shown below:



Normally, an RS-232 data line is in the **SPACE (0)** condition. A transmitted character consists of bits in the following order:

- 1) **START** bit (always 1)
- 2) 5 to 8 data bits (least-significant bit first)
- 3) parity bit (optional)
- 4) one or more **STOP (0)** bits.

The duration of each bit is defined by the *baud rate*, which indicates the number of bits per second that can be transmitted.

Parity is optional, but if used can be one of:

odd	sum of data bits plus parity bit is odd
even	sum of data bits plus parity bit is even
mark	always 1
space	always 0

Session control RS-232 uses the DTR and DSR lines to control communication sessions. The terminal raises DTR when it is powered up and available. Similarly, the modem raises DSR when it is powered up and available (but not necessarily *connected* to a remote modem). No communication is expected to occur unless both DTR and DSR are raised.

A terminal indicates that it no longer wishes to communicate by dropping the DTR line, which causes most modems to hang up the telephone line, thus releasing the connection.

A modem indicates that it has established a connection by raising CD. Some modems will also indicate that they have detected (but not yet answered) an incoming call by raising RI.

Flow control

The RTS and CTS lines control the flow of data between terminal and modem. The terminal raises RTS when it is capable of receiving data on the Rx line. Similarly, the modem raises CTS when it can accept data on the Tx line.

Configuring serial ports

You use the **stty** utility to set the four major parameters that define an RS-232 link.

Data bits

QNX supports four character sizes. You choose the size of data character with one of the following **stty** commands:

```
stty bits = 5  
stty bits = 6  
stty bits = 7  
stty bits = 8 (default)
```

This parameter defines how many bits following the start bit will be used to form the least-significant portion of a received character.

Stop bits

It's possible to transmit data that is followed by either one or two stop bits. Two stop bits are used only to slow down the overall transmission of data so that the remote end has a chance to keep up. Using **stty**, you specify one of these:

```
stty stopb=1 (default)  
stty stopb=2
```

Parity

To disable the transmission of parity bits, and suppress the checking (in hardware) of received parity bits, you specify:

```
stty par=none (default)
```

If parity is used, you specify one of the following values:

```
stty par=odd  
stty par=even  
stty par=mark  
stty par=space
```

Baud rate You can specify the baud rate with the `baud=number` option of the `stty` utility:

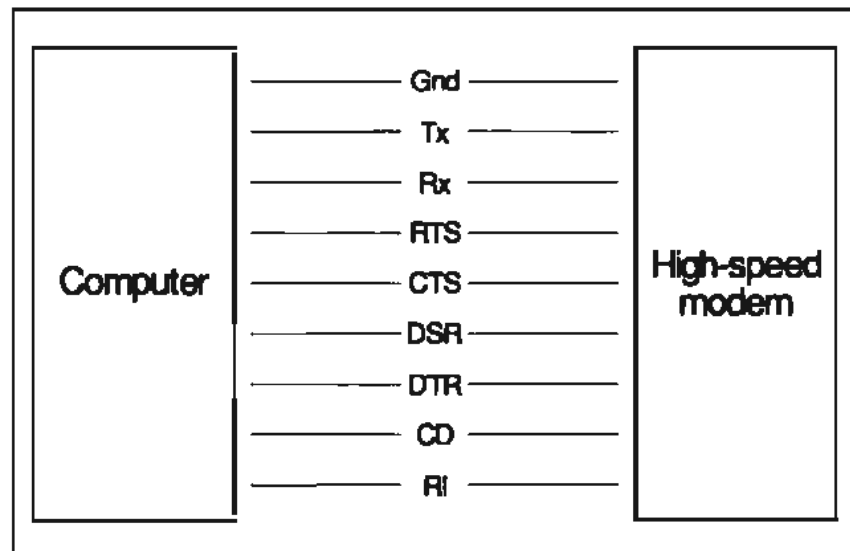
```
stty baud=number
```

QNX defaults to 9600 baud.

Connecting serial devices

High-speed ECC modems

High-speed, error-correcting modems are becoming very sophisticated—they work best when all hardware handshaking signals are respected. These modems often communicate with the host computer at a fixed high-speed baud rate (e.g. 38400 baud) and use the RTS/CTS handshaking lines to regulate the actual flow of data over the communications link. QNX is ideally suited for communicating with such modems. A straight-through cable is used to connect the modem to the computer.



Cabling assignments for linking a computer to a high-speed modem.

Using the `stty` utility, you should enable hardware handshaking in both directions and lock it on:

```
stty +ihflow +ohflow +lkhflow </dev/ser1
```

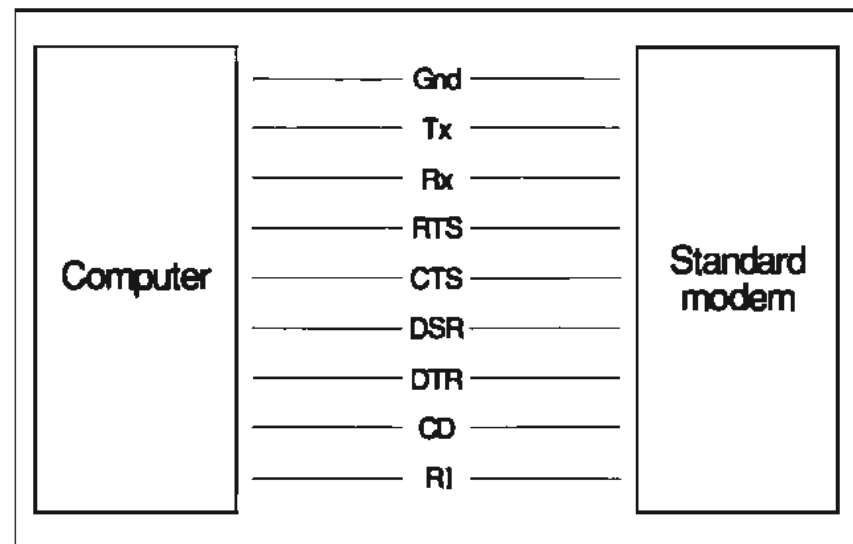
Alternately, you could enable and lock hardware handshaking when starting `Dev.ser`:

```
Dev.ser -f 3f8,4 &
```

Standard modems

Older modems are less sophisticated. Once a communications channel is established at one of the baud rates supported by that modem, all subsequent communication over the modem link takes place at exactly that baud rate. In other words, all transmitted data is simply passed through the modem link at the same rate that it's presented. Although these modems don't typically support hardware flow control, they often won't function properly unless all four flow control lines are connected. You don't need to use `stty`'s flow control options when using these modems.

A straight-through cable is used to connect the modem to the computer.

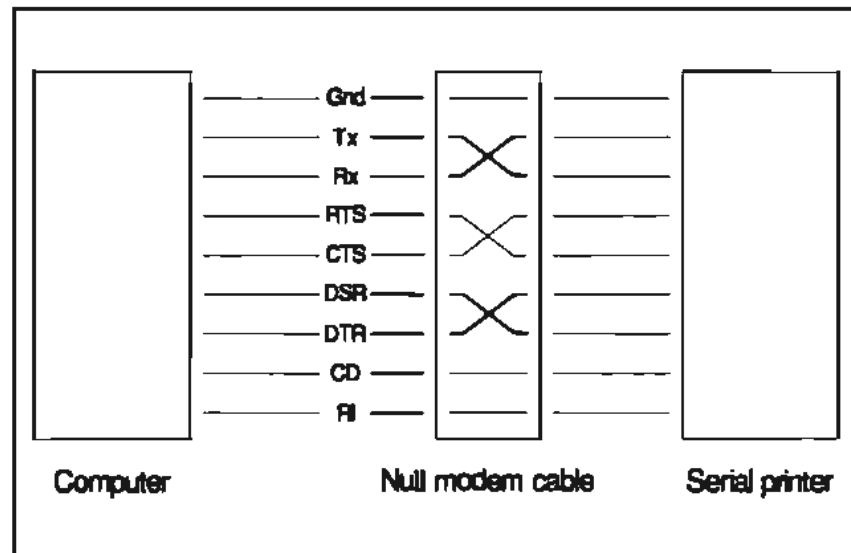


Cabling assignments for linking a computer to a standard modem.

Serial printers

Serial printers are usually *bidirectional* devices. Data flows from computer to printer as expected, but since printers can't keep up with the host computer, serial printers often use software flow control to regulate the flow of data. In other words, they transmit XON and XOFF characters back to the computer. Some printers use the hardware handshaking lines for this purpose, some support both forms of flow control.

To be safe, you should connect all nine signals, although printers that support only software flow control may function just as well with a three-wire cable (Rx, Tx, and Gnd). Also, since printers are usually configured as Data Terminal Equipment (DTE)—just like the host computer—you may need to use a null-modem cable.

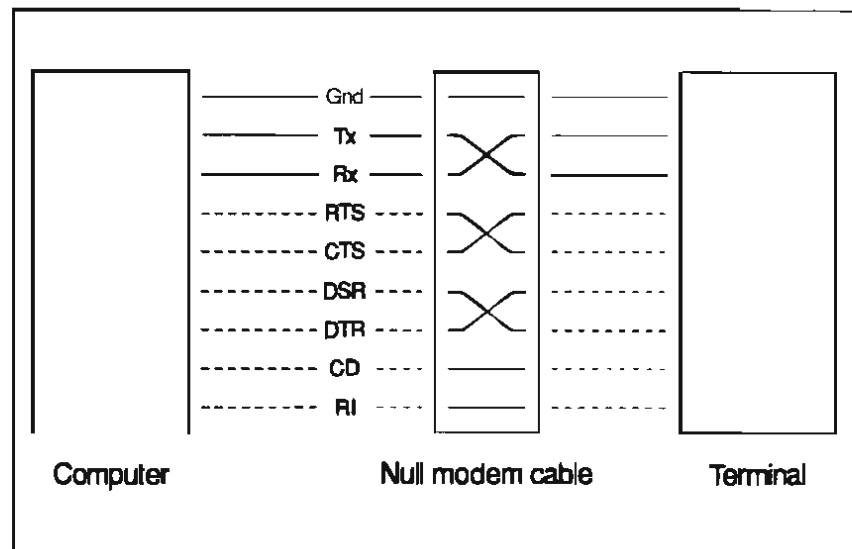


Cabling assignments for linking a computer to a serial printer.

If the printer uses:	you use:
software flow control	<code>stty +osflow </dev/ser1</code>
hardware flow control	<code>stty +ohflow </dev/ser1</code>
both software flow control and hardware flow control	both <code>stty</code> options

Terminals Terminals operate with or without flow control and at a fixed baud rate. Unlike printers, terminals can usually keep up with the host computer at a supported baud rate. Thus, a simple three-wire cable is usually sufficient, although you can still use a nine-wire cable.

Like the host computer, terminals are normally configured as DTE devices, so a null-modem cable is usually required.



Cabling assignments for linking a computer to a terminal.

If your terminal supports software flow control (e.g. VT100), it's a good idea to enable output software flow control and lock it on:

```
stty +osflow +lksflow </dev/ser1
```

Alternately, you could use the **-x** option when starting **Dev . ser**.

Configuring serial lines for terminals and users

QNX can be used as a full-function time-sharing system. Many users can be connected to some or all of the computers in a QNX network via hardwired terminals or through modem links to terminals at remote sites.

Assuming that the terminal/modem is properly configured, you'll also need to provide a mechanism for users to "log in" to the system.

Simple example

Assume a terminal, properly connected and configured, on the serial port `/dev/ser1`. The simplest way of allowing a user sitting at that terminal to log in is to use this command:

```
ontty /dev/ser1 login
```

The user will be able to log in and execute commands. However, once logged off (i.e. once the user terminates that shell session), the user won't be able to log back in.

Automated login

To automate this login process, you can use the `tinit` utility. This utility will watch a specified device, waiting for any key to be pressed on that terminal. It will then start a `login` automatically.

When the user has logged in and then logged off again, `tinit` will once again wait for another key to be pressed. To start logins when a data key is pressed on two serial devices called `/dev/ser1` and `/dev/ser2`, you can use the following command:

```
tinit -T /dev/ser1 /dev/ser2 &
```

Suppose instead you wanted logins to be automatically started without first waiting for the user to press a key. You would use the `-t` option:

```
tinit -t /dev/ser1 /dev/ser2 &
```

Launching custom applications

You don't always have to start up `login`. You can specify any program or even specify a different program on each device. In some environments, a "canned" application is always expected on a given terminal.

For more information, see the documentation for `tinit` in the *Utilities Reference*.

Modem access

It's fairly easy to configure hardwired terminals, since all possible parameters are known. Modems, on the other hand, present special problems since they can have different configurations at different times. For example, most modems will answer calls at one of several baud rates (300, 1200, 2400, etc). You can't always know in advance the baud rate of an incoming call.

A special utility called **modem** is provided to respond to modems. Used in conjunction with **tinit**, **modem** can provide excellent dial-up capabilities. A typical dial-up system using QNX might have several serial ports (**/dev/ser1**, **/dev/ser2**, etc.) and might use the following command to permit dial-up access through Hayes-compatible modems:

```
tinit -c modem -t /dev/ser1 /dev/ser2 &
```

To set up a modem that uses a locked baud rate, you could use a command similar to the following:

```
tinit -c "modem -b 38400 -L" -t /dev/ser1 &
```

The **tinit** utility will automatically launch **modem** on each of the serial lines. When communication is established, **modem** will:

- answer the phone
- determine and set the proper baud
- *exec* into **login**

When the user either logs off or hangs up, **tinit** will once again launch a new **modem**, which will wait for another call. For more information, see **modem** in the *Utilities Reference*.

Note that you don't have to dedicate a serial line for incoming calls. The **qtalk** terminal emulator checks if the line is free, and if so, will connect underneath **modem**, allowing you to place an outgoing call. When the **qtalk** session is complete, **modem** will again be in control, waiting for incoming calls to arrive. For more information, see **qtalk** in the *Utilities Reference*.

The **terminfo** database

Full-screen programs in QNX have access to a database of terminal capabilities known as the **terminfo** database, which is implemented as a collection of files in the **/usr/lib/terminfo** directory.

A separate file exists for each terminal supported in QNX. Since these files use an industry-standard format, they can be shared between different UNIX systems. Your QNX system will have been shipped with several of these files, with names such as:

/usr/lib/terminfo/q/qnx	(the QNX console)
/usr/lib/terminfo/v/vt100	(a VT100 terminal)

A QNX program will access the appropriate file in the database, provided that the **TERM** environment variable has been set.

If you don't have a **terminfo** file for your type of terminal, there are several things you can try:

- Access the Quantum Update Service (QUICS) and download a **terminfo** file (if it exists).
- Access another UNIX system and look for the appropriate **terminfo** file.
- Use **infocmp** and **tic** to make your own **terminfo** file based on the reference manual for your terminal.

Defining terminal type

Full-screen programs use the **TERM** environment variable to determine which type of terminal is currently used. They also automatically adjust their behavior to properly display data and interpret keys from that type of terminal.

This all works as expected, provided that the **TERM** environment variable is properly defined.

For hardwired terminals running canned applications, you can preset this environment variable when the application is launched. For example:

```
TERM=vt100 ontty /dev/ser1 custom_application
```

However, users logging in via **login** through **tinit** represent a special problem. If the type of terminal is known, then **tinit** can be told to define this environment variable before launching **login**:

```
tinit -c login -t /dev/ser1 TERM=vt100 &
```

Unless you restrict dial-up access to only a particular type of terminal, modem users will need to run a special program to query the user for his terminal type. In QNX, such a program is provided—`termdef`. This program will display all terminal types supported on this system and will ask the user to select one. Once a terminal type has been selected, `termdef` will set the `TERM` environment variable appropriately, then `exec` into `login`.

```
tinit -c "modem -c termdef" -t /dev/ser1 &
```

Troubleshooting serial device problems

The following table describes actions you can take if you encounter some of the common problems involved in connecting serial devices.

Problem	Probable Cause	Remedy
Data characters are unrecognizable	Wrong baud rate or parity	Use <code>stty</code> to set correct baud rate and/or parity
Some characters are fine, some aren't	Wrong parity	Try different parity using <code>stty</code>
No data can be sent or received	Wrong cable	Check cables; use null-modem if necessary
	Wrong I/O ports	Check hardware settings and verify correct parameters to <code>Dev . ser</code>
	Interrupt conflict	Change interrupt on adapter card
Data is received and transmitted only when another serial port is in use	Interrupt conflict	Check hardware interrupts and <code>Dev . ser</code> startup parameters; make sure that two serial adapters are <i>not</i> using the same IRQ

(cont'd on next page)

Serial devices

Problem	Probable Cause	Remedy
Data is lost occasionally	Flow control supported, but not enabled	Determine the type of flow control supported by the device, and enable with <code>stty (ihflow, ohflow, isflow, and osflow)</code>
	Flow control not supported	Reduce baud rates and/or increase stop bits; if only received data is lost, specify larger input buffer to <code>Dev.ser (-I option)</code>
	Cabling problems	Make sure cable is well grounded and not too long; also, verify that all RS-232 wires in the cable are connected

Chapter 8

Setting up a Second Filesystem

In this Chapter...

Partitioning the pathname space

Setting up a DOS filesystem

Introduction

A filesystem rooted at slash (/) may be composed of one or more *physical filesystems* grafted together. A physical filesystem is a separate disk or partition. One of the physical filesystems is typically assigned to be the root (/), while the other filesystems are mounted as subdirectories. These subdirectories may exist either on disk—as in the case of symbolic links—or in an independent space called the prefix tree. The prefix tree maps pathnames to I/O managers, which in turn map to disks and devices. In all the examples in this chapter, the prefix tree is used to partition the pathname space.

For more information on the prefix tree, see the chapter on “I/O Namespace” in *System Architecture*.

Partitioning the pathname space

The following examples should help clarify how the pathname space is partitioned. We'll look at the following configurations:

- a hard disk and a floppy
- two hard disks
- two or more QNX partitions
- local and remote hard disks



In these examples, we assume that **Fsys** as well as the appropriate drivers are running, and that the **mount -p** command has been done on your hard disk.

A hard disk and a floppy

The hard disk is mounted as slash (/) and forms the root of the filesystem. The floppy is mounted as **/fd0**:

```
mount /dev/hd0t77 /  
mount /dev/fd0 /fd0
```

Any reference to a pathname starting with `/fd0` will be directed to a QNX filesystem on the floppy. For example:

```
ls -aR /fd0    show all files on the floppy
```

Two hard disks (same node)

The first hard disk is mounted as slash (`/`) and forms the root of the filesystem. The second hard disk is mounted as `/home2`:

```
mount /dev/hd0t77 /
mount /dev/hd1t77 /home2
```

Any reference to a pathname starting with `/home2` will be directed to a QNX filesystem on the second hard drive. For example:

```
ls -aR /home2    show all files on second hard drive
```

Two or more QNX partitions

You can have three QNX partitions on a single hard drive. The primary partition should be type 77 (see the `fdisk` utility in the *Utilities Reference*). A second partition should be assigned type 78, and a third should be assigned type 79. For example:

```
mount /dev/hd0t77 /
mount /dev/hd0t78 /home2
```

Any reference to a pathname starting with `/home2` will be directed to a QNX filesystem on the second partition. For example:

```
ls -aR /home2    show all files on second partition
```

Local and remote hard disks

In a network, you might have disks with QNX filesystems on more than one machine. You may configure these filesystems to be:

- independent
- primary/secondary
- linked independent



In the following examples, you'll notice that local filesystems are associated with block special files (e.g. `/dev/hd0t77`) while remote filesystems are associated with pathname prefix mappings. This prefix mapping redirects requests to a remote filesystem that will be associated with a remote block special file.

Independent In this configuration, you treat each machine as an independent, self-contained filesystem. To access a file on a remote machine, you would precede a pathname with the remote machine's node number. For example:

<code>/etc/motd</code>	file on local filesystem
<code>//10/etc/motd</code>	file on remote filesystem (node 10)

The ability to specify a filesystem for a particular node will always work and is the most general mechanism for accessing remote files.

Primary/secondary In this configuration, you treat one filesystem as the primary and you mount the second filesystem as a subdirectory under the primary. For example, assume node 1 has the primary and node 2 has the secondary mounted as `/home2`. Node 1 boots from hard disk and has its root set to its local disk by a `mount` utility built into the operating system image (with `fsys` and a driver). Its system initialization file would invoke the `prefix` utility to mount the remote filesystem as follows:

```
prefix -A /home2=//2/home2
```

Node 2 boots over the network from node 1 and has its filesystem root set to `/=//1/` using the `-r` option of the `sinit` utility built into the operating system. Its system initialization file would invoke the `mount` utility to mount the local filesystem as follows:

```
mount /dev/hd0t77 /home2
```

The Filesystem Manager (`fsys`) and its driver may be built into the image, but they are more likely started from the system initialization file. In other words, node 2 boots like a simple diskless workstation, then starts its filesystem after booting.

Both nodes 1 and 2 will access the filesystem on node 1 as `/` and the filesystem on node 2 as `/home2`.

Linked independent In this configuration, you treat each machine as an independent self-contained filesystem, but you link portions of them together via the `prefix` utility. For example, assume that the filesystem on node 1 has a `/home1` directory while the filesystem on node 2 has a `/home2` directory. You could map each home directory into the other filesystem's space as follows:

On node 1:

```
prefix -A /home2=//2/home2
```

On node 2:

```
prefix -A /home1=//1/home1
```

Other than this link, each filesystem is self-contained with its own copies of `/bin` etc. The advantage is greater redundancy: if one department uses node 2, and node 1 goes down, the department using node 2 can continue to work (except that the files under `/home1` won't be available).



Instead of using prefixes to partition the pathname space, you may want to use symbolic links. For more information, see Chapter 5, *The Filesystem Manager*, in *System Architecture*.

Setting up a DOS filesystem

The **Dosfsys** filesystem manager provides totally transparent access to DOS files and directories that reside on a DOS disk. **Dosfsys** can support up to eight drives.

You can create files on DOS disks, read them, write to them, and delete them with most QNX programs. You can also create, read, and delete directories on DOS disks with standard QNX utilities, such as **mkdir**, **ls**, and **rmdir**. Most standard QNX utilities will work with DOS files, provided that the DOS file structure allows for the functionality required by the utility.

Your own programs will also be able to process DOS files just as they process QNX files, by using the standard I/O functions such as *open()*, *read()*, *write()*, *close()*, *seek()*, etc. When you read DOS directories, they'll be presented to you in QNX format.

Invocation modes

Dosfsys has four invocation modes:

```
Dosfsys [-S|-s] [-e] [-m] [-t] [dos_drive=qnx_drive[,R]] &
```

```
Dosfsys -i [-n node] [dos_drive_path]...
```

```
Dosfsys -o [-n node]
```

```
Dosfsys -x [-n node]
```

The **-i** option lets you get information about the currently adopted DOS drives. The **-o** option lets you see the names of any files currently open on each device. The **-x** option terminates the **Dosfsys** server.

If you don't specify **-i**, **-o**, or **-x**, **Dosfsys** will start up and try to adopt the specified drives.

To start or terminate the **Dosfsys** server, you must be logged in as the superuser (**root**).

Starting Dosfsys

When you start **Dosfsys**, it:

- opens the specified drive(s)
- adopts the root DOS name (**/dos**)
- registers the name **qnx/dosfsys** with the local Process Manager

If no options have been specified, or if either **-S** or **-s** has been specified, **Dosfsys** scans the **/dev** directory for valid DOS drives to adopt. Unless otherwise specified on the command line, **Dosfsys** will adopt up to eight drives. DOS primary partitions (**/dev/hd0t1**, **/dev/hd0t4**, **/dev/hd0t6**, ...) and extended partitions (**/dev/hd0t1.1**, **/dev/hd0t1.2**, ...) are mounted starting at drive C (**/dos/c**). Floppy drives are mounted as follows:

```
/dev/fd0 → /dos/a
/dev/fd1 → /dos/b
```

Dosfsys name adoption

Dosfsys can adopt up to eight drives. As mentioned above, **Dosfsys** will adopt the name **/dos** as a system prefix. It will also manage each specific drive's name as **/dos/a**, **/dos/b**, etc. These names aren't registered in the system prefix tree but are kept internally by **Dosfsys**. This will be transparent to the user except for the fact that the user won't be able to create files or directories at the **/dos** root.

DOS devices

A DOS device could be one of the following:

- a DOS partition on a hard disk
- a floppy diskette
- an image of a DOS partition or diskette

To create an image of a DOS diskette or DOS partition, you use the QNX `cp` utility. For example, to copy an image of a DOS floppy in your floppy drive 0, you could use the following:

```
cp /dev/fd0 /usr/qnx/dosa
```

and then invoke `Dosfsys` as follows:

```
Dosfsys a=/usr/qnx/dosa &
```

The same could be done with a hard disk partition. `Dosfsys` will handle these images just as it would the actual device.

For all *non-removable* devices, `Dosfsys` immediately reads the DOS boot parameter block (BPB) as well as part of the file allocation table (FAT) at startup. For *removable* devices, the BPB and the FAT are read only when the drive is being accessed.

When `Dosfsys` has a *non-removable* device open, the device is locked for WRITE so no other process can write to this device without going through `Dosfsys`. Removable devices are kept open and locked only during accesses (e.g. during reading or writing to the disk). Note that unless you specify the `R` option, all drives have READ/WRITE access.

DOS version support

The `Dosfsys` manager supports all DOS partitions formatted with DOS 2.1 or later, including standard primary DOS partitions, DOS large partitions (DOS 4.0 and DOS 5.0 >32M), and DOS extended partitions (type 5). Hard disks, 5¼" floppies, and 3½" floppies are supported.

DOS partition types

These are the standard DOS hard disk partition types:

Partition type:	Description:
1	DOS primary partition (12-bit FAT)
4	DOS primary partition (16-bit FAT; ≤ 32M)
5	DOS extended partition (DOS 3.3 or later)
6	DOS primary partition (DOS 4.0 or later; >32M)

DOS text files

DOS uses a structure for text files that's different from the one used in QNX (by text files we mean line-oriented files containing lines of ASCII text separated by line separator sequences). In DOS, each line of a text file is terminated with a carriage return sequence (CR/LF), while in QNX each line is terminated by a line-feed character (LF). The `Dosfsys` manager doesn't translate these files. All files are treated "as is." Therefore, you may need to use the QNX `textto` utility to convert your text files before copying them to or from QNX and DOS disks.

Note also that the text files created by some DOS programs may contain a SUB character (^Z) as the last character of the file. This is also treated as is.

DOS binary files

Since `Dosfsys` doesn't translate the contents of files, binary files may be copied to or from the QNX/DOS partitions as is.

QNX-to-DOS character and name mapping

In DOS, you can't include any of these characters in a filename:

/ \ [] : * | + = ; , ?

Nor can you use these filenames:

CLOCK\$	CON	PRN
NUL	AUX	COM1
COM2	COM3	COM4
LPT1	LPT2	LPT3

If you attempt to create a file that contains one of the invalid DOS characters or that has an invalid filename, you'll be denied access.

Since all DOS filenames and filename characters are allowed under QNX, no validation is required on these filenames.

DOS also maps all alphabetical characters to uppercase, so **Dosfsys** maps these characters to uppercase when creating a DOS filename; it maps a filename to lowercase when returning the filename to a QNX application.

Translating QNX filenames

If you specify **-t** when starting **Dosfsys**, the server will translate and/or truncate any QNX filenames that don't conform to the DOS naming conventions. The following table describes how names are translated:

If the QNX filename:	Dosfsys will:
starts with a dot: .	change the . to a \$
contains multiple dots	change all but the last dot to a \$
has a prefix longer than eight characters	truncate the prefix to eight characters
has a suffix longer than three characters	truncate the suffix to three characters

Here are some examples:

QNX filename:	DOS filename:
.profile	\$profile
a.b.c.d	\$a\$b\$c.d
longfilename	longfile
longfilename.extension	longfile.ext
a..b..c.def.g.h	a\$\$b\$\$c\$.h

DOS volume labels

DOS uses the concept of a volume label, which is an actual directory entry at the root of a DOS filesystem. To distinguish between the volume label and an actual DOS file, **Dosfsys** places an equal sign (=) as the first character of the volume label name. **Dosfsys** treats this directory entry as a zero-length, read-only file whose permissions cannot be changed.

DOS/QNX permission mapping

DOS doesn't support all of the permission bits that QNX does. The DOS attribute bits are as follows:

READ_ONLY
HIDDEN
SYSTEM
VOLUME_LABEL
DIRECTORY
ARCHIVE

Attribute-bit translations

Dosfsys uses the following mapping logic to handle the QNX-to-DOS attribute-bit translations:

- if the entry is a directory, set the DOS DIRECTORY file bit
- if the entry is a file, and if all of the QNX WRITE bits are off, set the DOS READ_ONLY bit

Permission-bit translations

The following mapping logic is used to handle the DOS-to-QNX permission-bit translations:

- set the QNX READ permission bits for user, group, and other
- if the entry isn't a volume label, and if the entry isn't read only, set the QNX WRITE permission bits for user, group, and other
- if the entry is a directory, set the QNX DIRECTORY and EXECUTE bits for user, group, and other
- if the entry is a file, set the QNX REGULAR FILE bit



If a file is written to, the DOS ARCHIVE bit will be set.

If **Dosfsys** is started with the **-e** option, all DOS files and directories will have the QNX EXECUTE bits set for group/owner/other.

File ownership

Although the DOS filesystem doesn't support user IDs and group IDs, **Dosfsys** will not return an error code if an attempt is made to change the group ID or user ID with the **chown** utility or **chown()** library function. An error isn't returned because a number of utilities make use of the **chown()** library function, which could result in many error messages being displayed.

All files under **Dosfsys** are owned by the superuser (uid=1, group=0) with access to all.

Terminating Dosfsys

The **-x** option terminates the **Dosfsys** server. If you specify **-x**, no new **open()** requests will be accepted and the server will terminate once all active files (i.e. files that are still open) are closed.

Error codes returned by Dosfsys

If a request made to **Dosfsys** isn't supported, the EOPNOTSUPP error code will be returned to the application making the request. Examples of requests not supported by **Dosfsys** include:

- LINK
- BLOCK_READ
- BLOCK_WRITE
- MOUNT_PARTITION
- MOUNT_RAMDISK
- PIPE
- DISK_GET_ENTRY
- RECORD LOCKING
- SYMBOLIC LINK

If **Dosfsys** detects a corrupt filesystem, it will return EBADFSYS, at which point you may wish to run the **CHKDSK** utility under DOS to correct the problem.

The DOS filesystem structure is such that the root directory's size is fixed at format time and cannot be resized. If it does become full, an error will be returned (ENOSPC).

Chapter 9

Advanced Network Configuration

In this Chapter...

Introduction

Multiple boot servers

Multiple network links

Configuring multiple network links

Introduction

In this chapter, you'll learn about setting up multiple boot servers; you'll also learn about configuring multiple network links in a single computer. Before proceeding with any of the operations described here, you must first install a single QNX network that includes a boot server (see Chapter 5, Network Installation).

Multiple boot servers

You can set up your network with multiple boot servers to provide fault-tolerance in case your primary boot server becomes inoperative, and also to distribute the boot resources to avoid boot bottlenecks. Before doing so, however, you must understand the difference between the two methods a network card may use to alert a boot server that it wishes to boot.

In the first method, the network card sends a boot request to a specific boot server; in the second method, the card *broadcasts* a boot request to all nodes, expecting that a boot server will respond to the request. The Quantum Arcnet card, which has nonvolatile RAM that can store the node ID of the server, uses the specific boot method. Most other network cards don't have the means to store the physical node ID of the server and must use the broadcast method.

Broadcast booting with a QNX ROM

In order to boot QNX from a network card other than Quantum's Arcnet (e.g. Ethernet, Token Ring), you must install a QNX boot ROM in the ROM socket on the card as described in the card's documentation.

When the QNX boot ROM attempts to boot, it broadcasts a boot request to a server it symbolically refers to as server "A" (primary server); if it gets no response, the ROM will start broadcasting boot requests to server "B" (secondary server), and finally to server "C" (tertiary server).

With broadcast booting, *netboot* needs to know whether it should act as primary, secondary, or tertiary server with respect to each node. The utility finds out by opening the `/etc/config/netboot`

file, in which each line specifies a logical node ID, a filename identifying the operating system image to boot, and a server mode (optional):

```
logical_node_ID  f=buildfile|F=imagefile  [server_mode]
```

If *server_mode* isn't specified, **netboot** will act as a primary boot server for that node. The possible values for *server_mode* are:

- A primary server (default)
- B secondary server
- C tertiary server

The following example indicates to **netboot** that it is to be the primary server for all nodes except for node 4, where it is to be a secondary server:

```
4  f=build/ws.arcnet  B
*  f=build/ws.arcnet  A
```

Running the **netboot** utility

Since broadcasts are potentially seen by all nodes on the network, you can run **netboot** on several nodes, creating multiple boot servers. This lets you associate specific groups of nodes with different boot servers.

Although every **netboot** process usually receives each broadcast boot request, it responds only in accordance with its own unique `/etc/config/netboot` file, which defines the relationship to each requesting node (e.g. "I am node 5's secondary boot server").

No two boot servers can have the same *server_mode* entry for any given logical node ID. If this rule isn't followed, every boot server with the same *server_mode* entry for a given logical node ID will respond to that node's boot request.

Let's now look at a 15-node Ethernet network, with nodes 1 to 9 belonging to the "tech" department and nodes 10 to 15 belonging to the "marketing" department. If you wanted to make node 1 the primary server for tech, and node 10 the primary server for marketing, you would run **netboot** on both nodes, each with its own configuration file:

On node 1 (`//1/etc/config/netboot`):

```
2 f=build/ws.ether8003 A
3 f=build/ws.ether8003 A
4 f=build/ws.ether8003 A
5 f=build/ws.ether8003 A
6 f=build/ws.ether8003 A
7 f=build/ws.ether8003 A
8 f=build/ws.ether8003 A
9 f=build/ws.ether8003 A
* f=build/ws.ether8003 B
```

On node 10 (`//10/etc/config/netboot`):

```
2 f=build/ws.ether8003 B
3 f=build/ws.ether8003 B
4 f=build/ws.ether8003 B
5 f=build/ws.ether8003 B
6 f=build/ws.ether8003 B
7 f=build/ws.ether8003 B
8 f=build/ws.ether8003 B
9 f=build/ws.ether8003 B
* f=build/ws.ether8003 A
```

In this case, you've also made node 10 the secondary boot server for tech, and node 1 the secondary server for marketing.

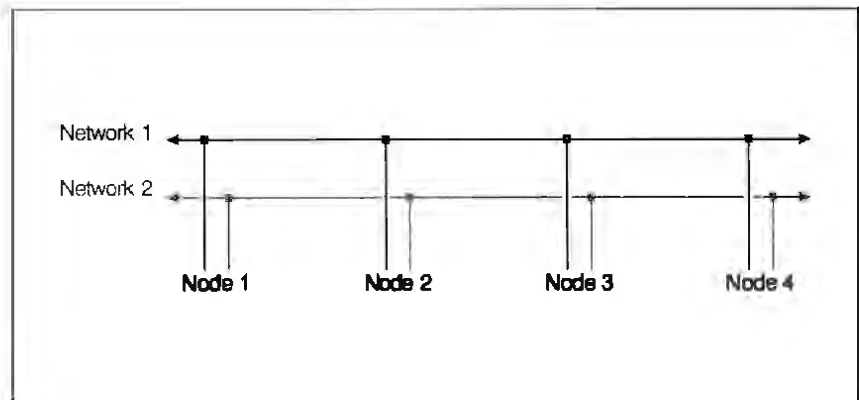
Multiple network links

Benefits

As described in *System Architecture*, having more than one network link in a computer provides fault-tolerance and increased throughput, as well as added connectivity (i.e. you can access nodes not directly connected to your network by remotely executing on nodes that can access both your network and other networks).

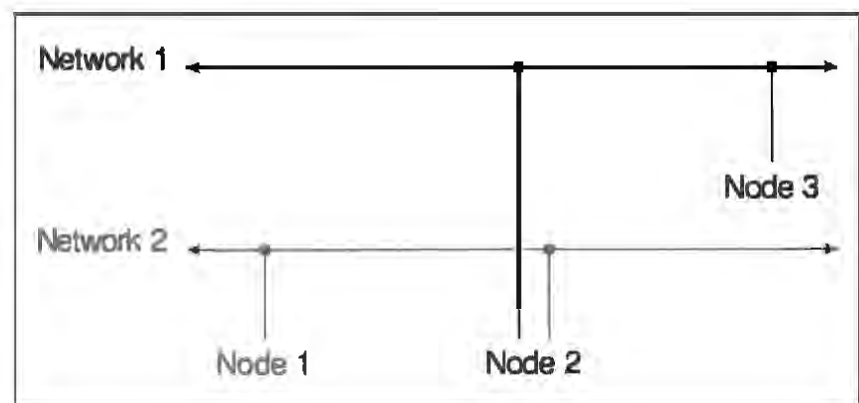
Communicating across networks

In QNX, a node can communicate with multiple networks simultaneously. However, the node can communicate only with the nodes it is directly connected to. Consider the following diagram where four computers are linked via two networks:



In this fault-tolerant network, each of the four nodes can communicate directly with any other node through either network 1 or network 2.

Compare this to the following diagram, where nodes 1 and 3 are connected to separate networks, while node 2 is common to both networks. Nodes 1 and 3 can't directly communicate with each other, but they can both access node 2.



With this configuration, the following command executed on node 1 would fail because node 1 cannot talk directly to node 3:

```
sin -n 3
```

However, since QNX allows transparent remote process execution, the command could be successfully executed from node 1 using either of these methods:

```
//2 sin -n 3
```

OR

```
//2 sh
sin -n 3
exit
```

The second method starts a remote session on node 2. With this new session, all subsequent commands will by default be executed on node 2 until the `exit` command is executed.

Configuring multiple network links

To set up multiple network links in a single computer, you must first choose a unique logical network ID for every network you'll be configuring. You'll then need to:

- add network cards
- start appropriate network drivers
- edit the `/etc/config/netmap` and `/etc/config/netboot` files

In this section, we'll discuss these tasks and then look at examples of typical installations using multiple network links.

Adding network cards

Quantum has provided technical notes for installing network cards. These notes, which you'll find in `/etc/readme/technotes`, have filenames beginning with `Net` (e.g. `Net.ether8003`). You should read the technical note for the type of card you're using before installing the card.

You should install each network card according to the instructions provided with the card. Remember to note the physical node ID of the card. To use the network card, you'll need to start an additional network driver.



The network card whose ROM resides at the higher memory address is the last one found during the “Power On Self Test” sequence performed by the BIOS. This card will *usually* be the one your computer uses to boot with over the network.

Starting network drivers

A node can be a member of one or more networks; for every network link, you must start a network driver. Where you start a network driver depends on whether or not it's an *essential* driver.

Essential drivers

An *essential* driver is a network driver in the boot image that enables a workstation to communicate with its boot server, allowing the workstation to access its `sysinit.node` file on the server's hard disk. Only workstations require essential drivers.

A workstation's boot image or build file (as specified in the `/etc/config/netboot` file) must always include the appropriate drivers to talk to the boot server. For example, the default `/boot/build/ws.arcnet` build file includes the Arcnet network driver (`Net.arcnet`) as part of the image transmitted to the booting workstations.

If a boot server has more than one entry in its `/etc/config/netboot` file for a particular workstation—thus indicating that the workstation's `sysinit.node` file can be transmitted over multiple network links—that workstation's image should have a network driver for every network link the boot server may try to use. These network drivers are considered essential and thus should be added to the workstation's build file. Instead of modifying a default build file, you should make a copy with a meaningful name—such as `ws.arc_eth8003` for a build file that will contain both Arcnet and Ethernet 8003 network drivers—then modify the new file. Note that the `-d` option to `Net` is required if more than two network drivers are used. We also recommend that you see `netmap` in the *Utilities Reference* to learn how to mask nodes.

Additional drivers

Additional drivers are those *not* required for a node to boot. Like other programs, you usually add these drivers to the `sysinit.node` file. For example, a boot server boots from disk and usually has its `sysinit.node` file modified to include the Network Manager and appropriate drivers for the networks the server will communicate with.

Specifying logical network IDs

By default, network drivers connect to logical network ID 1, simplifying installation of a single network. However, when connecting multiple networks, you have to specify the logical network ID to every network driver that's not on network 1. To do this, you use the **-l (el)** option of the network driver.

For example, the following three commands in the **sysinit.node** file of a boot server would start the Ethernet drivers for two networks:

Net &	Network Manager
Net.ether8003 -p 220 &	network driver for logical network 1 (default)
Net.ether8003 -p 340 -l 2 &	network driver for logical network 2

Note that when you place two similar cards in the same machine you cannot use the completely automatic determination of interrupt, I/O ports, etc. that network drivers usually provide. Instead, you must specify some of this information using command-line options. You must also configure the cards such that they have no hardware conflicts.



If you have to start more than two network drivers, you'll need to use the **-d** option to **Net** to specify the number of drivers to be used (e.g. **-d 3** for three network drivers). Because **Net** is started in the build file for workstations and from the **sysinit.node** file for the boot server, you'll need to modify both files to add the **-d** option to the Network Manager.

Editing the netmap file

The `/etc/config/netmap` file is the default network and node ID mapping file used by the `netboot` and `netmap` utilities. This file associates the physical node ID, logical node ID, and logical network ID of each node.

By default, this file is set up for a single-network Arcnet installation, where the physical node ID is the same as the logical node ID. It uses a one-to-one mapping to make the logical node ID the same as the physical node ID:

one-to-one

If you've already set up a network where the logical and physical IDs differ, you'll have made changes to this file, specifying the physical node IDs for your nodes. For example, you may have set up an Ethernet network:

```
1 1 0000C0 7B7113
2 1 0000C0 164C2D
```

For a single network, the logical network ID is 1 by default. But when you're dealing with multiple networks, you must specify every computer's logical node and logical network ID as well as its physical node ID. For more information, see the "Examples" section below. To learn about masking, unmasking, and deleting nodes, see `netmap` in the *Utilities Reference*.

Editing the netboot file

When a workstation sends a boot request to the boot server, the Network Manager receiving the request passes the following information about the booting node to the `netboot` utility:

- the node's physical node ID (determined by the hardware)
- the driver's logical network ID (a parameter specified to the driver when it was started)

With this information, `netboot` uses the mapping file (default is `/etc/config/netmap`) to determine the logical node ID of the booting machine. The `netboot` utility then uses the `/etc/config/netboot` file to determine what image to transmit to the node that sent the boot request.

The `/etc/config/netboot` file specifies which image or build file is to be used to boot each workstation. You may need to edit the file if different networks are to be supported by the same boot server.

For example, this entry:

```
* f=build/ws.arcnet
```

uses the same build file for all nodes. In the case of a mixed network, you need to specify different build files for some of your nodes.

For example, let's say you had an eight-node Arcnet network that worked with the default netboot entry, and you then added three Ethernet-only nodes. For these nodes to boot from the same server (assuming the server also has an Ethernet card and driver), you would modify your `/etc/config/netboot` file to be:

```
9 f=build/ws.ether8003
10 f=build/ws.ether8003
11 f=build/ws.ether8003
* f=build/ws.arcnet
```

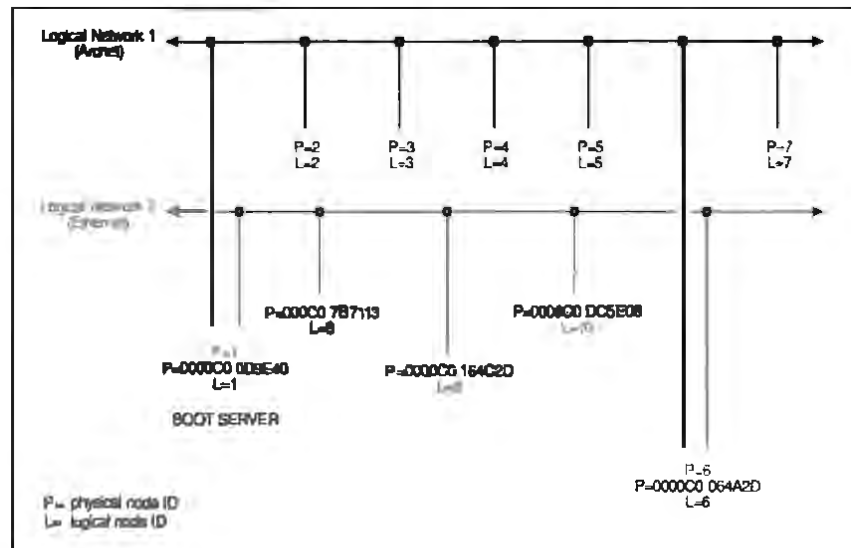
Examples

Now that we've discussed what's involved in setting up multiple network links, let's look at how to:

- add several Ethernet nodes to an Arcnet network
- set up a fault-tolerant Ethernet network
- set up a private network link

Adding several Ethernet nodes to an Arcnet network

Let's assume the following network layout:



As you'll recall, the format of the `/etc/config/netmap` file is as follows:

logical_node_ID logical_network_ID physical_node_ID

For this example, the file would contain the following entries:

```
1      1      1
1      2      0000C0 0D9E40
2      1      2
3      1      3
4      1      4
5      1      5
6      1      6
6      2      0000C0 064A2D
7      1      7
8      2      0000C0 7B7113
9      2      0000C0 164C2D
10     2      0000C0 DC5E08
```

Note that the logical node IDs are unique across both interconnected networks.

To boot workstations, the **netboot** utility requires that each node have an OS image or build file associated with it. Workstations booting on the Arcnet network don't use the same build files as workstations booting on Ethernet. Consequently, the **/etc/config/netboot** file would contain the following entries:

```
2    f=build/ws.arcnet
3    f=build/ws.arcnet
4    f=build/ws.arcnet
5    f=build/ws.arcnet
6    f=build/ws.arc_ether8003
7    f=build/ws.arcnet
8    f=build/ws.ether8003
9    f=build/ws.ether8003
10   f=build/ws.ether8003
```

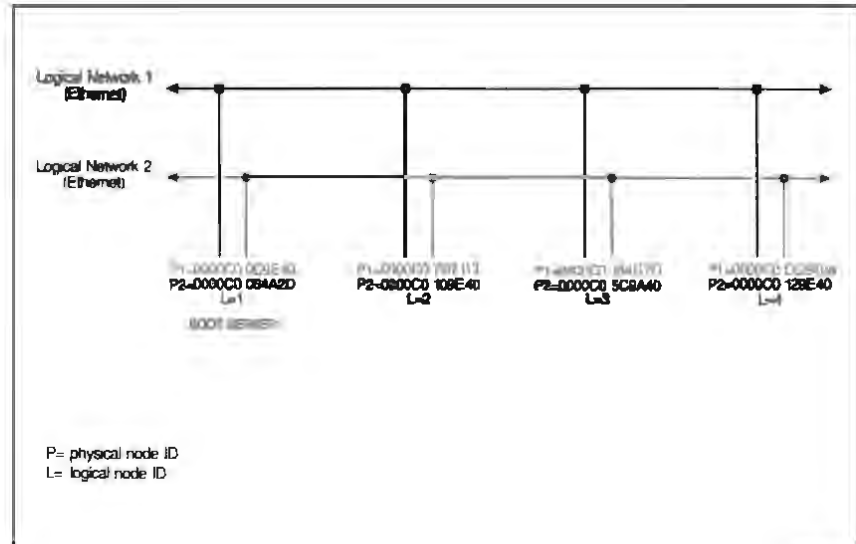
Node 1 has no entry since it boots from a hard disk. However, its **sysinit.node** file must start both an Arcnet driver and an Ethernet driver.

Note that you could configure node 6 to boot from either Arcnet or Ethernet. (Which network the node boots from depends on which network card is found last during the ROM scan at powerup—the highest addressed card is always found last). In this example, we've opted to configure node 6 to boot over Arcnet. Since node 6 will be on both Arcnet and Ethernet, you should create a special build file that will contain both **Net .arcnet** and **Net .ether8003** (e.g. **cp ws.arcnet ws.arc_ether8003**), and then add the Ethernet driver to the file. This will ensure that the boot server can transmit the **sysinit.node** file on either network.

The default build file **ws.arcnet** starts the Network Manager, which supports up to two drivers by default. If further network drivers are required, you'll have to add the **-d** option to **Net**.

Setting up a
fault-tolerant Ethernet
network

Let's assume the following network layout:



The corresponding `/etc/config/netmap` file would contain the following entries:

```
1 1 0000C0 0D9E40
1 2 0000C0 064A2D
2 1 0000C0 7B7113
2 2 0000C0 109E40
3 1 0000C0 164C2D
3 2 0000C0 5C9A40
4 1 0000C0 DC5E08
4 2 0000C0 129E40
```

The corresponding `/etc/config/netboot` file could contain the following entry:

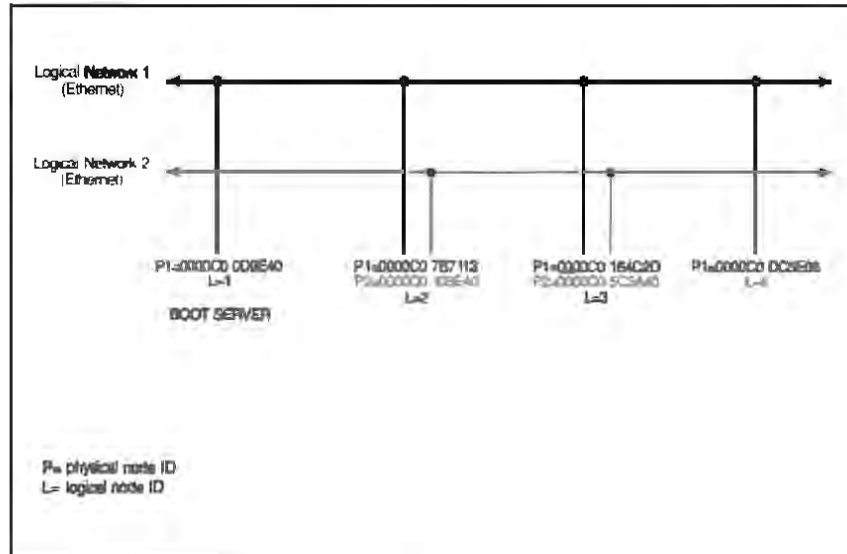
```
* f=build/ws.eth8003x2
```

Since an additional Ethernet driver is required, you need to create a special build file (e.g. `ws.eth8003x2`) and add a second copy of the `Net.ether8003` driver to the file.

Configuring multiple network links

Setting up a private network link

If there's a lot of traffic between certain nodes on a network, you can set up those nodes with a private network link in order to offload the communication flow from the main network. For example:



Chapter 10

Print spooling

In this Chapter...

Introduction

Using the spool utilities

Spooler architecture

The spool setup file

Using setup files

Example setup files

Accessing spoolers and queues

Introduction

Sharing resources on a network

QNX encourages the distribution of resources throughout a network with few artificial boundaries. Every device (disk, modem, printer, etc.) connected to a computer is, by default, a *shared* resource. Thus, a program running on any computer has equal access to any device on the network, whether the device is connected to the same computer (local) or is connected to another machine (remote).

Although it's easy for a user to transparently access any resource on the network, the system administrator may need to take steps to control access to certain types of resources. Printers and modems, for example, can be used by only one user at a time (unlike hard disks, which can be accessed by many users concurrently) and therefore require some sort of enqueueing facility to avoid conflicts. To allow convenient access to these shared, yet exclusive, resources, QNX provides a set of *spooling* services.

Spoolers

A *spooler* is simply a mechanism that accepts requests for a resource and then allocates the use of the resource according to a set of rules that have been specified.

To understand how a spooler can be useful, let's look at how a spooler controls access to a printer. A printer must be available at all times to users, yet it can print only one job at a time. If a spooler is present, users can send their data through the spooler rather than directly to the printer. Upon receiving data destined for the printer, the spooler writes this data into a temporary file instead of sending it immediately to the printer. Later, when the printer becomes available, the spooler will write the data to the printer. Thus, many users can freely submit print jobs, even though only one physical printer exists.

QNX implements spooling through the use of named queues that are referenced by the “lp” set of utilities; these queues also exist in the file namespace in the `/dev/spool` directory. Data written to a queue will be placed on an internal list and ultimately sent to a defined output device.

The QNX spooling server is:

lpsrvr

This server can maintain many different spool queues. The following utilities operate on spool queues:

lp	submit files to a spool queue
lprm	remove jobs from a spool queue
lpc	control spooler queue
lpq	display spool queue status

For more information on these utilities, see the *Utilities Reference*.

Using the spool utilities

Starting the spooler

Before any spooling can occur in a QNX system, you must run **lpsrvr**, the spool server:

lpsrvr &

To determine what resources it has available, and how it's expected to manage them, the **lpsrvr** utility first looks for a setup file called **/etc/config/lpsrvr.node** (where *node* is the node ID of the node **lpsrvr** is running on). If no setup file is found with a *.node* extension, **lpsrvr** will use the **/etc/config/lpsrvr** file. This file is found on the standard QNX distribution disks.

Submitting spool jobs

The following **lp** command will cause the file **report** to be inserted into the default spool queue (more on the default queue in "The spool setup file") and ultimately printed:

lp report

In systems where more than one spool queue is available, you can specify the queue name. The following command inserts **report** into a spool queue called **txt**:

lp -P txt report

Alternately, you could use a command that writes directly to the queue file:

```
cp report /dev/spool/txt
```

Querying spool jobs

To examine the spool queue, you can use the **lpq** utility. The following is a sample output from **lpq**:

```
1:  steve  [job #39]   1400 bytes  lalist.doc
2:  aboyd  [job #42]   2312 bytes  netdrv.c
```

This utility lets you determine when any submitted jobs have been completed; it also provides the spool job ID for use with other **lp** utilities.

Canceling spool jobs

The **lprm** utility lets you remove jobs from a spool queue. You can remove a job explicitly by specifying its job ID number. Given the state of the default queue shown above, **steve's** job (#39) could be canceled with this command:

```
lprm 39
```

If job #39 is currently in progress, it will be abandoned. The success of abandoning current spool jobs may vary with the type of output device you're using—some printers have large internal buffers.

The superuser may also remove all jobs belonging to a particular user. For example, all of **steve's** jobs can be canceled with the following command:

```
lprm steve
```

Controlling the spool queues

The **lpc** utility is a system administration tool for managing spoolers. It lets you perform many control functions, such as starting up or shutting down a queue. The following basic functions are provided:

- suspend/resume enqueueing of jobs
- suspend/resume dequeuing of jobs
- suspend/resume the current job
- delete the current job

- rearrange the jobs in a queue
- move jobs to a different queue
- display the status of queues

Note that `lpc`'s functionality overlaps that of `lpq` and `lprm`. This overlap is convenient, because unlike `lpq` and `lprm`, `lpc` can be used interactively.

Spooler architecture

The QNX spooling system is based on two objects: *queues* and *targets*. These work with each other to provide a flexible method of controlling data transformations and queuing.

A queue is an internal list of pending data to be sent to a target. As mentioned earlier, each queue is given a name that users specify when submitting jobs. A target is associated with the physical output device (e.g. a printer) and removes jobs from queues. You can connect the output of a queue to one or more targets or connect multiple queues to a single target. However, you can connect the output of a target to only one device.

Queues can have optional attributes called *filters*. There are two types: *copy-in* (`ci`) filters, which perform operations on the data before it's copied onto a queue (e.g. pagination); and *copy-out* (`co`) filters, which operate on the data after it's removed from a queue.

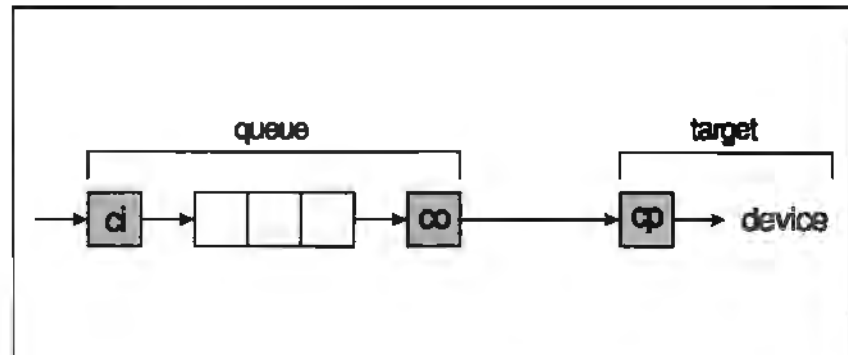
Targets may have an optional *control program* (`cp`) that allows the output device to be initialized between jobs if required. For example, if a job were aborted and the device needed to be primed again, a device-specific control program could detect SIGTERM and take whatever action was necessary to restore the device to a stable state.

The following diagrams illustrate how queues, filters, and targets can be configured to work with each other. Note that shading indicates an optional element.

Spooler architecture

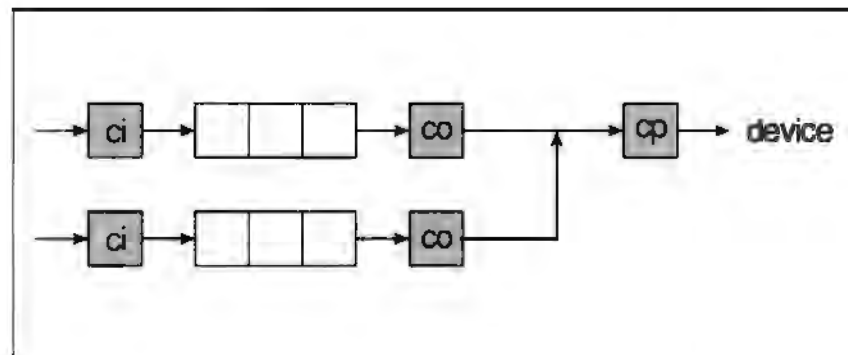
One queue feeding a single target

The following configuration **could** be used where there's a single printer and where a single (or no) translation of the data is required (e.g. a dot matrix printer):



Multiple queues feeding a single target

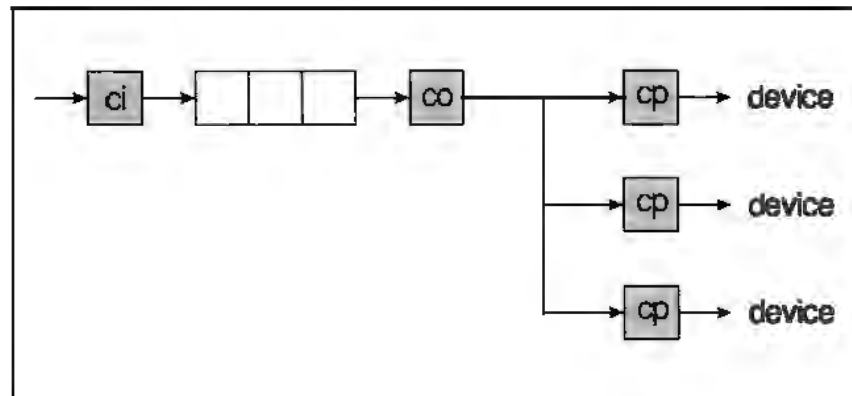
Multiple queues may feed a single target, in which case the target will select the appropriate job from all the jobs in those queues based upon queue priority, then upon time of submission (i.e. the oldest pending job).



At the end of this chapter, you'll find several example setup files. The above configuration is used in the example file in which one queue converts ASCII to PostScript, while the other is a direct PostScript queue. Both queues feed the same target, and the target sends the data to the PostScript laser printer.

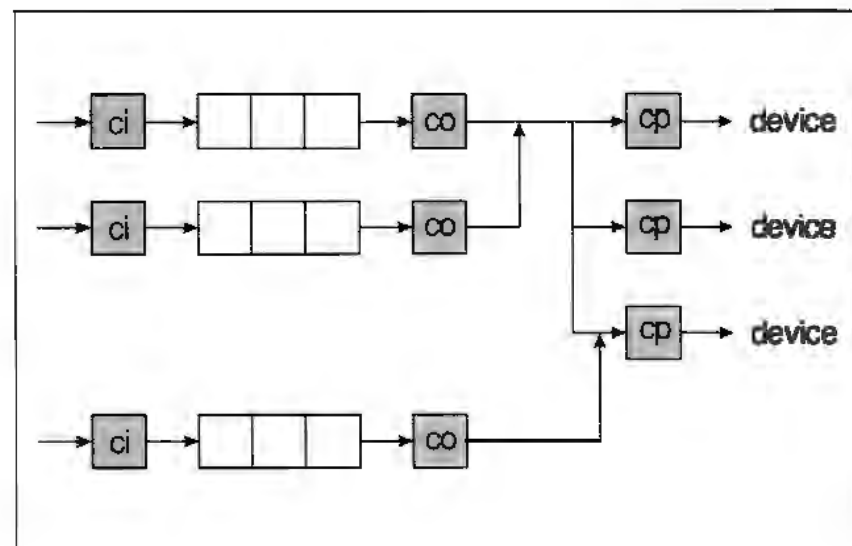
**One queue feeding
multiple targets**

If the output of a queue is sent to many targets, the spooler will select whichever target is available. The following configuration is useful if you have three printers side-by-side and it doesn't matter which one prints your job.



**Multiple queues feeding
multiple targets**

The following configuration is a combination of the previous two examples. A third queue has a separate channel to one of the targets. This channel could be used to ensure that jobs requiring the third printer are always sent to that printer (e.g. the printer has color capabilities).

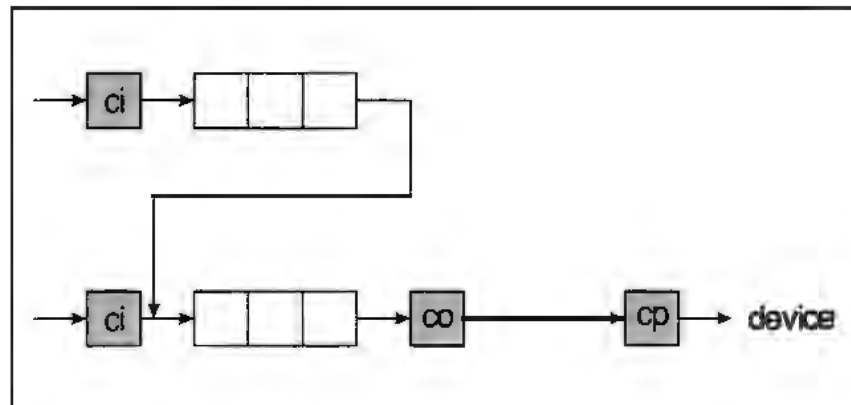


The spool setup file

Chaining queues

The output of a queue is usually sent to one or more targets, but it's possible instead to *chain* the output into another queue.

With chaining, the output of a queue is placed directly onto the destination queue, thus avoiding any possible copy-in operation on that queue. If the final queue in a chain has a copy-out filter, the filter will be applied.



The spool setup file

When started, the spooler accesses a file to get its configuration information. If no file is specified on the command line, the spooler uses the `/etc/config/lprvr` file. This file defines queues, targets, and the relationships between them.

Syntax definitions

Queues and targets have symbolic names as well as a set of attributes. Each entry in the setup file has the following format:

```
[name]
attribute
attribute
.
.
.
```

The definition of a queue or target begins with a *[name]* directive and consists of all valid attribute specifications until the next *[name]* directive. All leading white space is ignored. Comment lines start with a pound sign: `#`

If a single attribute spans two lines, the last character before the `<newline>` must be a backslash: `\`

The *name* may be up to 48 characters long, and may contain only alphanumeric characters.

If the object being described is a target, the name must be preceded by a dash (`-`). The dash is for delineation only; it isn't considered part of the name.

Each attribute consists of a two-letter key in one of the following forms:

<i>key</i>	Boolean
<i>key#number</i>	numeric
<i>key=string</i>	character

All *numbers* are assumed to be decimal numbers, unless they start with a leading zero (meaning octal) or a leading 0x (meaning hex).

All *strings* contain printable characters. The backslash (`\`) is a "special" character. It can be used to escape other characters. For example, a "real" backslash must be represented with: `\\`

The following table describes all defined keys, including the default used for each key if its corresponding attribute isn't specified.

The spool setup file

Key:	Description	Default	Use*
ab=string	Executed when target abandons command for any reason	nil; no command is executed	T
af=string	accounting file that commands may use	accounting isn't performed (lparvr logging is unaffected)	Q
ad=string	directory in which temporary spool files are found	/usr/spool/lp	G
ci=string	copy-in command; used before placing the job on the queue	binary copy-in of job; no transformation applied	Q
co=string	copy-out command; used when removing job from queue	binary copy-out of job; no transformation applied	Q
cp=string	device control program; used by the target	binary copy of job; no transformation applied	T
dv=string	device that the target uses	output is sent to standard output of lparvr	Q, T
mn#numeric	minimum number of jobs for queue before flushing	0; jobs are despoiled as soon as possible	Q
mx#numeric	maximum number of jobs the queue will hold	no limit; queue limits are based upon memory and disk space	Q
na=string	name; a string that describes the queue or target	nil string	Q, T
ok=string	command executed when target completes normally	nil; no action is performed	T
pr#numeric	priority to run this queue at (1-100); 100 is the highest	50	Q
qn=string	queue to chain a despoiled job onto	nil; delete job after despooling	Q
sp=string	registered name of spooler maintaining this queue	/qnx/spooler ; registered only if no queue specifies a name	G
ta=string	target to be associated with this queue	output is sent to standard output of lparvr	Q
wa#numeric	wait this number of seconds before despooling each job	0; jobs are despoiled as soon as possible	Q

* Q=queue; T=target; G=global

Since the keys are case-sensitive, Quantum reserves all keys formed by two lowercase letters. You can safely implement custom extensions by using uppercase or mixed-case keys. The spooler utilities will ignore any options they don't understand.

Global keywords

Two keywords, **sp** and **cd**, can define global information for the spooler. To indicate that they are to apply globally, you precede these keywords with a pair of empty brackets (**[]**).

Registering names—**sp**

Since the spooler always adopts the **/dev/spool** file namespace, only one spooler may run on each node. You can run multiple spoolers on your network if each spooler registers a different global name. Otherwise, each spooler will attempt to register the same default global name: **/qnx/spooler**. With multiple spoolers, you may wish to use names such as **/qnx/spooler2**, **/qnx/spooler3**, and so on.

To specify the global name to be registered by a spooler, you use the **sp** command in the setup file. The name must always begin with a leading slash (/). If no **sp** keyword is specified, the default name **/qnx/spooler** is globally registered (i.e. network-wide).

Specifying a temporary directory for spool files—**cd**

The **cd** keyword lets you define the directory to use for the creation of the spooler's temporary spool files. By default, temporary files are created in the **/usr/spool/lp** directory. These files get deleted when they're no longer required.

Any path you specify to the **cd** keyword should begin with a leading slash. Note that the specified directory must exist, with appropriate access rights assigned.

The following example setup file informs the spooler to register the name **/qnx/spooler2**; it also places temporary files in the **/tmp/spool2** directory:

```
# Global spooler variables
[]
  sp=/qnx/spooler2
  cd=/tmp/spool2

# Text queue
[txt]....
```

Metavariables

The spooler will set the following metavariables appropriately when it encounters them:

Variable:	Description:
<code>\$(file)</code>	the filename of the submitted file or “ <code>--standard input--</code> ” if no name is provided
<code>\$(fname)</code>	the fully resolved pathname of the submitted file or “ <code>--standard input--</code> ” if no name is provided
<code>\$(spfile)</code>	the name of the spool data file
<code>\$(username)</code>	the login name of the user who submitted the job
<code>\$(userid)</code>	the numeric user ID of the user who submitted the job
<code>\$(queue)</code>	the name of the queue the job currently belongs to
<code>\$(target)</code>	the name of the target the job currently belongs to
<code>\$(device)</code>	the name of the device the job is scheduled on
<code>\$(ncopies)</code>	the number of copies the user requested
<code>\$(jobid)</code>	the job ID number of this job

In addition, all the keys defined above can be referenced as metavariables. For example, `$(ci)` will expand to the name of the copy-in command, `ci=string`.

Default behavior

The `cat` command is used as the default copy-in and copy-out filter commands. Also, unless otherwise specified in the setup file, the standard input and standard output of filter commands are automatically connected to default files or processes. The default for copy-in is as follows:

```
cat < $(fname) > $(spfile)
```

There are two possible defaults for a copy-out, depending on whether or not a *control program* (i.e. `cp`) is defined:

```
cat < $(spfile) > $(device)
cat < $(spfile) | $(cp) > $(device)
```

For example, a setup file like this:

```
[txt]
ta=lpt
ci=pr -f -h
co=txt2ps
cp=init_printer
[-lpt]
dv=/dev/par
```

would result in the following substitutions:

```
pr -f -h < $(fname) > $(spfile)
txt2ps < $(spfile) | init_printer > /dev/par
```

Using setup files

Queues and targets

When you create a setup file, you specify each queue by giving it a name and an optional list of parameters; you specify each target by starting its name with a dash (-). To illustrate, here's a very simple setup file:

```
[txt]
ta=lpt
[-lpt]
dv=/dev/par
```

In this file we have a queue called **txt** and a target called **lpt**. When data is sent to the **txt** spool queue, the data is saved in a temporary spool file and is known as a “job.” When the spooler removes that job from the queue, the job is placed on the target, **lpt**, which then directs the data to the parallel port, **/dev/par**.

Filters

It's often necessary to filter spooled data, so **lprvr** provides the *copy-in* and *copy-out* mechanisms. As mentioned earlier, copy-in is run before the data is placed on the queue, while copy-out is run after the data is removed from the queue.

Note that if you have many queues feeding a single target and one of the queues has a copy-out filter that may take a long time to run, the target will be temporarily unavailable to the remaining queues if that queue is selected. (For more information, see “Example setup files” below.)



It's good practice to put double quotes around metavariables to ensure they're “safe” for the command.

Using copy-in

A good example of the use of a copy-in filter is to pass the data through `pr` to paginate it:

```
[txt]
  ci=pr -f -h "${file}"
  ta=lpt
[-lpt]
  dv=/dev/par
```

Using copy-out

You might use a copy-out filter, for example, when you have both a PostScript laser printer and an HP laser printer. You could have two copy-out filters to generate the proper output format. Let's say you have two programs: `txt2ps`, which generates PostScript, and `txt2hpgl`, which generates `hpgl`:

```
[ps]
  ta=lpt1
  co=txt2ps
[hp]
  ta=lpt2
  co=txt2hpgl
[-lpt1]
  dv=/dev/ser
[-lpt2]
  dv=/dev/par
```



Don't hardcode the output device in a copy-out definition; use targets for this purpose. Targets must have *exclusive* use of a device, and should be the only means of accessing a device within the spooling system.

Chaining queues

Queues can be chained—this moves the job from queue to queue. The following is a simple example of chaining onto a queue named **tmp**, which then sends the data to the **lpt** target:

```
[txt]
qn=tmp

[tmp]
ta=lpt

[-lpt]
dv=/dev/par
```

**Accounting
information**

The **af** keyword lets you specify a filename that the commands may access as **\$ (af)** so they can write any information they want to log.

**Dealing with
errors**

Input errors—The invoking **lp** utility will report any errors that occur during the input of data into a queue. If you submit data by directly writing to the spool file in the **/dev/spool** directory, write errors will occur if something prevents a successful copy into the queue.

Output errors—You can use the **ab** keyword to specify a command to execute if a target abandons a job. The following would inform the invoking user of the error via a mail message:

```
ab=echo lparvr print job ${jobid}, file ${file} failed \
| mailx $(username)
```

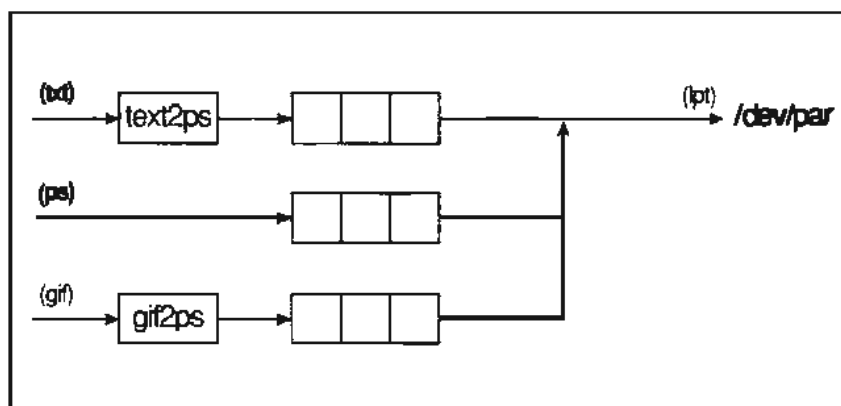
Example setup files

Multiple queues feeding a single target

The following example shows a set of queues that share a common target. The three queues are named:

txt (ASCII text files)
ps (PostScript files)
gif (Graphic Interchange Format files)

The configuration is as follows:



Here's the file to set this up:

```
# ASCII to postscript queue:
[txt]
  ta=lpt
  ci=text2ps
  pr#50

# direct postscript queue:
[ps]
  ta=lpt
  pr#60
```

```
# GIF to postscript queue:
[gif]
  ta=lpt
  ci=gif2ps
  pr#5

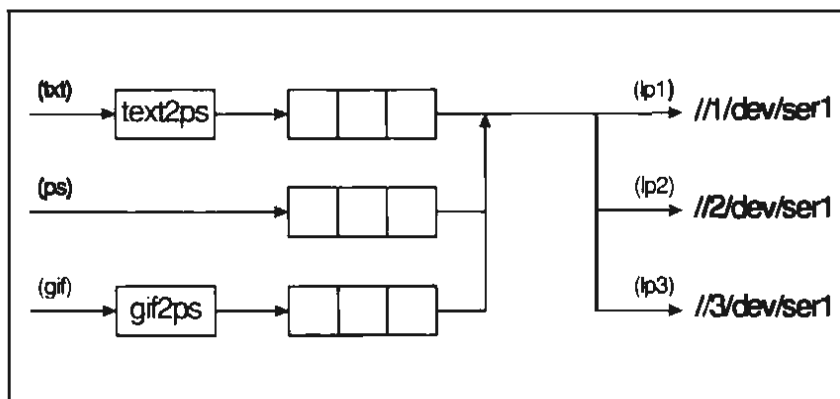
# target printer:
[-lpt]
  dv=/dev/par
```

In this example, users send files with the `lp` utility to the appropriate queue, which converts the file through a copy-in filter (except for the `ps` queue). The queue then sends the converted data to the target, `/dev/par`.

Since the hypothetical filter programs `text2ps` and `gif2ps` may take a relatively long time to process a job, copy-in filters are used rather than copy-out filters. A copy-out filter will tie up the target, preventing other jobs from being dequeued while the filter is running. The chosen configuration allows other jobs to be sent to the target while the PostScript translation is being generated. Also, since GIF files tend to be large, they're assigned a lower priority than the others.

Multiple queues feeding three targets

The following example is a further refinement of the above setup, with some additional features required for a larger configuration. There are now three laser printers, all PostScript, located in different parts of the building (connected to `//1/dev/ser1`, `//2/dev/ser1`, and `//3/dev/ser1`). The configuration looks like this:



Here's the file to set this up:

```
# ASCII to postscript queue:
[txt]
  ta=lp1,lp2,lp3
  ci=text2ps
  pr#50

# direct postscript queue:
[ps]
  ta=lp1,lp2,lp3
  pr#60

# GIF to postscript queue:
[gif]
  ta=lp1,lp2,lp3
  ci=gif2ps
  pr#5

# target printers:
[-lp1]
  dv=/1/dev/ser1
  ok=echo file ${fname} sent to ${target} | mailx ${username}
  ab=echo file ${fname} did not get printed | mailx ${username}

[-lp2]
  dv=/2/dev/ser1
  ok=echo file ${fname} sent to ${target} | mailx ${username}
  ab=echo file ${fname} did not get printed | mailx ${username}

[-lp3]
  dv=/3/dev/ser1
  ok=echo file ${fname} sent to ${target} | mailx ${username}
  ab=echo file ${fname} did not get printed | mailx ${username}
```

The above configuration uses the same three queues described earlier (**txt**, **ps**, and **gif**) but they now feed three separate targets. The spooler selects the first available target from the set of targets (**lp1**, **lp2**, **lp3**) and sends the data to its corresponding printer.

Queue selection is based on priority first, then on the age of the job.

In this example, a mail message is sent to the submitter indicating whether or not the job completed normally.

Accessing spoolers and queues

Many spoolers may be present on a network; each of these can maintain multiple queues. To communicate with any given spooler, the **lp** utilities first locate the spooler through the unique global name that the spooler registers (see the section on “Global keywords”).

When using any **lp** utility, you can specify a spooler and queue in several ways. You can:

- omit both the spooler and the queue
- specify only the spooler
- specify only the queue
- specify both the spooler and the queue

If you don't specify the spooler on the command line, the utility checks the **LPSRV**R environment variable, which contains the name of the default spooler. However, if **LPSRV**R isn't defined, the utility will use the spooler that has registered the default global name:

/qnx/spooler

If the utility successfully locates a spooler, but you haven't specified a queue on the command line, the utility will check the **LPDEST** environment variable, which contains the name of the default queue. However, if **LPDEST** isn't defined, the utility will use the first queue entry in the setup file of the located spooler.

LPSRVR and **LPDEST**

The **LPSRV**R and **LPDEST** environment variables are used when the command-line information given to the **lp** utilities doesn't fully specify which spooler or queue to use.

LPSRVR

LPSRVR specifies the default spooler. The following setting of **LPSRV**R would indicate that the default spooler is the one with the name **/qnx/spooler2**:

```
export LPSRVR=/qnx/spooler2
```

The name specified in **LPSRV**R must always begin with a leading slash. **LPSRV**R must never contain the name of a queue.

LPDEST **LPDEST** specifies the default queue. You can use **LPDEST** in two ways. You can specify a queue only:

```
export LPDEST=waybills
```

Or you can specify a spooler and a queue:

```
export LPDEST=/qnx/spooler2/waybills
```

Spooler Spooler Queue

If you specify a spooler in **LPDEST**, the **LPSRVR** variable is ignored, even if it's defined.



For compatibility, the **lp** utilities look at the traditional UNIX environment variable **PRINTER** if **LPDEST** isn't present.

Examples

Let's look at a few simple examples that show some of the ways you can specify spoolers and queues. For these examples, let's assume you have two spoolers on the network, each with two queues.

The first spooler uses the default global name, **/qnx/spooler**; its queues are named **txt** and **ps**. The second spooler uses the name **/qnx/spooler2**; its queues are named **cheques** and **waybills**.

First spooler:

```
/qnx/spooler/txt
/qnx/spooler/ps
```

Second spooler:

```
/qnx/spooler2/cheques
/qnx/spooler2/waybills
```

**Naming neither a
spooler nor a queue**

Let's say you enter the following **lp** command, specifying neither a spooler nor a queue:

```
lp test.dat
```

The utility will first try to locate a spooler by checking **LPSRVR**. If that variable isn't defined, the utility will locate the spooler with the default global name, **/qnx/spooler**.

The utility will then try to determine the queue by checking **LPDEST**. If that variable isn't defined, the utility will select the first queue specified in the setup file of the located spooler.

Naming only the spooler

If you specify a string that begins with a leading slash, the string is always assumed to be a spooler name. Thus, if you enter the following command, the utility will treat **/qnx/spooler2** as a spooler:

```
lp -P /qnx/spooler2 test.dat
```

Because the second spooler uses **/qnx/spooler2** as its name, that spooler will be located. The utility will then try to use **LPDEST** as the default queue. If **LPDEST** isn't defined, the job is submitted to the queue **cheques**, since that's the first queue in the setup file of **/qnx/spooler2**.

Naming only the queue

If you specify a string that *doesn't* begin with a leading slash, the string is always assumed to be a queue name. Thus, if you enter the following command, the utility will treat **txt** as a queue:

```
lp -P txt test.dat
```

The utility will first try to locate a spooler by checking **LPSRV**. If that variable isn't defined, the utility will use the first spooler, since that spooler has registered the default global name, **/qnx/spooler**.

**Naming both the spooler
and the queue**

In the following example, both the spooler and the queue are named. Since the string begins with a leading slash, the utility will initially attempt to find a spooler called **/qnx/spooler2/waybills**.

```
lp -P /qnx/spooler2/waybills test.dat
```

However, since the spooler **/qnx/spooler2/waybills** doesn't exist, the search will fail, at which point the utility will treat the specified string as a spooler name followed by a queue name. The spooler with the name **/qnx/spooler2** will be located, and jobs will be submitted to the **waybills** queue on that spooler.

Initialization files

You might find it useful to configure several default spoolers, if, for example, your marketing, sales, and tech departments each has a spooler running:

LPSRVR=/qnx/spooler	marketing
LPSRVR=/qnx/spooler2	sales
LPSRVR=/qnx/spooler3	tech

You normally initialize environment variables beforehand in system initialization files. You may wish to use one of the following files to initialize the variables:

/etc/config/sysinit.node	run at boot time
/etc/default/login	run at login time
/etc/profile	run by every login shell

Chapter 11

Building a Custom Operating System Image

In this Chapter...

Introduction

Constructing a build file

Introduction

QNX is a modular operating system composed of a microkernel and one or more processes that provide services. For example, a process named **Fsys** provides filesystem services, while a process named **Dev** provides device services.

When you build an operating system image, you select the services that you want to be available immediately after boot, and include the processes that provide those services into a custom-built operating system image. You create this image with the **buildqnx** utility. The image can be booted from disk by the QNX partition loader or booted over the network using the **netboot** utility.

Constructing a build file

The **buildqnx** utility takes a text file containing a list of process names as input and produces a binary image file containing these processes as output. The input file is called a “build” file. The build files are kept in the **/boot/build** directory and the image files are kept in the **/boot/images** directory.

You can create an image by invoking **buildqnx** directly or by using the **make** utility and the **Makefile** in the **/boot** directory. For example, you could use:

```
buildqnx build/hard.1 images/hard.1
```

OR

```
make b=hard.1
```

Each program you want included in the created image occupies two lines in the build file. Empty lines are ignored. The first line is the pathname of the program you want included. The second line starts with a **\$** followed by a number representing the size of the heap (alloc space) to be used, followed by a command.

Setting heap sizes

When an image is created for a process, space is reserved in the image's data segment for local memory allocation. If this space is set too small, the system will have to dynamically grow the data segment at runtime, which can result in minor memory fragmentation. If the space is set too large, you'll end up wasting memory. For more information on setting appropriate heap sizes, see the documentation on **Proc** in the *Utilities Reference*.

Setting priorities

You can set the initial priority of a process by following the heap size with an optional `:nn` argument, where `nn` is the desired priority. For example, the following would start the Device Manager (**Dev**) with a 4K heap and a priority of 24:

```
/bin/Dev  
$ 4000:24 Dev
```

By setting the priority in this way, you can control the order in which processes start up.

Selecting processes for an image

The processes you include in an operating system image are determined by several factors. You can typically group images into three classifications:

- images that boot from disk
- images that boot over the network
- images for embedded systems

For images that boot from disk or over the network, you can start most processes after booting. You do so by placing their command line in the system initialization file that QNX executes after boot (see the section "The system initialization file" in Chapter 4). This lets you keep the boot image small and simple.

When you're building an image, remember that there are two required processes:

- the Process Manager/Microkernel (`/boot/sys/Proc`)
- the system shared library (`/boot/sys/Slib`)

Disk images

For hard disk booting, you need to include:

- the two required system processes (**Proc** and **Slib**)
- the Filesystem Manager (**Fsys**)
- the driver required to access the drive (**Fsys.driver**)



For more information, see **buildqnx** and **sinit** in the *Utilities Reference*.

The Makefile for disk booting

The **Makefile** under the directory **/boot** contains an entry for making a generic hard disk boot. You can provide command-line macros to change the defaults. The following macros are defined:

Macro	make argument	Default	Description
\$b	b=buildfile	<i>no default</i>	build file
\$d	d=driver	Fsys.wd	disk driver
\$h	h=hard disk	/dev/hd0	hard disk device
\$i	i=image file	images/buildfile	image file
\$p	p=parameters	<i>no default</i>	hard disk parameters

This is the makefile:

```
FILES = @build/$b

# Define some defaults which may be overridden
# by h=xx and d=xx

d      = Fsys.wd
h      = /dev/hd0
i      = $b

$b:    images/$b

# Each OS image file is dependent on the core
# modules it is made from.

images/$b: build/$b $(FILES)
          buildqnx -v d=$d h=$h p=$p build/$b images/$i
```

Examples The following examples will help clarify the **Makefile** usage:

Use build file **/boot/build/hard.1** and make an image **/boot/images/hard.1** that contains a Western Digital driver and a **mount** command that mounts a QNX filesystem on **/dev/hd0t77**:

```
make b=hard.1
```

Use build file **/boot/build/hard.1** and make an image **/boot/images/hard.1** that contains a PS/2 SCSI driver instead of the Western Digital driver and a **mount** command that mounts a QNX filesystem on SCSI ID 6:

```
make b=hard.1 d=Fsyp.sps2s h=/dev/hd6
```

Use build file **/boot/images/hard.1** and make an image **/boot/images/p2s** that contains a PS/2 SCSI driver instead of the Western Digital driver and a **mount** command that mounts a QNX filesystem on SCSI ID 6. The image is written to the image file **p2s** rather than the default, which is the name of the **b=** option (in this case **hard.1**).

```
make b=hard.1 d=Fsyp.sps2s h=/dev/hd6 i=p2s
```

**Copying an image to
/.boot**

Once you've built an image, it won't become the new boot image until you copy it to the **/.boot** file. However, before you do this, you should make a backup of the current **/.boot** file by copying it to the **/.altboot** file:

```
cp /.boot /.altboot
```

If for any reason your new image doesn't work properly, you can press <Esc> when prompted during the boot process and load the **/.altboot** file instead of the **/.boot** file. When you select the alternate boot image, the normal check for the **/etc/config/sysinit.node** file is replaced by a check for the **/etc/config/altsysinit** file. You should ensure that the **altsysinit** file contains the latest copy of your working **sysinit** file:

```
cp /etc/config/sysinit.node /etc/config/altsysinit
```

Note that the `hard.1` build file doesn't contain the Device Manager `/bin/Dev`. The Device Manager and its drivers are usually started in the system initialization file *after* the system boots.



If the system initialization file isn't executed, the Device Manager will *not* be started. As a result, your keyboard and system console won't function.

Network images

For network booting, you need to include:

- the two required system processes (`Proc` and `Slib`)
- the Network Manager (`Net`)
- the driver required to access the network (`Net.driver`)

The Makefile for network booting

The `Makefile` under `/boot` contains an entry for making a workstation boot.

The following example uses the build file `/boot/build/ws.arcnet` and makes an image `/boot/images/ws.arcnet`:

```
make b=ws.arcnet
```

When you boot over the network, you have the option of loading a pre-built image or building one on the fly. If you build an image on the fly—which is recommended—then you won't need to build one manually as shown above. This option is specified in the `/etc/boot/netboot` file and documented with the `netboot` utility.



When the `netboot` utility invokes `buildqnx` to build an image on the fly, the image file is *not* written to disk.

Embedded images

An embedded system doesn't need to contain a filesystem or a network manager. As such, the image should contain *all* the processes needed to run the system. This image would then have to be transferred to the embedded system (most likely in ROM).

Chapter 12

Using the Revision Control System (RCS)

In this Chapter...

What is RCS?

Getting started

Automatic identification

RCS file format

RCS commands

What is RCS?

The Revision Control System (RCS) helps you manage multiple revisions of text files. RCS automates the storing, retrieving, logging, identifying, and merging of revisions. You should find RCS useful for any text that's revised frequently, including programs, documentation, graphics, papers, and form letters.

More specifically, RCS lets you do the following:

- **Store and retrieve multiple revisions of text**—With RCS, changes don't destroy the original; previous revisions remain accessible. You can retrieve revisions according to ranges of revision numbers, symbolic names, dates, authors, and states.
- **Maintain a complete history of changes**—RCS logs all changes automatically. In addition to the text of each revision, RCS stores the author, the date and time of check-in, and a log message summarizing the change. The logging makes it easy for you to find out what happened to a module, without making you compare source listings or track down colleagues.
- **Resolve access conflicts**—When two or more programmers try to modify the same revision, RCS alerts the programmers and prevents one modification from corrupting the other.
- **Maintain a tree of revisions**—RCS can maintain separate lines of development for each module. It stores a tree structure that represents the ancestral relationships among revisions.
- **Merge revisions and resolve conflicts**—You can coalesce two separate lines of development of a module by merging. If the revisions to be merged affect the same sections of code, RCS alerts you about the overlapping changes.
- **Control releases and configurations**—You can assign symbolic names to revisions and mark the revisions as released, stable, experimental, etc. With these facilities, configurations of modules can be described simply and directly.

- **Automatically identify each revision**—RCS lets you identify each revision with name, revision number, creation time, author, etc. The identification is like a stamp that you can embed at an appropriate place in the text of a revision. The identification makes it simple to determine which revisions of which modules make up a given configuration.
- **Minimize secondary storage**—RCS needs little extra space for revisions; it needs space only for the differences. If intermediate revisions are deleted, the corresponding deltas are compressed accordingly.

Getting started

You should find the basic user interface for RCS easy to learn. You can perform most essential functions with only two commands:

- **ci**—“check-in” command. This command deposits the contents of a file into an archival file called an RCS file. An RCS file contains all revisions of a particular file.
- **co**—“check-out” command. This command retrieves revisions from an RCS file, creating a working file.

Checking in

Now let’s suppose you have a file `f.c` that you wish to put under control of RCS. Invoke the check-in command:

```
ci f.c
```

This command:

- 1) creates the RCS file `f.c,v`
- 2) stores `f.c` into `f.c,v` as revision 1.1
- 3) deletes `f.c`
- 4) asks you for a description

The description you enter should be a summary of the contents of the file. All later check-in commands will ask you for a log entry, which should summarize the changes that you made.

Files ending in `,v` are called RCS files (`v` stands for “versions”); all others are called working files.

Checking out

To get back the working file `f.c` in the above example, use the check-out command:

```
co f.c
```

This command extracts the latest revision from `f.c,v` and writes it into `f.c`.

If you want to edit `f.c`, you must lock it when you check it out with the `co` command:

```
co -l f.c
```

You can now edit `f.c`.

Let's say you've done some editing and you want to see what changes you've made. The command:

```
rcsdiff f.c
```

tells you the difference between the most recently checked-in version and the working file. You can then check the file back in by invoking:

```
ci f.c
```

This increments the revision number properly.

Locking files

If `ci` complains with the message:

```
ci error: no lock set by your name
```

you've tried to check in a file even though you didn't lock it when you checked it out. Of course, it's too late now to do the check-out with locking, because another check-out would overwrite your modifications. Instead, you should invoke:

```
rccs -l f.c
```

This command will lock the latest revision for you, unless somebody else got ahead of you already. In this case, you'll have to negotiate with that person.

Locking assures that you—and only you—can check in the next update, and avoids problems if several people work on the same file. Even if a revision is locked, it can still be checked out for reading, compiling, etc. The only thing that locking prevents is a check-in by anybody but the locker.

If your RCS file is private (i.e. you're the only person who's going to deposit revisions into it), strict locking isn't needed and you can turn it off. If strict locking is turned off, the owner of the RCS file need not have a lock for check-in; all other users still do. You turn strict locking on and off with these commands:

```
rcs -L f.c      sets locking to strict
```

```
rcs -U f.c      sets locking to non-strict
```

Creating an RCS directory

If you don't want to clutter your working directory with RCS files, create a subdirectory called **RCS** in your working directory, and move all your RCS files there. RCS commands will look first into that directory to find needed files. All the commands discussed above will still work, without any modification. You can specify pairs of RCS and working files in three ways:

- both files are given
- only the working file is given
- only the RCS file is given

Both RCS and working files may have arbitrary path prefixes; RCS commands pair them up intelligently.

Protecting your working file

To avoid deleting the working file during check-in—in case you want to continue editing or compiling—invoke either of these commands:

```
ci -l f.c
```

```
ci -u f.c
```

These commands check in **f.c** as usual, but perform an implicit check-out. Thus, they save you one check-out operation. The **ci -l** command also locks the checked-in revision—it's useful if you want

to continue editing. The `ci -u` command doesn't lock the checked-in revision—it's useful if you just want to read the file. Both commands update the identification markers in your working file (see "Automatic Identification").

Assigning revision numbers

You can give `ci` the number you want assigned to a checked-in revision. Let's say all your revisions were numbered 1.1, 1.2, 1.3, etc., and you want to start release 2. Either of these commands:

```
ci -r2 f.c
```

```
ci -r2.1 f.c
```

assigns the number 2.1 to the new revision. From then on, `ci` will number the subsequent revisions with 2.2, 2.3, etc. The corresponding `co` commands:

```
co -r2 f.c
```

```
co -r2.1 f.c
```

retrieve the latest revision numbered 2.x and the revision 2.1, respectively. If you invoke `co` without a revision, it selects the latest revision on the trunk (i.e. the highest revision with a number consisting of two fields). Numbers with more than two fields are needed for branches. For example, to start a branch at revision 1.3, invoke:

```
ci -r1.3.1 f.c
```

This command starts a branch numbered 1 at revision 1.3, and assigns the number 1.3.1.1 to the new revision. For more information about branches, see the section on "RCS file format."

Automatic identification

RCS lets you place special identification strings into your source and object code. You simply place the marker:

\$Id\$

into your text—inside a comment, for instance. RCS will replace this marker with a string of the form:

\$Id: filename revision date time author state \$

With such a marker on the first page of each module, you can always see which revision you're working with. RCS keeps the markers up to date automatically. To propagate the markers into your object code, simply put them into literal character strings. In C, you do this as follows:

```
static char rcsid[] = "$Id$";
```

The command **ident** extracts such markers from any file, even object code and dumps. Thus, **ident** lets you find out which revisions of which modules were used in a given program.

You may also find it useful to place the marker **\$Log\$** into your text, inside a comment. This marker accumulates the log messages that are requested during check-in. Thus, you can maintain the complete history of your file directly inside the file. There are several additional identification markers; see **co** in the *Utilities Reference* for details.

Want more information?

If you're interested in finding out more about RCS, we recommend the following:

Walter F. Tichy, "RCS—A System for Version Control,"
Software—Practice & Experience 15, 7, July 1985

RCS file format

An RCS file's contents are described by the grammar below. The text is in free format—the following characters, which collectively form white space, have no significance except in strings:

<space>
<backspace>
<tab>
<newline>
<vertical tab>
<form feed>
<carriage return>

Strings are enclosed by at-signs (@), one at each end of the string. If a string is to contain an at-sign, the @ must be doubled. Otherwise, strings may contain arbitrary binary data.

The meta syntax uses the following conventions:

Convention:	Purpose:
	separates alternatives
{ and }	encloses optional phrases
(and)*	encloses phrases that may be repeated zero or more times
{ and }+	encloses phrases that must appear at least once and may be repeated

Note also that terminal symbols are in **boldface**; nonterminal symbols are in *italics*.

```

rcstext    ::= admin {delta}* desc {deltatext}*

admin      ::= head      {num};
              { branch    {num}; }
              access      {id}*;
              symbols      {id : num}*;
              locks        {id : num}*;    {strict;}
              { comment    {string}; }
              { expand      {string}; }
              { newphrase }*

delta      ::= num
              date         num;
              author       id;
              state        {id};
              branches      {num}*;
              next          {num};
              { newphrase }*

desc       ::= desc      string

deltatext  ::= num
              log         string
              { newphrase }*
              text        string

num        ::= {digit{ . } }+
digit      ::= 0 | 1 | ... | 9
id         ::= letter{idchar}*
letter     ::= any letter
idchar     ::= any visible graphic character except special
special    ::= $ | , | . | : | ; | @
string     ::= @{any character, with @ doubled}*@
newphrase  ::= id word* ;
word       ::= id | num | string | :

```

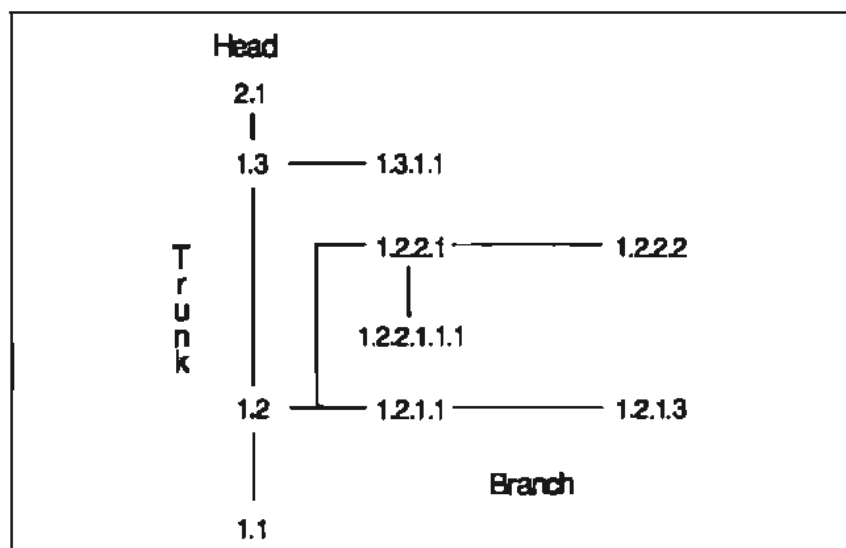
Identifiers are case sensitive. Keywords are in lowercase only. The sets of keywords and identifiers may overlap. In most environments, RCS uses the ISO 8859/1 encoding:

- letters are octal codes 101-132, 141-172, 300-326, 330-366 and 370-377
- visible graphic characters are codes 041-176 and 240-377
- white space characters are codes 010-015 and 040

The *newphrase* productions in the grammar are reserved for future extensions to the format of RCS files. No *newphrase* will begin with any keyword already in use.

The *delta* nodes form a tree. All nodes whose numbers consist of a single pair (e.g. 2.3, 2.1, 1.3, etc.) are on the trunk, and are linked through the *next* field in order of decreasing numbers. The *head* field in the *admin* node points to the head of that sequence (i.e. it contains the highest pair). The *branch* node in the *admin* node indicates the default branch (or revision) for most RCS operations. If empty, the default branch is the highest branch on the trunk.

All delta nodes whose numbers consist of $2n$ fields (n) (e.g. 3.1.1.1, 2.1.2.2, etc.) are linked as follows. All nodes whose first $2n-1$ number fields are identical are linked through the *next* field in order of increasing numbers. For each such sequence, the *delta* node whose number is identical to the first $2n-2$ number fields of the deltas on that sequence is called the branchpoint. The *branches* field of a node contains a list of the numbers of the first nodes of all sequences for which it is a branchpoint. This list is ordered in increasing numbers.

Example

RCS commands

The following table summarizes the RCS commands. All of these commands are described in the *Utilities Reference*.

Command:	Description:
ci	check in RCS revisions
co	check out RCS revisions
ident	identify files
rce	change RCS file attributes
rcsclean	clean up working files
rcsdiff	compare RCS revisions
rcsmerge	merge RCS revisions
rlog	print log messages and other info about RCS files

Chapter 13

Making Backups

In this Chapter...

Introduction

When to back up

Backup formats

Backup media

Compression

Archive examples

Introduction

This chapter deals with making a copy of your data to guard against hardware, software, or human error that may destroy the original. If your data is important to you, you should regularly perform a set of backup procedures that would allow you to restore lost data with minimal cost to you in time and money. Remember: hard disks *do* fail and people *do* make mistakes. It's too late to start a backup after your data is gone!

You can back up an entire filesystem or only portions of it. Users may elect to back up their own files only, usually to floppy diskette. To back up large portions of the filesystem with files owned by many users, you'll need to have read permissions on these files. The superuser (`root` login account) has such privileges.

When to back up

You should back up often enough so that you can recover data that is still current or can be made current with minimal work. In a software development group, this may range from a day to a week. Each day of out-of-date backup will generally cost you a day of redevelopment. If you're saving financial or point-of-sale data, then daily or even twice-daily backups are common.

Backup formats

QNX supports a variety of backup formats that can be classified into two groups: *archives* and *regular filesystems*.

An *archive* consists of one or more files, merged into a single unit with its own directory of contents. This single unit can be saved either to a regular QNX file or to a raw block device like a floppy or tape.

Saving to a *regular filesystem* simply involves copying the files. In this case, the destination must be a device with a mounted QNX filesystem on it.

Archive backups

QNX supports three major archive utilities:

- **cpio**
- **tar**
- **pax**

Both **cpio** and **tar** are implemented as links to **pax**, which is capable of reading and writing both **cpio** and **tar** formats. Both **cpio** and **tar** are common on UNIX systems. The **pax** utility is a cover utility for **cpio** and **tar**, so it doesn't support its own archive format. By default, **pax** will use the **tar** format when creating an archive.

The **pax** utility will detect when you've reached the end of a media on a volume and will prompt you to insert the next volume to be used for the save. The result is a backup spanning several media (diskettes, tapes, etc.).

Unfortunately, the **tar/cpio** format does *not* label the media with volume IDs. If you mixed up your media or inserted the wrong one out of order, you would end up restoring bad data.

To safeguard against this possibility, QNX is shipped with the **vol** utility, which labels each volume with its sequence number and therefore prevents you from inserting out-of-sequence media. The **vol** utility will, by default, step over block one of all media. This is important for floppies and cartridge disks that contain a QNX signature in block one. This signature contains the size of the diskette (360K, 1.2M, etc.) and allows for automatic remounting of removable media by the filesystem. Quantum ships its distribution diskettes using **pax** to create an archive, **freeze** to compress the data, and **vol** to write to the floppies.



If you wish to save data for restoration on a UNIX system other than QNX, don't use **freeze** or **vol**, since you won't find those utilities to do the restore at that end. Instead, use **pax** to save and restore directly to the target media.

Filesystem backups

You back up to a filesystem by copying files, probably with the `cp` utility or with `cpio -p`. If your destination media is a floppy diskette, the `cp` utility will prompt you for more diskettes, but remember that no single file can be larger than the diskette size. If you wish to back up to floppy, we recommended that you use one of the archive utilities.

Backup media

Your choice of backup media will be determined by available hardware and cost. There are four common choices:

- floppy
- tape
- removable disk
- fixed disk

Floppy

Floppies are the most widely available device for backups. Their major shortcoming is their limited size. Since the QNX `pax` and `vol` utilities let you span media, your only concern will be having to feed several floppy diskettes into the drive. If you have to deal with more than four or five floppies, this will make the procedure unpleasant enough that backups may occur less frequently. You might want to consider compressing your data as described below.

In order to back up/restore from a floppy diskette, you must make sure the floppy driver has been started. This is described under `Fsys . floppy` in the *Utilities Reference*. For most people, the following command line will start the driver (assuming that `Fsys` is already running), which will then create a block special file for each floppy:

Command line	Block special files
<code>Fsys . floppy</code>	<code>/dev/fd0</code> (floppy drive A) <code>/dev/fd1</code> (floppy drive B)

If you use an archive utility for your backups, it will read and write directly to the floppy block special file. If you elect to use the `cp` utility, you'll have to mount a filesystem on the block special device. For example:

```
mount /dev/fd0 /fd0
```

Remember to format and initialize new floppies. The following example would initialize a high density 5¼" floppy diskette:

```
fdformat -s 1.2m /dev/fd0
dinit /dev/fd0
```

You may now treat the floppy as a QNX filesystem mounted as `/fd0`.

Tape

QNX provides a driver for the QIC-02 interface standard. This is a standard ¼-inch cartridge tape that can hold from 60 to over 300 megabytes of data on a single tape.

To back up/restore from a tape, you must make sure the tape driver has been started. This is described under `Fsys.qic02` in the *Utilities Reference*. For most people, the following command line will start the driver, which will then create a block special file for the tape:

Command line	Block special files
<code>Fsys.qic02</code>	<code>/dev/tp0</code>

The archive utilities will read and write directly to the tape's block special file. You can't mount a filesystem on this type of block special file.

When the QIC-02 driver receives a request to read or write, it does so by starting at the current tape position. If you're starting a new backup, you'll need to erase and rewind the tape. A number of tape position and control functions are provided by the `tape` utility, which is described in the *Utilities Reference*. For example, the following command would erase, then rewind the tape in preparation for an archiving procedure:

```
tape erase
```

Removable disk

Removable hard disks are popular in both magnetic and optical formats. The optical formats hold more data, but they're slower. Most units use a SCSI interface to the computer, so you may want to consider making your internal fixed disk a SCSI drive. Unlike floppies and tapes, a removable hard disk lets you avoid using the archive utilities like **pax**. Instead, you'd likely use **cp** to copy your data to a real filesystem on the cartridge. This allows you to recover single files very easily and quickly.

Fixed disk

You can place a second hard disk in your machine or you can back up to a hard disk on another machine in the network. This will give you an online backup. However, it's still a good idea to maintain off-site storage.

Compression

You can use a compression utility to reduce the amount of space required to store data. The amount of compression will depend on the nature of the data you're saving. Some databases containing large amounts of repetitive data may compress up to 90%. Other data might compress less than 10%. Although compressing may save media space, it has two side effects:

- Compressing requires a fair amount of computation and may slow down the saving of data.
- You can't recover compressed data should a defect develop, such as a bad block, on your saved backup media. Potentially, all data following a bad block can be lost. In comparison, uncompressed data will simply have a "hole" in the middle.

You may use the **freeze** utility to compress your data and the **melt** utility to restore it. Both these utilities will act on a stream of data as well as on files. This ability to act as a filter lets you connect them to one of the standard archivers via a pipe. For example, Quantum distributes the QNX operating system in compressed form on floppies using **pax**, **freeze**, and **vol**.

Archive examples

Compressed floppy archive

Collect files under **/home** into a **tar** format archive, compress the archive and write it out to as many floppies as are needed, adding sequence numbers to the diskettes:

```
pax -w -x ustar /home | freeze | vol -w /dev/fd0
```

Read data off floppies, uncompress it back into an archive, and restore the files:

```
vol -r /dev/fd0 | melt | pax -r
```

UNIX-compatible floppy archive

Save files under **/home/brianc** in a **tar** format archive for restoration on a UNIX system:

```
pax -w -t/dev/tp0 -xustar /home/brianc
```

Save files under **/home/brianc** in a **cpio** format archive for restoration on a UNIX system:

```
pax -w -xcpio -t/dev/tp0 /home/brianc
```

Restore data in a **tar** or **cpio** format archive from another UNIX system and place all files under **/usr/unix**:

```
pax -r -s -t/dev/fd0 ",,/,usr/unix/, "
```

Tape archive

Start a new tape archive and save all files onto a QIC-02 tape:

```
tape erase  
pax -w -t/dev/tp0 /
```

Append files that have changed since the date of the file **lastsave** to the end of an existing archive tape. After the save, update the time of **lastsave**:

```
tape forward  
find / -newer lastsave | pax -w -t/dev/tp0  
touch lastsave
```

Restore all the files on a tape that are under the directory
`/home/robo`:

```
tape rewind
pax -r "/home/robo/*" -t/dev/fd0
```

Cartridge/optical

Copy all files from the filesystem on node 1 to the filesystem on
node 2:

```
cp -Rp //1/ //2/
```

In the following example, the disk on node 2 is a very large optical.
A full backup is done each Friday and a partial backup of modified
files could be done each day of the week:

```
cp -Rp //1/ //2/fri
cp -Rp -a date //1/ //2/mon
cp -Rp -a date //1/ //2/tue
.
.
.
```

You can also do this with `cpio -p`.

Chapter 14

Disk & File Recovery

In this Chapter...

Introduction

Overview of QNX 4 disk structure

File maintenance utilities

Disk recovery procedures

What to do if your system will no longer boot

Recovering lost files and directories



We recommend that you read this chapter in its entirety in case you ever have to perform any disk recovery procedures.

Introduction

The QNX filesystem achieves high throughput without sacrificing reliability. The filesystem is designed to be as robust as possible; however, there will always be situations in the real world where disk corruption will occur. Hardware will fail eventually, power will be interrupted, and users will no doubt reboot the computer at just the wrong time.

The QNX filesystem has been designed to tolerate such catastrophes. It is based on the principal that the integrity of the filesystem as a whole should be consistent at all times—any single failure will damage no more than the contents of files opened for writing at the time of a crash.

Effectively, this means that the most common types of failures—power failure, reboot, CPU crash—usually leave the QNX filesystem intact.

If a crash occurs, we've provided recovery utilities that let you examine the filesystem to determine if any damage was done to files that were open for writing at the time of the crash. These same utilities can also fix such damage, and in many cases will completely restore the filesystem.

Sometimes the damage may be more severe. For example, it's possible that a hard disk will develop a bad block in the middle of a file, or worse, in the middle of a directory or some other critical block. Again, the utilities we've provided can help you determine the extent of such damage. You can often rebuild the filesystem in such a way as to avoid the damaged areas. In this case, some data will be lost, but with some effort, a large portion of the affected data may be recovered.

Overview of QNX 4 disk structure

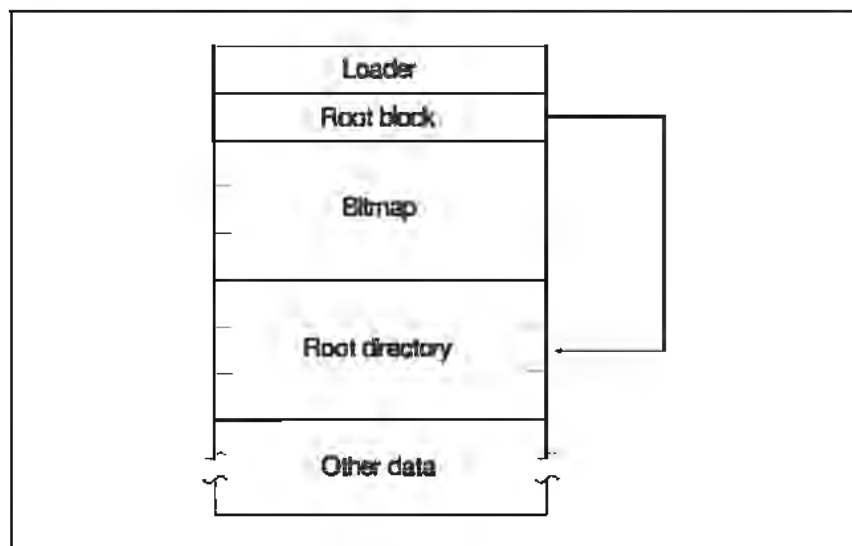
In this section, we describe how the QNX filesystem stores data on a disk. Reading this section should enable you to recognize and possibly correct filesystem damage if you ever have to rebuild a filesystem.

If you have a C development package, the header file `<sys/fsys.h>` contains the definitions for all terms used in this section.

For an overall description of the QNX filesystem, see *System Architecture*, Chapter 5, The Filesystem Manager.

Partition components

A QNX filesystem may be an entire disk (in the case of floppies) or it may be one of many partitions on a hard disk. Within a disk partition, a QNX filesystem contains the following components:



The following blocks are always found, in this order, on a QNX 4 disk partition:

- 1) loader block
- 2) root block
- 3) bitmap blocks
- 4) root directory

Loader block

The first block of a QNX partition, the loader block contains the bootstrap loader that loads the QNX OS into memory.

Root block

The second block of a QNX partition, the root block contains the directory entry for the root (`/`), the inode entries for the inode file, and a label field.

Overview of QNX 4 disk structure

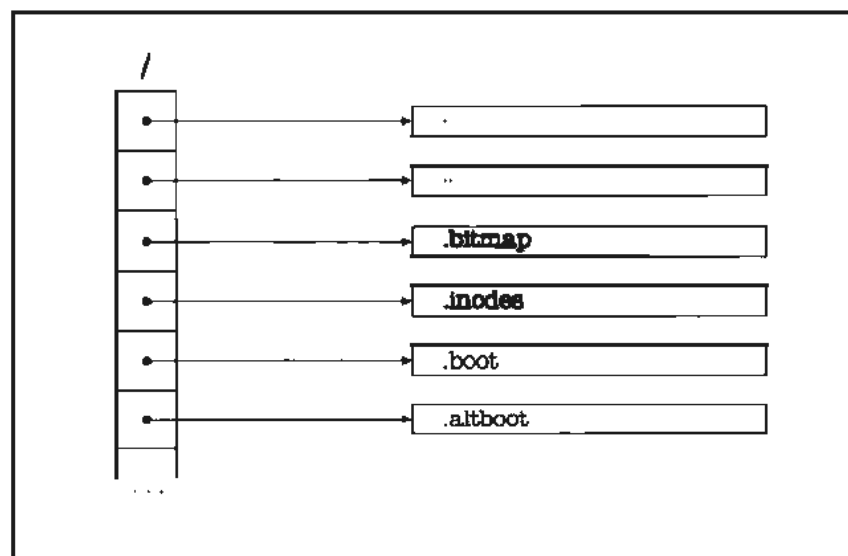
Bitmap blocks Several consecutive blocks follow the root block. The bitmap blocks form the bitmap for the QNX partition. One bit exists for each block on the partition, thus one bitmap block will be used for every 4096 disk blocks (corresponding to 2M of disk space).

If the value of a bit is zero, its corresponding block is unused. Unused bits at the end of the last bitmap block (for which there are no corresponding disk blocks) are turned on.

Bit assignments start with the least-significant bit of byte 0 of the first bitmap block—which corresponds to QNX block #1.

Root directory The root directory follows the bitmap blocks. The root directory is a “normal” directory (see “Directories,” below). It is initially created by the `dinit` utility with enough room for 32 directory entries (4 blocks).

As the following illustration shows, the root directory (/) contains directory entries for several special files that always exist in a QNX filesystem. The `dinit` utility creates these files when the filesystem is first initialized.



The type of directory entry is determined by the bits in the **d_status** field, as follows:

Bit 3 (FILE_LINK)	Bit 0 (FILE_USED)	Comment
0	0	unused directory entry
0	1	normal, used directory entry
1	x	link to an entry in / . inodes (which may be used or unused)

The first directory entry is always for the file “.” and includes a directory signature (“I♥QNX”). This entry refers to the directory itself by pointing to the entry within the parent directory that describes this directory.

The second entry is always for the “..” file. This entry refers to the parent directory by pointing to the first block of the parent directory.

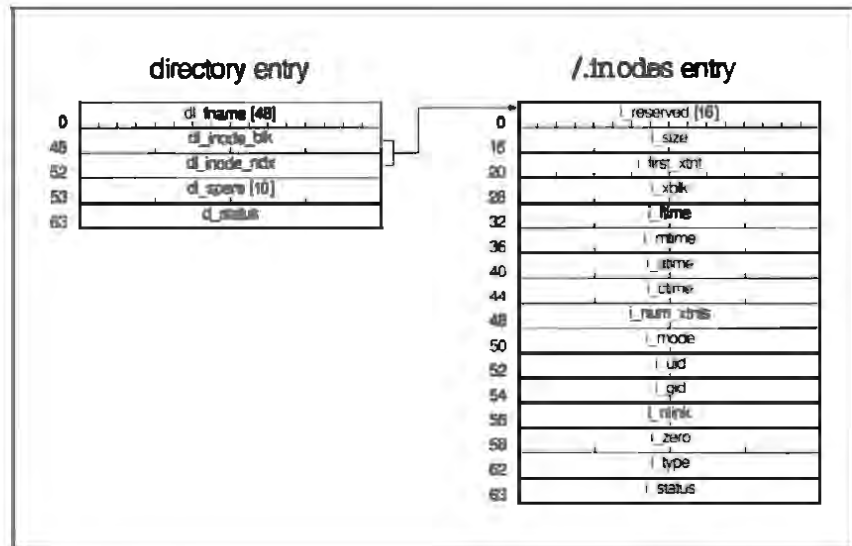
Every directory entry either defines a file or points to an entry within the **/ . inodes** file. Inode entries are used when the filename exceeds 16 characters or when two or more names are linked to a single file.

The first extent (if any) of a file is described in the directory/inode entry. Additional file extents require a linked list of extent blocks whose header is also in the directory/inode entry. Each extent block in the chain points to between 1 and 60 extents.

Links

Files with names greater than 16 characters long and *links* to other files are implemented with a special form of directory entry. These entries are identified with the **_FILE_LINK** bit (0x08) of the **d_status** field being set.

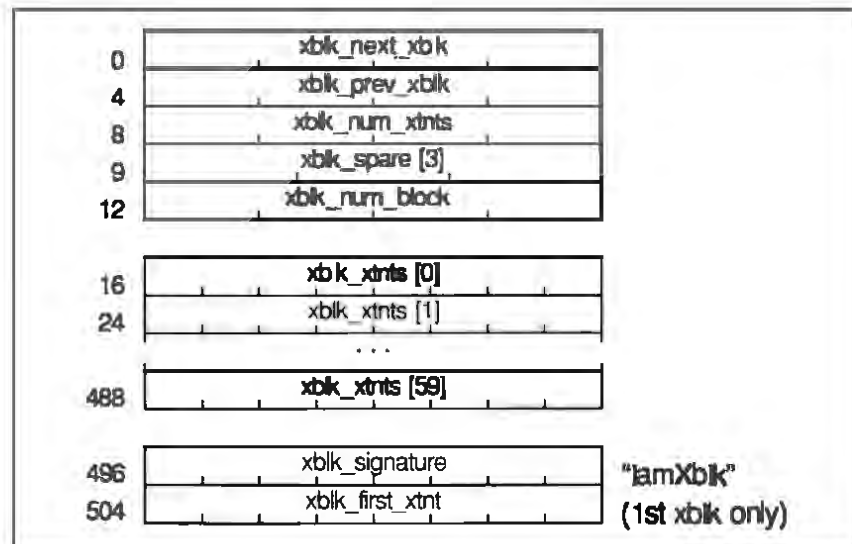
For these files, a portion of the directory entry is moved into the **/ . inodes** file.



Extent blocks

Extent blocks are used for any file that has more than a single extent. The directory entry **di_xbtk** points to one of these extent blocks, which in turn defines where the second and subsequent extents are to be found.

An extent block is exactly one 512-byte disk block with the following form:



Each extent block contains:

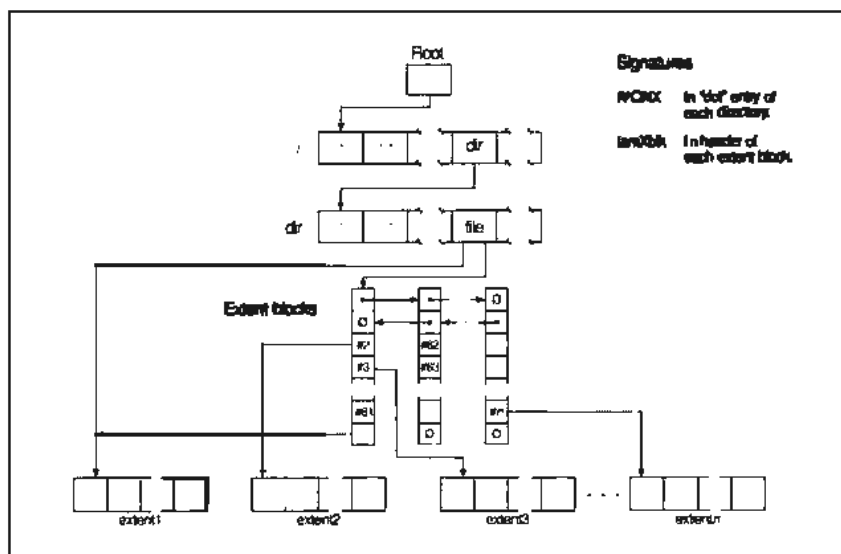
- forward/backward pointers
- a count of extents
- a count of all the blocks in all the extents defined by this extent block
- pointers and block counts for each extent
- a signature ("IamXblk")

The first extent block also contains a redundant pointer to the first file extent (also described within the directory/inode entry). This allows you to recover all data in the file by locating this block alone.

Files

Files or file extents are groupings of blocks described by directory/inode entries; they have no structure imposed on them by the QNX filesystem.

Most files in QNX 4 have the following overall structure:



File maintenance utilities

The following utilities let you create and maintain the disk/file structure:

- **fdisk**
- **dinit**
- **chkfsys**
- **zap**
- **spatch**

fdisk The **fdisk** utility creates and maintains the *partition block* on a hard disk. This block is compatible with other operating systems and may be maintained by other OS versions of **fdisk** (although ours has the advantage of recognizing QNX-specific information). If the partition loader is missing or damaged, **fdisk** can create it.



We recommend you keep a hard copy of the partition table information for every disk in your network.

dinit The **dinit** utility creates (but **fsys** maintains) the following:

- boot block
- root block
- **bitmap** blocks
- root directory
- inode file (**/ . inodes**)

chkfsys The **chkfsys** utility is your principal filesystem maintenance tool. This utility:

- checks the directory structure of an entire disk partition, reports any inconsistencies, and fixes them, if possible
- verifies overall disk block allocation
- writes a new **/ . bitmap**, upon your approval

The **chkfsys** utility assumes that the root block is valid. If the root block isn't valid, **chkfsys** will complain and give up—you'll need to try restoring the root block with the **dinit** utility.

zap The **zap** utility lets you remove files or directories from the filesystem without returning the used blocks to the free list. You might do this for several reasons, including:

- the directory entry is damaged
- two files occupy the same space on the disk (an error)

To remove zapped file entries and recover the used disk blocks, you use the **chkfsys** utility.

spatch The **spatch** utility lets you browse the raw disk and patch minor problems. You can sometimes cure transient disk problems by reading and writing the failing block with **spatch**.

Disk recovery procedures

Using **chkfsys**

The **chkfsys** utility is your principal tool for checking and restoring a potentially damaged filesystem. It can identify and correct a host of minor problems as well as verify the integrity of the entire disk system as a whole.

Normally, **chkfsys** requires that the filesystem be idle and that no files be currently open on that device. You'll have to shut down any processes that have opened files or that may need to open files while **chkfsys** is running.

To run **chkfsys** on an idle filesystem called **/dev/hd0t77**, you would simply type:

```
chkfsys /dev/hd0t77
```

The utility will scan the entire disk partition from the root down, building an internal copy of the bitmap and verifying the consistency of all files and directories it finds in the process. When it has finished processing all files, **chkfsys** will compare the internal bitmap to the bitmap on the disk. If they match, **chkfsys** is finished. If any discrepancies are found, **chkfsys** will—upon your approval—rewrite the bitmap with data consistent with the files it was able to find and verify.

In addition to verifying block allocation (bitmap), **chkfsys** attempts to fix any problems it finds during the scan. For example, **chkfsys** can:

- “unbusy” files that were written during a crash
- fix the file size in a directory entry to match the real data

When to run `chkfsys`

It’s a good idea to run **chkfsys** as part of your regularly scheduled maintenance procedures—this lets you verify that the data on your disk is intact.

It’s especially important to run **chkfsys** after a system crash so that you can identify those files, if any, that have been damaged.

If you “zap” any files for any reason—possibly because **chkfsys** suggested you do so—you’ll need to run **chkfsys** at some point to recover the *lost* blocks on the disk.

Using `chkfsys` in a live system

If you’re running an application that can’t afford downtime, you might not be able to shut down your system long enough to run **chkfsys**. In order to provide a *comprehensive* verification of the disk, **chkfsys** normally requires exclusive use of the filesystem. But you can identify potential problems in a live system by using a special, read-only mode of **chkfsys**:

```
chkfsys -f /dev/hd0t77
```

The **-fix** mode won’t be able to correct problems, but it will give you a feeling for the overall sanity of your filesystem.

If serious problems are detected that must be fixed right away, you can use another special mode—the “Rambo” mode:

```
chkfsys -R /dev/hd0t77
```

Remember: you can use “Rambo” mode *only* if you are sure that:

- the affected files are *not* presently opened
- the affected files will *not* be opened during the rebuild process
- any files currently opened will *not* grow during the process

Using Rambo mode will fix most errors, but since you're fixing the disk underneath a live filesystem with opened files, it will *not*:

- attempt to recover bitmap blocks
- check for block allocation errors
- attempt to rebuild the `/ . inodes` file



There is some risk to running **chkfsys** on a live system—both **chkfsys** and the filesystem are reading and possibly writing the same blocks on the disk. Also, the filesystem has internal cached data about files and directories that cannot be updated when **chkfsys** makes a change. Static changes, in place, on files or directories that **Fsys** doesn't currently have opened will *probably* not cause problems.

Recovering deleted files

If a file is accidentally deleted, it's sometimes possible to recover the file using the **zap** utility. To recover a file using **zap**, these conditions apply:

- the directory entry for that (now deleted) file must *not* be reused
- the disk blocks previously used by the file must *not* be reassigned to another file

Still, if you act quickly enough, or if you're lucky enough, there's a chance that the following command might restore the file:

```
zap -u directory
```

Recovering from a bad block in the middle of a file

Hard disks occasionally develop bad blocks as they age. In some cases, you might be able to recover most or even all the data in a file containing a bad block.

Some bad blocks are the result of power failures or of weak media on the hard disk. In these cases, sometimes simply reading then rewriting a block will “restore” the block for a short period of time. This may allow you to copy the entire file somewhere else before the block goes bad again. This procedure certainly can't hurt, and is often worth a try.

To examine the blocks within a file, you use the **spatch** utility. When you get to a bad block, **spatch** should report an error, but it may have actually read a portion of “good” bytes from that block. Writing that same block back will often succeed. At the same time, **spatch** will rewrite a correct CRC (Cyclic Redundancy Check) that will make the block good again (but with possibly incorrect data). You can then copy the entire file somewhere else, and then **zap** the previously damaged file. To complete the procedure, you mark the marginal block as bad (by adding it to the **/ .bad_blks** file), then run **chkfsys** to recover the remaining good blocks.

If this procedure fails, you can use the **spatch** utility to copy as much of the file as possible to another file, and then **zap** the bad file and run **chkfsys**.

What to do if your system will no longer boot

If a previously working QNX system suddenly stops working and will no longer boot, then one of the following may have occurred:

- the hardware has failed or the data on the hard disk has been damaged
- someone has either changed/overwritten the boot file or changed the system initialization file—these are the two most common scenarios

The following steps can help you identify the problem. Where possible, corrective actions are suggested.

Step 1

If you have a network...

Try booting from floppy or across the network

If you have a network to boot over, try booting your machine over the network. Once the machine is booted, you’ll need to log in as **root** and then start up a local filesystem:

Fsys &

If you don’t have a network...

If you don’t have access to a QNX network, then you’ll need to boot from the QNX boot floppy that was used to install your system onto the hard disk. The filesystem will already be running in this case. Note that you’ll need to log in as **root**.

Step 2 Start the hard disk driver

You now have to start the appropriate hard disk driver. For example, to start a disk connected to a standard Western Digital compatible controller, you would type:

```
fsys.wd &
```

This should create a block special file called `/dev/hd0` that represents the entire hard disk.

Step 3 Run `fdisk`

Running the `fdisk` utility will immediately give you useful information about the state of your hard disk.

The `fdisk` utility might report one of several types of problems:

Error reading block 1

Probable cause—If `fdisk` reports an error trying to read the partition block (which is the first physical block on the disk), then chances are that either the disk controller has failed or the hard disk has failed.

Remedy—If the disk is good, replacing the controller card *might* allow you to continue using the disk. Otherwise, you have little option but to replace the hard drive, reinstall QNX, and restore your files from backup.

Wrong disk parameters

Probable cause—If `fdisk` successfully reads block 1, but reports incorrect disk parameters (#heads, #cylinders, #sectors/cylinder), then your hardware has probably “lost” its information about this hard drive. The most probable cause is that the battery for the CMOS memory is running low.

Remedy—Rerunning the hardware *setup* procedure (or the programmable option select procedure on a PS/2) will normally clear this up. Of course, replacing the battery will make this a more permanent fix.

Bad partition information

Probable cause—If the disk size is reported correctly by **fdisk**, but the partition information is wrong, then the data in block 1 of the physical disk has somehow been damaged.

Remedy—Use **fdisk** to recreate the correct partition information. It's a good idea to write down or print out a hard copy of the correct partition information in case you ever have to do this step.

Step 4 Mount the partition as a block special file

At this point, you have verified that the hardware is working (at least for block 1) and that a valid partition is defined for QNX. You now need to create a block special file for the QNX partition itself:

```
mount -p /dev/hd0
```

This should create a volume called **/dev/hd0t77**, or its equivalent. If the partition information is correct, this shouldn't fail.

Step 5 Mount a QNX filesystem

At this point, you still don't know the health of the filesystem on the QNX partition. Your goal now is to run the **chkfsys** utility on the disk to determine—and possibly fix—the filesystem. But first you'll need to create a mount point for a QNX filesystem. Since the root (/) already exists (on a floppy or on a remote disk on the network), you should mount the local hard disk partition as a read-only filesystem with the name **/hd**:

```
mount /dev/hd0t77 /hd -r
```

Depending on the state of the QNX partition, the **mount** may or may not fail.



If you have booted from floppy and you don't suspect there's any damage to the filesystem on your hard disk (e.g. the system was unable to boot because of a simple error introduced in the boot file or system initialization file), you can change the root prefix to your hard disk partition at this point with the following command; this will resume normal operation of the system:

```
/hd/bin/prefix -R /=/hd/
```

If you run this command, you can skip the rest of this section.

If the **mount** fails

If the **mount** fails, the first portion of the QNX partition is probably damaged (since **Fsys** will refuse to mount what it considers to be a corrupted filesystem).

In this case, you can use the **dinit** utility to overlay sufficient good information onto the disk to satisfy **Fsys**:

```
dinit -hr /dev/hd0t77
```

The **-r** option tells **dinit** to rewrite:

- the root block
- the bitmap (with *all* blocks allocated)
- the constant portions of the root directory

After doing this, you'll need to rebuild the bitmap with **chkfsys**, even on a good partition.

You should now be able to reissue the **mount** command and once again try to create a mount point for a QNX filesystem called **/hd**.

Step 6 Run `chkfsys`

At least a portion of your QNX filesystem should now be accessible. You can use `chkfsys` to examine the filesystem and recover as much data as possible.

If you booted:	enter this command:
from floppy	<code>/hd/bin/chkfsys /dev/hd0t77</code>
across the network	<code>chkfsys /dev/hd0t77</code>

In either case, you should make note of any problems reported, and allow `chkfsys` to fix as much as it can. What you do next depends on the result of running `chkfsys`.

If the disk is unrecoverable

If, for any reason, your disk is completely unrecoverable, read the next section, “Recovering lost files and directories.” In some cases, you may need to reinstall QNX from floppy, and restore your disk from your backup files.

If significant portions of the filesystem are irreparably damaged, or important files are lost, then restoring from backup might be your best alternative.

If the filesystem is intact

If your filesystem is intact, yet the machine still refuses to boot from hard disk, then either of the following is probably damaged:

- the partition loader program in physical block 1
- the QNX loader in the first block of the QNX partition

To rewrite a partition loader, use `fdisk`:

```
fdisk /dev/hd0 loader
```

To rewrite the QNX loader, use `dinit`:

```
dinit -b /dev/hd0t77
```

You should now be able to boot your system.

Recovering lost files and directories

You may sometimes find that files or directories have been completely lost due to disk corruption. If after running **chkfsys** you know that certain key files or directories were *not* recovered, then you might be able to use **spatch** to recover some or all of this data.

Before attempting this, you should first familiarize yourself with the details of a QNX filesystem, as described in the section “**Overview of QNX 4 disk structure**” in this chapter. You should also study the documentation for the **spatch** utility (see the *Utilities Reference*).

The **lost+found** directory

The **spatch** utility can take a range of disk blocks and *link* them into the filesystem as a real file (or directory) under the **lost+found** directory.

A space for the **lost+found** directory always exists and has room for at least 126 entries on a hard disk.

After you use **spatch** to link all such pieces of the disk into the **lost+found** directory, you should immediately run **chkfsys** to rebuild the bitmap and verify the integrity of the filesystem that now includes these new files.

Once **chkfsys** validates the disk, you can safely move these files (or directory trees) to other directories on the disk.

It's a good idea to keep the **lost+found** directory empty so that there will be as much room as possible to recover files in any future disk recovery episodes.

Using **spatch**

The **spatch** utility has a special menu item called **Mark** that allows you to identify a range of disk blocks in a partition.

The following command would “spatch” the primary QNX partition:

```
spatch /dev/hd0t77
```

Remember that you must have adequate permissions to access the block special file.

The first time you select **Mark**, the **spatch** utility will mark the current disk block as the first block in a range of blocks. You can then move to the last block of the region, and select **Mark** a second time. This will identify the region of blocks to **spatch**. A special menu will now appear giving you the option of:

- removing all marks
- linking the marked blocks into **lost+found** as a file
- linking the marked blocks into **lost+found** as a directory
- copying the marked blocks to a file
- copying with append the marked blocks to a file
- ignoring the second mark, allowing you to select a different one

If you choose to link or copy the blocks into a file/directory, **spatch** will ask if you first want to perform a scan of the entire disk looking for an *xblk* that references this range of blocks. If you say yes, then **spatch** will perform the scan, and if it finds a block, it will give you the option of linking in all the extents defined by that *xblk* and any other associated *xblks*.

The *xblk* scanning allows potentially large disk files (or directories) to be recovered in one piece.

If you choose not to scan for *xblks*, or no such *xblk* is found, then a file/directory will be created that has only a single extent containing just these blocks.

If you copy, you should specify a file on another disk (filesystem).

Remember to always run **chkfsys** after linking or copying a number of files or directories.

The Quantum Update System

In this Chapter...

Introduction

Reporting bugs

Free software

Electronic mail

Conferencing system

About our hardware

How to phone us

Logging in

Using QUICS



Depending on the Support Service Plan you subscribe to, you may or may not be able to take advantage of all of the support services described here.

Introduction

The Quantum update system, called QUICS, runs 24 hours per day on a computer that you may call via modem to:

- download fixes and changes to the operating system, utilities, and libraries
- download a variety of free software from various sources
- send electronic mail to Quantum
- engage in conferences with other QNX users on a large variety of topics

Reporting bugs

Quantum's software refresh policy provides prompt response. For example, if you report a bug, perhaps in a utility or library routine, you can often download a revised version within hours of reporting the problem. We maintain the most current versions of all programs, including operating system modules, for download. Along with these, we provide a list of all changes made to the system. You may want to sign on once a month and make a log of this list.

Free software

We also maintain a growing list of free software that may be of interest to you. You'll find utilities, games, and megabytes of public domain source provided by ourselves and by our customers. This source can often help you understand some of the intricacies of QNX.

Electronic mail

You can use the private mail facility to communicate privately with Quantum employees or with any other user of QUICS.

Conferencing system

If you'd like to make a comment or ask a question that you feel other users may be interested in, then you'll want to use our conferencing system.

All Quantum developers are members of the conferencing system, as are most of our more active third-party developers. It's not unusual to pose a question and have several comments on it within an hour or so. You'll find topics on real time, hardware, utilities, C, and so on.

About our hardware

At the time of this printing, the QUICS computer supports seven modem lines and eight X.25 channels into the Datapac public packet switching network.

All modems support a variety of baud rates and data compression protocols. The following table shows the hardware configuration at the time of printing:

Hunt group	Baud rates	Modem type	Protocols
613-591-0934	38400 to 300	HST-Dual-Std	MNP5,ARQ,V42bis,V32
613-591-0935	38400 to 300	HST-Dual-Std	MNP5,ARQ,V42bis,V32
613-591-0940	38400 to 300	HST-Dual-Std	MNP5,ARQ,V42bis,V32,V32bis
613-591-1735	38400 to 300	HST-Dual-Std	MNP5,ARQ,V42bis,V32,V32bis
613-591-3009	38400 to 300	HST-Dual-Std	MNP5,ARQ,V42bis,V32,V32bis
613-591-3128	19200 to 300	Telebit	MNP5,PEP,V42bis
613-591-6093	19200 to 300	Telebit	MNP5,PEP,V42bis

The modems are on a hunt group with the following primary number:

(613) 591-0934 First number of the hunt group.

On the update system, we maintain an up-to-date configuration on the number of modems, their types, and phone numbers. You can read this when you give us a call.

To connect to QUICS via X.25, you have to make a pre-paid call or register with us for an X.25 account number (allowing you to make collect calls). Our X.25 network address is:

3020 85701416

Quantum's Datapac address

Datapac's network code

How to phone us

Since **qtalk** is shipped with every system, we describe how to contact us from within **qtalk**. We assume that your modem is connected to the QNX device `/dev/ser1`. If it isn't, you'll have to tell **qtalk** where your modem is.

Step 1 Set your serial port settings

Set your serial port settings to the following values. Refer to the section on configuring modems if you need help with this.

```
par=none
bits=8
```

You'll also need to set your baud rate to whatever rate your modem will support.

Step 2 Start **qtalk** and have your modem dial us

If you're in North America and have a Hayes-compatible modem, you can use **qtalk**'s dialing directory capability. The default dialing directory, `/etc/config/dial.dir`, provides an entry called **quics**, which contains Quantum's phone number. For more information, see the documentation on **qtalk** in the *Utilities Reference*.

```
qtalk quics [-m modem]
```

The `-m modem` option is needed only if your modem is *not* connected to the `/dev/ser1` device.

Logging in

Once you have successfully established communications with QUICS, you'll be asked to log in. If this is your first time signing onto QUICS, you should log in as user ID **newuser**.

Login: **newuser**

Next you'll be asked to enter information about you and your company as well as the name you wish to create as your own personal user ID.

Since there are many thousands of QUICS users, we ask that you avoid simple names (like your first name alone) when selecting your user ID. A good rule that works well is to select a name consisting of your first two initials, followed by your last name. For example, the user ID for "Johann Sebastian Bach" would be:

jsbach

From now on, you log in with your user ID (instead of **newuser**) each time you access QUICS.

Using QUICS

After logging in, you'll see the main menu. QUICS is a fully menu-driven system. We encourage you to explore the system to get familiar with its many capabilities.

By the way, one of the submenus lets you download a manual on how to use QUICS.

If at any time you experience difficulty, just give our technical support line a call:

(613) 591 0941

We at Quantum look forward to seeing you on QUICS!

Appendix A

The QNX Terminal

In this Appendix...

Standard console character set

QNX keyboard codes

Composed character input sequences

Terminal escape sequences

Introduction

QNX supports a logical terminal on the virtual consoles. That is, it can recognize certain escape sequences and special characters and can perform terminal control functions when the special sequences are encountered. A number of manufacturers have also developed real ASCII terminals that use the same protocol as the QNX logical terminal. This terminal is defined in the **terminfo** database as the **qnx** terminal.

Standard console character set

DECIMAL VALUE		0	16	32	48	64	80	96	112
	HEXA DECIMAL VALUE	0	1	2	3	4	5	6	7
0	0	BLANK (NULL)	▶	BLANK (SPACE)	0	@	P	'	p
1	1	☺	◀	!	1	A	Q	a	q
2	2	☹	↑	"	2	B	R	b	r
3	3	♥	!!	#	3	C	S	c	s
4	4	♦	¶	\$	4	D	T	d	t
5	5	♣	§	%	5	E	U	e	u
6	6	♠	■	&	6	F	V	f	v
7	7	•	↓	'	7	G	W	g	w
8	8	●	↑	(	8	H	X	h	x
9	9	○	↓	)	9	I	Y	i	y
10	A	◉	→	*	:	J	Z	j	z
11	B	♂	←	+	;	K	I	k	{
12	C	♀	└	,	<	L	\	l	!
13	D	♪	↔	—	=	M	J	m	}
14	E	♫	▲	.	>	N	^	n	~
15	F	☼	▼	/	?	O	—	o	△

DECIMAL VALUE		128	144	160	176	192	208	224	240
	HEXA DECIMAL VALUE	8	9	A	B	C	D	E	F
0	0	Ç	É	á	⋮	⋮	⋮	∞	≡
1	1	ü	æ	í	⋮	⋮	⋮	β	±
2	2	é	Æ	ó	⋮	⋮	⋮	Γ	≥
3	3	â	ô	ú	⋮	⋮	⋮	π	≤
4	4	ä	ö	ñ	⋮	⋮	⋮	Σ	∫
5	5	à	ò	Ñ	⋮	⋮	⋮	σ	∫
6	6	å	û	ä	⋮	⋮	⋮	μ	÷
7	7	ç	ù	o	⋮	⋮	⋮	τ	≈
8	8	ê	ÿ	ï	⋮	⋮	⋮	δ	°
9	9	ë	Ö	Γ	⋮	⋮	⋮	θ	•
10	A	è	Ü	Γ	⋮	⋮	⋮	Ω	•
11	B	ï	ç	½	⋮	⋮	⋮	δ	√
12	C	î	£	¼	⋮	⋮	⋮	∞	n
13	D	ì	¥	ì	⋮	⋮	⋮	φ	2
14	E	Ä	℞	«	⋮	⋮	⋮	€	■
15	F	Å	ƒ	»	⋮	⋮	⋮	∩	BLANK FF

QNX keyboard codes

		<i>Upper 4 bits</i>							
		0	1	2	3	4	5	6	7
Normal codes	0	NUL	DLE	SP	0	@	P	'	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	\$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	BEL	ETB	'	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	
	D	CR	GS	-	=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	BEL

Control codes

NUL - Ctrl @	BS - Ctrl h	DLE - Ctrl p	CAN - Ctrl x
SOH - Ctrl a	HT - TAB	DC1 - Ctrl q	EM - Ctrl y
STX - Ctrl b	LF - Ctrl j	DC2 - Ctrl r	SUB - Ctrl s
ETX - Ctrl c	VT - Ctrl k	DC3 - Ctrl e	ESC - Esc
EOT - Ctrl d	FF - Ctrl l	DC4 - Ctrl t	FS - Ctrl \
ENQ - Ctrl e	CR - Enter	NAK - Ctrl u	GS - Ctrl]
ACK - Ctrl f	SO - Ctrl n	SYN - Ctrl v	RS - Ctrl ^
BEL - Ctrl g	SI - Ctrl o	ETB - Ctrl w	US - Ctrl _

Function codes

Upper 4 bits												
		2	...	8	9	A	B	C	D	E	F	
EXTENDED mode only	0	Reserved key follows		Shift Tab	Shift F6	Home	Ctrl Home	Alt Home	Ctrl Enter	Shift Enter	Alt P	0
	1	Key Up		F1	Shift F7	Up	Ctrl Up	Alt Up	Alt F1	Alt a	Alt q	1
	2	Shift Down		F2	Shift F8	PgUp	Ctrl PgUp	Alt PgUp	Alt F2	Alt b	Alt r	2
	3	Shift Up		F3	Shift F9	Minus	Ctrl Minus	Alt Minus	Alt F3	Alt c	Alt s	3
	4	Ctrl Down		F4	Shift F10	LEFT	Ctrl LEFT	Alt LEFT	Alt F4	Alt d	Alt t	4
	5	Ctrl Up		F5	Ctrl F1	Five	Ctrl Five	Alt Five	Alt F5	Alt e	Alt u	5
	6	Alt Down		F6	Ctrl F2	RIGHT	Ctrl RIGHT	Alt RIGHT	Alt F6	Alt f	Alt v	6
	7	Alt Up		F7	Ctrl F3	Plus	Ctrl Plus	Alt Plus	Alt F7	Alt g	Alt w	7
	8	Keyboard ED code		F8	Ctrl F4	End	Ctrl End	Alt End	Alt F8	Alt h	Alt m	8
	9	Right Shift follows		F9	Ctrl F5	DOWN	Ctrl DOWN	Alt DOWN	Alt F9	Alt i	Alt y	9
SCAN mode only	A	reserved		F10	Ctrl F6	Pgdn	Ctrl Pgdn	Alt Pgdn	Alt F10	Alt j	Alt n	A
	B	reserved		Shift F1	Ctrl F7	Ins	Ctrl Ins	Alt Ins	Shift F11	Alt k	Alt Enter	B
	C	reserved		Shift F2	Ctrl F8	Del	Ctrl Del	Alt Del	Shift F12	Alt l	Shift BS	C
	D	reserved		Shift F3	Ctrl F9	PrtSc	Ctrl PrtSc	Alt *	SysRq	Alt m	reserved	D
	E	reserved		Shift F4	Ctrl F10	F11	Ctrl F11	Alt F11	Ctrl BS	Alt n	reserved	E
	F	reserved		Shift F5	Ctrl Tab	F12	Ctrl F12	Alt F12	Alt BS	Alt o		F
		2	...	8	9	A	B	C	D	E	F	

Note The above codes are *always* preceded by a hex FF code.

Codes in the range FF 21 to FF 29 are generated only when the console is in SCAN mode.

FF 20 is generated only when the console is in EXTENDED mode.

Composed character input sequences

Alt C ,	Ç	Alt E '	É	Alt a '	á	Alt g a	α
Alt u "	ü	Alt A o	æ	Alt i '	í	Alt g b	β
Alt e '	é	Alt A E	Æ	Alt o '	ó	Alt g g	Γ
Alt a ^	â	Alt o ^	ô	Alt u '	ú	Alt p i	π
Alt a "	ä	Alt o "	ö	Alt n ~	ñ	Alt g e	Σ
Alt a '	à	Alt o '	ò	Alt N ~	Ñ	Alt g s	σ
Alt a o	å	Alt u ^	û	Alt a _	ä	Alt g u	μ
Alt c ,	ç	Alt u '	ù	Alt o _	ë	Alt g t	τ
Alt e ^	ê	Alt y "	ÿ	Alt ? ?	¿	Alt o	Φ
Alt e "	ë	Alt • "	Ö	Alt + _	┐	Alt o -	θ
Alt e '	è	Alt U "	Ü	Alt _ +	└	Alt g w	Ω
Alt i "	ï	Alt c l	¢	Alt / 2	½	Alt g d	δ
Alt i ^	î	Alt L -	£	Alt / 4	¼	Alt o o	∞
Alt i '	ì	Alt Y =	¥	Alt ! !	¡	Alt o /	φ
Alt A "	Ä	Alt P r	₨	Alt < <	«	Alt E E	∈
Alt A o	Å	Alt f -	f	Alt > >	»	Alt U U	∩
Alt = =	≡						
Alt + -	±						
Alt > =	≥						
Alt < =	≤						
Alt - :	÷						
Alt - ~	≈						
Alt o .	•						
Alt . .	•						
Alt e q	√						
Alt g n	π						
Alt ^ 2	²						

Note To generate any of these characters, you:

- 1) press and release <Alt>
- 2) type the indicated two-character sequence

Terminal escape sequences

The following escape sequences and special characters cause a **qnx** terminal to perform specified control functions. Escape sequences are signalled by an escape character **<Esc>** (1B hex) followed by a set of ASCII characters. Any other character preceded by an **<Esc>** character, including **<Esc>**, loses its significance as a control function and is simply displayed as a graphic character on the screen.

In the following table, spaces are included in the escape sequences to make the sequences easier to read—these spaces don't actually form part of the sequences.

	If you want to:	use:
Setting character attributes	turn on inverse video mode (if a terminal supports color, this also swaps the foreground and background colors set by the <Esc> @ sequence)	<Esc> (
	turn off inverse video	<Esc>)
	turn on underline mode (available only on monochrome)	<Esc> [
	turn off underline mode	<Esc>]
	turn on blinking mode	<Esc> {
	turn off blinking mode	<Esc> }
	turn on high-intensity mode	<Esc> <
	turn off high-intensity mode	<Esc> >
Defining and moving the cursor	define the type of cursor, where <i>c</i> is the cursor type (specify ASCII character 0 for no cursor, 1 for a normal cursor, and 2 for a block cursor)	<Esc> y c
	position the cursor at the screen coordinates <i>r</i> and <i>c</i> , where <i>r</i> is the row and <i>c</i> is the column; both <i>r</i> and <i>c</i> are offset by 20 hex (a space). Therefore, <i>rc</i> = <i><space><space></i> is row zero, column zero.	<Esc> = r c or <Esc> Y r c
	move cursor up one line without wrap or scrolling	<Esc> A
	move cursor down one line without wrap or scrolling	<Esc> B

	If you want to:	use:
Defining and moving the cursor (cont'd)	move the cursor down one line (if the cursor is positioned on the last line of the screen, the screen scrolls up one line)	LF (0A hex)
	move cursor right one column without wrap	<Esc> C
	move cursor left one column without wrap	<Esc> D
	move cursor to the top left corner of the screen	<Esc> H
	turn auto line wrapping on	<Esc> h
	turn auto line wrapping off	<Esc> i
	move cursor up with wrap	<Esc> a
	move cursor down with wrap	<Esc> b
	move cursor right with wrap	<Esc> c
	move cursor left with wrap	<Esc> d
	reverse linefeed (the screen scrolls down when the cursor is positioned on the top line of the screen)	<Esc> I
	move the cursor to the first column of the current row	CR (0D hex)
	enter a nondestructive backspace (wraps to the end of the previous line when at the beginning of the line; wraps to the bottom right corner of the screen when at the top left corner of screen)	BS (08 hex)
Inserting, deleting, and clearing	insert line at cursor location	<Esc> E
	delete line at cursor location	<Esc> F
	clear to end of screen with spaces (uses the fill color specified with the <Esc> ! sequence on color terminals)	<Esc> J
	clear to end of line with spaces (uses the fill color specified with <Esc> ! on color terminals)	<Esc> K

	If you want to:	use:
	clear the entire screen to spaces (uses the fill color specified with <Esc> ! and moves the cursor to the top left corner)	FF (0C hex)
	insert space at cursor position	<Esc> e
	delete character at cursor position	<Esc> f
	repeat the character <i>c</i> the number of times specified by <i>n</i> , where <i>n</i> is offset by 20 hex	<Esc> g n c
Saving attributes	save color and attribute information (This overrides any previously saved information; the cursor position isn't saved. QNX programs, such as editors that change the colors, assume that a default has been set on entry and use <Esc> R to restore the default on exit.)	<Esc> S
	restore the color and attribute information saved with <Esc> S (the cursor position isn't affected)	<Esc> R
Setting color	define the foreground and background color of displayed characters, where <i>fc</i> is the color of the cursor, and <i>bc</i> is the fill color—both <i>fc</i> and <i>bc</i> represent a single character taken from the color table below. This has no effect on a monochrome terminal or display.	<Esc> @ fc bc
	set the fill color that is used both for blank lines created by scrolling and for the entire screen when clearing with a form feed. This has no effect on a monochrome terminal or display.	<Esc> ! fc bc
Generating a tone	generate a short tone on the system console or ring the bell on a terminal	BELL (07 hex)

Color codes

You can use these color codes in the above escape sequences:

Color	ASCII character
black	0
blue	1
green	2
cyan	3
red	4
magenta	5
brown	6
white	7

Appendix B

Time Zone Rules

This appendix describes the various rules used for establishing time zones and time changes (e.g. daylight savings time). You'll find a time zone chart* and a map at the end of this appendix.

You specify time zone information via the **TZ** environment variable. This variable contains a coded string that lets you specify the time zone, as well as the rules for changing to and from daylight savings time (if used in your area). The value of **TZ** has the following form (spaces have been inserted for clarity):

std offset dst offset, rule

where:

<i>std</i>	standard time
<i>offset</i>	offset of standard time from UTC time
<i>dst</i>	daylight savings time
<i>offset</i>	offset of daylight savings time from UTC time
<i>rule</i>	specifies when to change between standard time and daylight savings time.

In more detail, **TZ** is coded as follows:

TZ=std[offset][dst[offset] [, start [/time] , end [/time]]]]

* The time zone chart was excerpted from *OAG® Desktop Flight Guide—Worldwide Edition* (May 1992, Vol. 17, No.3 pp. 1408-1409), published by Official Airline Guides, Inc., Oak Brook, Illinois.

The components of the **TZ** environment variable string are defined as follows.

std and dst Defines a string used to identify the time zone (e.g. **EST**). All characters are allowed, except for:

- a leading colon (:)
- digits
- a comma (,)
- a minus sign (-)
- a plus sign (+),
- ASCII NUL

std defines the name of the standard time zone and *dst* defines the name of the daylight savings time zone. If *dst* is omitted, then daylight savings time isn't used in this locale.

offset Indicates the value you must add to the local time to arrive at UTC. The form for *offset* is:

***HH* [: *MM* [: *SS*]]**

The minutes *MM* and seconds *SS* are optional. The hour *HH* is required and can be a single digit. You must specify an hour between 0 and 24. You can specify minutes and seconds, if required, between 0 and 59. Time zones east of the Prime Meridian (through England) should be preceded by a minus (-); time zones west of the Prime Meridian can be optionally preceded by a plus (+). An *offset* following *std* is required, but the *offset* following *dst* may be omitted. In this case, daylight savings time is assumed to be one hour ahead of standard time.



Note that POSIX deviates from the convention used in international time charts for showing offsets from UTC. In most time zone charts and maps, offsets east of the Prime Meridian are preceded by +, those west are preceded by -. You should therefore invert the sign of any numbers you take from the chart or map provided in this appendix. For example, U.S.A. Eastern Time would be +5 while Japan would be -9.

rule Specifies when to change from standard time to daylight savings time and back again. If you don't specify ***rule***, QNX assumes standard time is in effect. The ***rule*** has the following format:

date/time, date/time

where the first ***date*** specifies when the change to daylight savings time takes place and the second ***date*** specifies when the change to standard time takes place. Each ***time*** field specifies the exact time at which the change shall be made. The ***date*** field has one of the following formats:

- Jn*** Specifies the date as a Julian day where $1 \leq n \leq 365$. This format doesn't allow leap days to be referenced (e.g. February 28 is day 59 and March 1 is day 60; it's impossible to refer to February 29).
- n*** This specifies the date as a zero-based Julian day where $0 \leq n \leq 365$. Leap days are counted and you can refer to February 29.
- Mm.w.d*** This specifies the *d*th day ($0 \leq d \leq 6$) of week *w* of month *m* of the year ($1 \leq w \leq 5$, $1 \leq m \leq 12$, where week 5 means "the last *d* day in month *m* that may occur in either the fourth or fifth week"). Week 1 is the first week of month *m* in which the *d*th day occurs. Day zero is Sunday.

The time has the same format as ***offset*** except that no leading sign (+ or -) is allowed. The default, if ***time*** isn't specified, is 02:00:00.

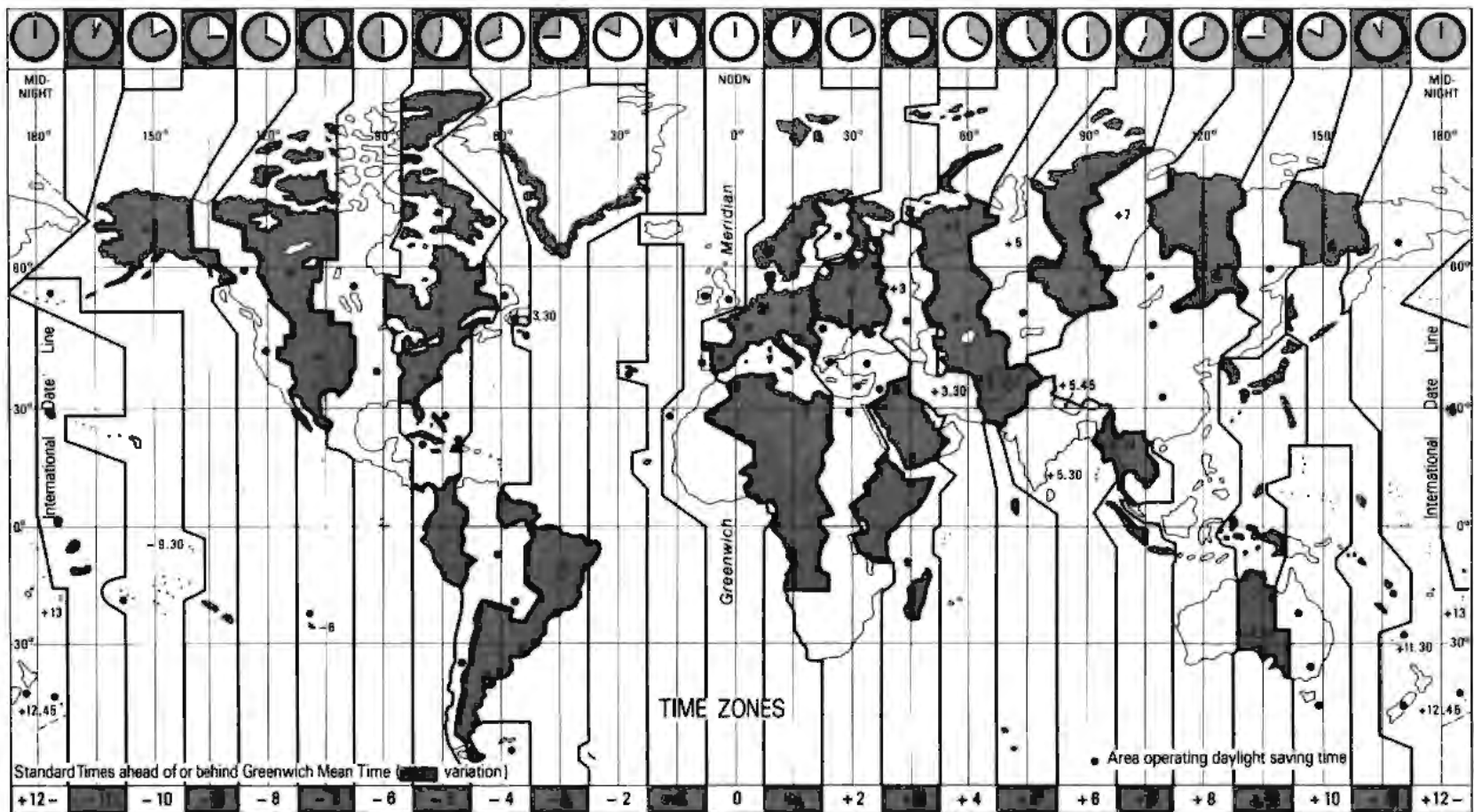
International Time Chart

- Δ - Affixes and parts of letters do not observe DST.
- * - Privates of Saskatchewan and private Canadian cities remain on Standard Time all year.
- ☆ - Except Sydney, N.S.W.: Adopts South Australian time.
- † - Certain States of Brazil do not observe DST.
- - Estimated Data Based On Previous Year.
- - Former USSR Republics use Moscow GMT. For Local Time Notes, see individual city listings in Africa Schedule, Section.

ASIAN SCHOONER SECTOR									
Country	Standard Time		Daylight Saving Time		Country	Standard Time		Daylight Saving Time	
	Hours from GMT	Time of 1988 Set	Hours from GMT	Effective Period (Start and End Day)		Hours from GMT	Time of 1988 Sunset	Hours from GMT	Effective Period (Start and End Day)
Peru	+5	07 00			Yugoslavia*	+3	15 00	+4	Mar. 20—Sep. 25, 1988
Philippine New Edition (And Bougainville Is.)	+10	06 00			Yemen	+3	16 00		
Poland	+1	05 00	+3	Oct. 8, 1984—Feb. 28, 1985	Thailand	+7	16 00		
Portugal	+1	01 00			Togo	GMT	12 00		
Romania	+2	03 00			Tonga	+13	01 00		
Russian Republic of	+3	20 00			Tuvalu and T. Islands	+4	08 00		
Russia	+3	13 00	+2	May 10—Sep. 28, 1988	Turkey	+3	13 00		
Saudi Arabia	+3	11 00	GMT	May 10—Sep. 28, 1988	Turkmenistan*	+5	14 00	+5	Mar. 10—Sep. 25, 1988
Seychelles	GMT	12 00			Turks and Caicos Is.	+8	13 00		
Singapore	GMT	12 00	+1	May 10—Sep. 28, 1988	Turkmenistan*	+6	07 00	+4	Apr. 5—Oct. 14, 1988
South Africa	+2	08 00			Ukraine	+2	23 00		
South Korea	+9	15 00			Uzbekistan	+5	15 00		
Spain	+2	16 00			United Arab Emirates (Abu Dhabi, Dubai, Sharjah, Ras Al Khaimah)	+4	16 00	+3	Mar. 10—Sep. 25, 1988
Spain (Balearic Islands)	+2	15 00	+3	Mar. 10—Sep. 28, 1988	Uzbekistan	GMT	12 00	+1	Mar. 10—Oct. 24, 1988
Spain (Canary Islands)	+2	14 00	+4	Mar. 10—Sep. 28, 1988	Uzbekistan	+5	07 00	+4	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	08 00	+5	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	09 00	+6	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	10 00	+7	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	11 00	+8	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	12 00	+9	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	13 00	+10	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	14 00	+11	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	15 00	+12	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	16 00	+13	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	17 00	+14	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	18 00	+15	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	19 00	+16	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	20 00	+17	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	21 00	+18	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	22 00	+19	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	23 00	+20	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	24 00	+21	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	25 00	+22	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	26 00	+23	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	27 00	+24	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	28 00	+25	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	29 00	+26	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	30 00	+27	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	31 00	+28	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	32 00	+29	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	33 00	+30	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	34 00	+31	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	35 00	+32	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	36 00	+33	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	37 00	+34	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	38 00	+35	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	39 00	+36	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	40 00	+37	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	41 00	+38	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	42 00	+39	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	43 00	+40	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	44 00	+41	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	45 00	+42	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	46 00	+43	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	47 00	+44	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	48 00	+45	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	49 00	+46	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	50 00	+47	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	51 00	+48	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	52 00	+49	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	53 00	+50	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	54 00	+51	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	55 00	+52	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	56 00	+53	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	57 00	+54	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	58 00	+55	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	59 00	+56	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	60 00	+57	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	61 00	+58	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	62 00	+59	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	63 00	+60	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	64 00	+61	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	65 00	+62	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	66 00	+63	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	67 00	+64	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	68 00	+65	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	69 00	+66	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	70 00	+67	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	71 00	+68	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	72 00	+69	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	73 00	+70	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	74 00	+71	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	75 00	+72	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	76 00	+73	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	77 00	+74	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	78 00	+75	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	79 00	+76	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	80 00	+77	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	81 00	+78	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	82 00	+79	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	83 00	+80	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	84 00	+81	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	85 00	+82	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	86 00	+83	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	87 00	+84	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	88 00	+85	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	89 00	+86	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	90 00	+87	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	91 00	+88	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	92 00	+89	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	93 00	+90	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	94 00	+91	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	95 00	+92	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	96 00	+93	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+10	06 00			Uzbekistan	+5	97 00	+94	Apr. 5—Oct. 24, 1988
Approximately every 12° from 30° 30' E to 170° 30' W, and 10° S	+								



Because POSIX deviates from the convention used in international time charts for showing offsets from UTC, remember to invert the sign of any of the numbers in this chart.



Because POSIX deviates from the convention used in international time charts for showing offsets from UTC, remember to invert the sign of any of the numbers in this map.

A

- access conflicts, resolving (RCS) 146
- access control utilities
 - login** 62
 - newgrp** 63
 - passwd** 62, 64
 - running with superuser permissions 68
 - su** 62
 - see also* password; permissions; resources; security
- access fields/modes
 - see* permissions
- accounting file 71 – 73
 - clearing 72
 - compressing 72
 - creating 71
 - discarding information 71
 - format 71
 - interpreting 71
 - spooling commands 126, 131
 - superuser access 71
 - typical example 72
- accounts
 - creating 62
 - creating without specifying home directory, shell, or password 63
 - deleting 63
 - accounts (cont'd)
 - setting up 61 – 73
 - see also* password; security
- adapter cards
 - see* network
 - see also* serial devices
- alloc space, setting in buildfile 141
- .altboot** file 143, 169
 - see also* booting
- alternate boot, selecting 36, 38
 - see also* **.altboot** file
- altesysinit** file 38
 - see also* system initialization file
- appending output to a file 16
- archives 159
 - compressing data 162
 - cpio** utility 159, 163
 - examples 163 164
 - pax** utility 159, 163
 - tar** utility 159, 163
 - vol** utility 159, 163
 - see also* backups
- Arcnet
 - drivers/cards, selecting physical node ID 51
 - Ethernet nodes, adding to Arcnet 112 – 113
 - see also* network

arrow keys 4 – 5
 auditing
 see accounting file

B

Backspace key 4
 backups
 archive backups 159
 compressing data 162
 filesystem backups 160
 fixed disks 162
 floppies 160
 formats 158
 media types 160
 removable disks 162
 tape 161, 163
 when to back up 158
 see also archives
 base-level services (*sysinit* file) 39
 baud rate
 locked rate, specifying 87
 specifying 40, 82
 bell, ringing on a terminal 201
 bitmap blocks 168
 creating 173
 .*bitmap* file 169
 rewritten by *chkfsys* 173
 bits
 data bits 81
 DOS attribute bits 100
 _FILE_LINK bit 170
 mode bits 65
 parity bits 81
 stop bits 81

blocks
 allocation verified by *chkfsys* 173
 bad block in middle of file,
 recovering from 176
 bad block on backup media 162
 bitmap blocks 168
 creating 173
 boot block, creating 173
 disk blocks, recovering 174
 DOS boot parameter block (BPB) 97
 error reading block 1 178
 examining 176
 extent blocks 170 – 171
 identifying 182
 loader block 167
 creating 173
 lost blocks, recovering 175
 partition block, creating 173
 restoring 176
 root block 167
 creating 173
 restoring 173
 boot block
 creating 173
 boot parameter block (DOS) 97
 boot ROM, for Ethernet & Token Ring cards 104
 boot server
 associating nodes with each server 105
 build file, selecting 57
 card, installing 54
 configuring 54 – 57
 customizing 56
 defined 52
 licenses, installing 54
 licenses, *name1oc* required 56
 mapping between logical & physical
 node IDs 57
 multiple boot servers 53, 104 – 105
 netboot file, modifying 57
 netboot, starting 56

- network manager & drivers, starting 55
 - selecting 52 – 53
 - sysinit.node** file, modifying 55
 - see also* network
 - booting
 - .altboot** file 143, 169
 - alternate boot, selecting 36, 38
 - .boot** file 143, 169
 - bottleneck, reducing 53, 104
 - broadcast booting 104
 - netboot** and 105
 - with a QNX ROM 104
 - diskless workstation 58, 109, 111, 144
 - from Ethernet cards 104
 - failure to boot 177 – 181
 - floppy boot 34
 - hard disk boot 39, 43, 46, 141
 - processes required 142
 - initial boot 36
 - makefile for disk booting 142
 - makefile for network booting 144
 - netboot** configuration file 57, 105, 111
 - network boot 44, 104, 109, 111, 141, 144
 - network card used for booting over net 109
 - networked machines from their hard disks 46
 - rebooting 9
 - specific boot method 104
 - from Token Ring cards 104
 - warm boot 9
 - what happens when you boot 36
 - what to do when system no longer boots 177 – 181
 - workstation 58, 109, 111, 144
 - see also* images; network
 - broadcast booting 104
 - netboot** utility acting as primary, secondary, or tertiary server 105
 - with a QNX ROM 104
 - buffer
 - output buffer, specifying 77
 - bugs, reporting 186
 - build file
 - constructing 140 – 144
 - selecting for your network media 57, 112
 - see also* images
 - buildqnx** utility 140
- ## C
- canonical mode
 - editing keys 4
 - chaining queues (spooling) 124, 131
 - characters
 - attributes, setting 199
 - composed character input sequences 198
 - console character set 194
 - deleting 4, 200
 - displayed characters, changing 7
 - DOS to QNX mapping 99
 - DOS, invalid characters 98
 - fonts, changing 7
 - inserting 4
 - international 9, 194, 198
 - mathematical 194, 198
 - overwriting 4
 - pattern-matching 19
 - QNX to DOS mapping 99
 - quoting special 14
 - unrecognizable 89
 - checking in/out
 - see* RCS
 - chkfsys** utility 173
 - Rambo mode 175 – 176
 - using 174
 - using in a live system 175
 - when to run 175
 - ci** command (RCS) 147 – 150

- clock
 - getting time from 39, 43
 - starting 42
 - see also* time zones
- co command (RCS) 147 – 148, 150
- codes (keyboard)
 - see* keyboard
- color, setting 201
- command interpreter
 - see* commands; Shell
- command lists 20
- commands
 - basics 11
 - common commands, placing in a shell file 39
 - entering 14
 - executing on another node or tty 17
 - file management commands 29
 - help messages 18
 - input, ending 10
 - key symbols 13
 - multiple commands, entering 20
 - multiple files, working with 19
 - options, grouping 14
 - quoting special characters 14
 - recalling 5
 - remote execution 17, 108
 - syntax 12 – 15
 - usage messages 18
- communicating across networks 106
- communication sessions
 - controlled by DTR and DSR 80
- comparing files 29
- composed characters 9
 - input sequences 198
- compressed data
 - recovering 162
- compression utilities 162
- concatenating files 29
- conferencing system 187
 - see also* QUICS
- connecting serial devices 82
 - see also* serial devices
- console driver
 - starting 39, 76
- consoles
 - character set 194
 - fonts, changing 7
 - generating a tone 201
 - logging in on an unused console 7
 - login, starting 41
 - multiple consoles, using 7
 - starting 39
 - switching 6
 - disabling 6
 - terminating sessions 7
 - tone, generating 201
 - virtual consoles, switching 6
 - won't function 144
 - see also* keyboard; ~~qnx~~ terminal; terminals
- control programs (spooling)
 - see* spooling
- converting DOS files to QNX format 98
- cooked mode
 - editing keys 4
- coordinated universal time (UTC)
 - see* time zones
- copy-in and copy-out filters
 - see* spooling
- copying files 29, 164
 - to a system in a different time zone 43
- counting words in a file 29
- cp utility 164
- cpio utility 159 – 160, 163
- creating
 - directories 29
- cron server
 - selecting a machine as 42
- current process
 - changing group ID of 63

current working directory 26
 changing 29
 printing 29
 see also directories
 cursor
 defining 199
 moving 4, 199
 customizing services for each machine
 see images
 see also system initialization file

D

data
 bits 81
 can't be sent or received 89
 compressing 162
 lost occasionally 90
 received/transmitted only when another
 serial port is used 89
 transmission of, slowing down 81
 unrecognizable characters 89
 date
 establishing 43
 see also clock; time zones
 daylight savings time
 changing to or from 203, 205
 time zone, defining 204
 see also time zones
 dead keys 9
 debugger, invoking & disabling 8
 Del key 4
 deleted files, recovering 176
 deleting
 accounts 63
 characters 4, 200
 directories 29
 files 29
 nodes 111

dequeues
 see targets
 device drivers
 Dev.con 39, 76
 Dev.par 40, 76
 Dev.ptx 76
 Dev.ser 40, 76 – 77
 floppy driver, starting 41
 starting 39, 76
 see also system initialization file
 Device Manager
 starting 39, 76
 devices
 output devices, reinitializing between
 spool jobs 121
 as shared resources 118
 see also parallel devices; serial devices
 dinit utility 173, 180
 directories
 access based on permissions 66
 changing 29
 commonly used commands 29
 contents, listing 29
 creating 29
 current working directory 26
 printing 29
 deleting 29
 DOS directories 95
 dot (directory link) 170
 dot dot (directory link) 170
 entries 169
 home directory 26
 location 28
 returning to 29
 links 170
 listing contents 29
 lost directories, recovering 182 – 183
 lost+found directory 182
 moving through 25
 permissions 66

directories (cont'd)

- RCS directory, creating 149
- recovering if lost 182 – 183
- removing 29
- removing without returning used blocks 174
- root directory 25
 - contents of 168
 - creating 173
- structure 169
 - checking 173
- see also* files; root directory

disk booting

- see* booting

disk images 142

- see also* images

disk partitions 167

- see also* partitions

diskless workstation

- see* workstation

disks

- backing up to or restoring from 160, 162
- bad blocks, restoring 176
- block allocation verified by **chkfsys** 173
- browsing 174
- DOS drives, adopting 96
- maintenance utilities 173
- patching minor problems 174
- recovering 165 – 183
- recovery procedures 174 – 176
- structure 166 – 172
- see also* partitions

displayed characters

- changing 7

DOS

- attribute bits 100
- binary files 98
- boot parameter block (BPB) 97
- converting files to QNX format 98
- devices 97
- directories 95

DOS (cont'd)

- directories, creating, deleting, & reading 95
 - disks, accessing via QNX programs 95
 - DOS-to-QNX character and name mapping 99
 - DOS-to-QNX permission-bit translations 100
 - drives, adopting 96
 - drives, getting info about 96
 - file allocation table (FAT) 97
 - file format 98
 - file ownership 101
 - filename characters, invalid 98
 - filenames
 - invalid 98
 - mapping & translating 99
 - seeing 96
 - files, accessing via QNX programs 95
 - filesystems, setting up 95 – 101
 - image of DOS diskette or partition, creating 97
 - ownership of files 101
 - partitions supported 97
 - permissions 100
 - QNX-to-DOS character and name mapping 98 – 99
 - QNX-to-DOS attribute-bit translations 100
 - text files
 - converting 98
 - structure of 98
 - versions supported 97
 - volume labels 100
- Dosfsys** filesystem manager
- access to DOS files 95
 - DOS version support 97
 - drives, adopting 96
 - error codes 101
 - invocation modes 95
 - name adoption 96
 - starting 96
 - starting in **sysinit** file 42
 - terminating 96, 101
- dot**(directory link) 170

dot command 39
dot dot (directory link) 170
down arrow key 5
downloading from QUICS
 see QUICS
drivers
 see device drivers
 see also network
drives
 see disks
DSR lines 80
DTR lines 80

E

editing keys 4
effective group and user IDs 64
 changing 64
electronic mail
 sending to Quantum employees or other
 users 186
 see also QUICS
embedded images 144
emu07 utility 40
encrypted passwords 70
End key 4
Enter key 4
entering
 input 4
 multiple commands 20
environment variables, defining 41
essential network drivers 109
/etc/acalog file
 see accounting file
/etc/config/lpsrvr file
 see spooling, setup file
/etc/syslog file 73
 see also accounting file

Ethernet
 drivers/cards, booting from 104
 drivers/cards, physical node ID,
 determining 51
 fault-tolerant network 115
 nodes, adding to an Arcnet network 112 – 113
 see also network
executable file
 running with privileges of its owner &
 group 68
execute permission 66
expansion, filenames 19
EXTENDED mode 197
 see also qnx terminal
extents
 structure 172
 where extents are stored 170 – 171

F

FAT table (DOS) 97
fault-tolerance (network) 53, 104, 106
fault-tolerant Ethernet network 115
fdisk utility 173
file maintenance utilities 173
 chkfsys 173 – 175
 dinit 173, 180
 fdisk 173
 spatch 174, 177, 182
 zap 174, 176
 see also **chkfsys**
_FILE_LINK bit 170
filenames
 DOS to QNX mapping 99
 DOS, invalid filenames 98
 expansion 19
 longer than 16 characters 170
 maximum length 24
 portable 24

filenames (cont'd)

- QNX to DOS mapping 99
- suffixes 24
- valid & invalid characters 24

files

- access based on permissions 66
- access conflicts, resolving 146
- accessing across the network 25
- accounting file 71 73
- appending 16
- backing up (see also backups) 157 – 163
- blocks, examining & restoring 176
- changes to, logging (RCS) 146 – 147, 151
- commonly used commands 29
- comparing 29
- concatenating 29
- copying 29, 164
- copying to a different time zone 43
- counting words, lines, or bytes 29
- deleting 29
- deleting without returning used blocks 174
- displaying contents 29
- DOS binary files 98
- DOS text files 98
- DOS text files, converting to QNX 98
- executable file running with privileges of its
 - owner & group 68
- extent blocks 171
- finding 29
- group ID, changing 67
- identifying 146, 151
- links 170
- listing 29
- locating 25 – 26
- locking (RCS) 148, 150
- maintenance utilities 173
- mode bits, changing 67
- moving 29
- multiple, working with 19
- naming 24

files (cont'd)

- network-wide access 25
- ownership (DOS) 101
- ownership, changing 67
- permissions 65
 - changing 67
- printing (see also spooling) 119
- RCS files, storing 149
- recovering 165 183
 - if deleted 176
 - if lost 182 – 183
- remote access 25
- removing 29
- removing without returning used blocks 174
- renaming 29
- revising 146
- setgid and setuid 67 – 68
- sorting 29
- string patterns, searching for 29
- structure 172
- temporary files, default location 28
- user ID, changing 67
- working file, protecting (RCS) 149
- see also* DOS; directories; extents;
 - filenames; RCS (Revision Control System)

filesystems

- backups 160
- configuring for network 93
- how data is stored on disk 166
- naming 25
- physical filesystems, defined 92
- restoring 174
- setting up second system 91 – 101
- typical filesystem 27 – 28
- see also* DOS; partitions; pathname space,
 - partitioning; recovering

filters (spooling)

- see* spooling

fixed disks, backing up to or restoring from 162

- see also* backups

floating point emulator 40
 floppies
 backing up to or restoring from 160
 block special file, accessing 41
 floppy driver
 starting 41
 flow control (software)
 enabling & locking 85
 fonts, changing 7
 free software 186
freeze utility 162
 full-screen programs
 terminfo database 88

G

global name server
 selecting a machine as 40
 Greenwich Mean Time
 see time zones, UTC
 group IDs 64
 changing 63, 67
 effective group IDs 64
 changing 64

H

hard disk booting
 see booting
 hard disks
 see disks
 hardware adapters
 see serial devices
 hardware handshaking
 enabling & locking 82
 heap sizes, setting 141

home directory 26
 location 28
 returning to 29
 Home key 4

I

I/O addresses (hardware adapters) 77 78
 identifying files (RCS) 151
 IDs
 see group IDs, logical node IDs,
 logical network IDs, physical node IDs,
 process IDs, and user IDs
 images
 .boot file, copying to 143
 build file, constructing 140 – 144
 buildqnx utility 140
 creating through **buildqnx** or the
 Makefile 140
 disk images 142
 embedded images 144
 hard disk booting, processes required 142
 heap size, setting 141
 image file location 140
 local memory, allocating 141
 makefile for disk booting 142
 makefile for network booting 144
 network images 144
 of DOS diskettes or partitions 97
 priority of processes, setting 141
 processes, selecting 141
 see also system initialization file
 incoming & outgoing calls on same serial line 87
 inodes
 .inodes file 169 – 170

input

- deleting 4
- end of input, indicating 10
- entering 4
- lines, editing 4
- lines, recalling 5
- maximum length of input line 5
- reading from a file 16
- redirecting 16
- submitting line of input 4

Ins key 4

inserting

- characters 4

installing

- additional software from Quantum and other vendors 35
- QNX onto a hard disk 34
- see also* network

international characters

- entering 9, 194, 198

international keyboards 9, 42

interrupt generated by each UART 77

inverse video mode 199

K

kbd utility 42

kedit utility 42

keyboard

- codes (**qnx** terminal) 196
- console fonts, changing 7
- consoles, switching 6
- conventions used in this manual vii
- custom layouts, creating 42
- dead keys 9
- debugger, invoking 8
- end of input, indicating 10
- EXTENDED mode (**qnx** terminal) 197
- international keyboards 9, 42

key chords vii

killing a process 8

line-editing keys 4, 10

non-USA keyboards 42

rebooting system 9

recalling commands 5

SCAN mode (**qnx** terminal) 197

summary of functions 10

suspending/resuming output 8

won't function 42, 144

killing a process 8

L

left arrow key 4

licensing 45 – 47, 54

activating new licenses 46

application licensing 45

based on number of logical nodes in the network 45, 50

expanding your license 46

license utility 46

network licensing, **nameloc** required 47, 56

operating system licensing 45

see also network

line configuration

changing 40

line-oriented input

entering 4

links 170

_FILE_LINK bit 170

symbolic links, using to partition pathname space 95

listing contents of a directory 29

loader block 167

creating 173

see also partitions

local memory allocation 141

locating
 files 25 – 26
 locking files (RCS) 148, 150
 see also RCS
 log file
 see accounting file
 logging changes to files (RCS) 146 – 147, 151
 logical network IDs
 assigning 50
 for multiple network links 110
 see also network
 logical node IDs
 assigning 50, 52, 56
 see also network
 login
 automated 86
 consoles 7, 41
 terminals 86
 logout 7
 lost+found directory 182
 lp utility 119
 lpc utility 120
 LPDEST environment variable 135 – 136
 lpq utility 120
 lprm utility 120
 LPSVR environment variable 135, 138
 see also spooling
 lpsvr file
 see spooling, setup file
 lpsvr utility 119
 ls utility
 viewing permissions 66

M

mail
 see QUICS, email
 makefile for disk booting
 examples 143
 location 142
 macros for changing defaults 142
 see also images
 makefile for network booting
 location 144
 see also images
 mapping between logical and physical node IDs
 see network
 masking nodes 111
 see also network
 mathematical characters
 entering 194, 198
 melt utility 162
 memory allocation, local 141
 metacharacters (shell) 15 – 16, 19
 mixed networks
 see multiple network links
 mode bits
 see permissions
 modem utility 87
 modems
 configuring 87
 connecting 82 – 83
 dedicated line for incoming calls
 not required 87
 flow control between terminals and modems 81
 hardware handshaking, enabling & locking 82
 high-speed, error-correcting 82
 incoming call, detecting 81
 indicating a connection has been made 81
 locked baud rate, using 87
 standard 83
 terminal type, querying user for 89

mounting filesystems 92 – 94
moving
 cursor 4, 199
 files 29
multi-port serial adapters 78
multiple commands, entering 20
multiple network links
 additional drivers, starting 109
 benefits 106
 card used for booting over net 109
 cards, adding 108
 communicating across networks 106
 configuring 108 – 116
 drivers, starting 109
 essential drivers, location and purpose 109
 Ethernet nodes, adding to Arcnet
 network 112 – 113
 fault-tolerant Ethernet network 115
 logical network IDs, specifying 110
 netboot configuration file 111
 netmap file 111, 113, 115
 private network link 116
 see also network

N

name server
 selecting a machine as 40
nameloc utility 47, 56
netboot
 see network
network
 additional drivers 109
 Arcnet cards 51
 Arcnet network, adding Ethernet
 nodes 112 – 113
 boot server, associating nodes with 105
 configuring 54 – 57
 selecting 52 – 53

network (cont'd)
 boot server, setting up multiple
 servers 104 – 105
 booting 44, 104, 109, 111, 141, 144
 from Ethernet/Token Ring cards 104
 build file, selecting for network media 57, 112
 card used for booting over net 109
 card, installing 54
 communicating across networks 106
 diagnosing problems 59
 drivers 51
 starting 55, 109
 starting multiple drivers 110
 errors, reporting 58
 essential drivers 109
 Ethernet cards 51
 booting from 104
 executing commands on remote nodes 17
 fault-tolerance, establishing 53, 104, 106
 fault-tolerant Ethernet network 115
 filesystem, configuring for network 93
 hardware 51
 images for network booting 144
 large networks, configuring 53, 104
 licensing 45 – 47, 50, 54, 56
 logical network IDs 50
 for multiple net links 110
 logical node ID, changing (effect of) 38
 logical node IDs, assigning 50, 52, 56
 manager, starting 55
 mapping between logical and physical
 node IDs 52, 57, 111, 113, 115
 multiple links (*see also* multiple
 network links) 106 – 116
 nameloc utility 56
 netboot configuration file 57, 105, 109, 111
 netboot utility 56, 105, 140, 144
 netmap file 52, 57, 111, 113, 115
 node IDs, changing (effect of) 38
 nodes, customizing 37

network (cont'd)
 nodes, masking, unmasking, & deleting 111
 parallel networks (see also multiple network links) 106 – 107
 pathnames 25
 physical node IDs 50
 determining 51
 displaying 57
 planning 53
 private network link, setting up 116
 problems, diagnosing 59
 setup considerations 51 – 53
 Token Ring cards 51
 booting from 104
 troubleshooting 59
 workstation, configuring 58 – 59
see also licensing; multiple network links;
 physical node IDs; workstation
 Network Manager
 drivers, starting 55
 starting 55
 nodes
 see network
 nohup command 18

O

online usage messages 18
 onnode command 17
 ontty command 17
 operating system images
 see images
 outgoing & incoming calls on same serial line 87
 output
 appending to a file 16
 piping to another command 16
 redirecting 16
 suspending and resuming 8
 output buffers, specifying 77

output devices
 reinitializing between spool jobs 121
 overwriting
 characters 4
 ownership of files
 changing 67
 DOS 101
 see also permissions

P

parallel devices 76
 multiple parallel ports 76
 output buffers, specifying 77
 single parallel port 76
 parallel driver
 starting 40, 76
 parallel networks
 see multiple network links
 see also network
 parent directory 170
 see also directories
 parity bits 81
 partitioning the pathname space
 see pathname space, partitioning
 partitions
 bitmap blocks 168
 directory structure, checking 173
 DOS 97 – 98
 key components 167
 loader block 167
 creating 173
 multiple partitions on single drive 93
 partition block 173
 root block 167
 root directory 168
 see also pathname space, partitioning

- password
 - changing 62
 - encrypted passwords 70
 - entering 62
 - passwd** utility 62
 - stringency 62
 - user ID ranges 62
 - see also* accounts; security
- password database
 - access to password files 69
 - default password files 70
 - /etc/group** file 69
 - /etc/passwd** file 62, 69
 - /etc/shadow** file 70
 - see also* security
- patching minor disk problems 174
- pathname space, partitioning 92 – 94
 - hard disk and floppy 92
 - local and remote hard disks 93
 - two hard disks 93
 - two or more QNX partitions 93
 - using symbolic links instead of prefixes 95
 - see also* *System Architecture*
- pathnames 25 – 26
 - network pathnames 25
- pattern-matching characters 19
- pax** utility 41, 159, 163
- permissions 65
 - bypassed by superuser 65
 - directories 66
 - DOS/QNX permission mapping 100
 - executable file running privileges of its
 - owner & group 68
 - execute permission 66
 - group ID, changing 67
 - mode bits 65
 - changing 67
 - how they're assigned 67
 - bits stored in **st_mode** field 65
 - ownership, changing 67
- permissions (cont'd)
 - read permission 66
 - regular files 66
 - setgid 68
 - changing 67
 - setuid 68
 - changing 67
 - superuser and 65, 67 – 68
 - umask permission bit mask 67
 - user ID, changing 67
 - viewing 66
 - write permission 66
 - see also* security
- phoning Quantum 188
- physical filesystem, defined 92
- physical node IDs
 - assigning 50
 - determining 51
 - displaying online 57
 - formats 51
 - location 50 – 51
 - specifying in decimal 52
 - unique within each network 50
 - see also* network
- pipng output to another command 16
- portability
 - filenames 24
 - portable character set 24
- prefix tree 92
 - see also* *System Architecture*
- print jobs
 - canceled 120
 - querying 120
 - selected on queue priority & age of job 122
 - submitting 119, 136
 - see also* spooling
- print spooling
 - see* spooling
- PRINTER** environment variable 136
 - see also* spooling

- printers
 - parallel printers, improving turnaround time 77
 - reinitializing between spool jobs 121
 - serial printers, connecting 84
 - serial printers, flow control 84
 - see also* parallel devices; serial devices; spooling
 - priority of processes
 - setting in buildfile 141
 - private network link
 - setting up 116
 - see also* network
 - privileges
 - giving to another user 62
 - non-privileged users performing privileged functions 68
 - see also* permissions; security
 - processes
 - displaying processes currently running 29
 - group ID, changing 63
 - heap size, setting in buildfile 141
 - killing 8
 - priority, setting in buildfile 141
 - remote execution 17, 108
 - selecting for an OS image 141
 - programs, revising (RCS) 146
 - see also* RCS
 - pseudo tty driver
 - starting 76
 - public-domain software 186
- Q**
- qnx** terminal 191
 - attributes, saving 201
 - bell, ringing 201
 - character attributes, setting 199
 - color, setting 201
 - composed character input sequences 198
 - qnx** terminal (cont'd)
 - cursor, defining and moving 199
 - escapc sequences 199
 - EXTENDED mode 197
 - keyboard codes 196
 - lines, deleting and inserting 200
 - SCAN mode 197
 - standard console character set 194
 - see also* consoles
 - qtalk** terminal emulator 87
 - Quantum update system
 - see* QUICS
 - Quantum, phoning 188
 - see also* QUICS
 - queues (spooling)
 - accessing 135 – 138
 - chaining 124, 131
 - chaining despoiled job onto 126
 - controlling 120
 - copy-in filters 121
 - default 128
 - specifying 126
 - vs copy-out filters 133
 - copy-out filters 121
 - default 128
 - specifying 126
 - default queue 135 – 137
 - defined in **1p0xvz** file 124
 - describing 126
 - located in **/dev/spool** directory 118
 - maximum no. of jobs 126
 - minimum no. of jobs before flushing 126
 - multiple queues feeding multiple targets 123
 - feeding single target 122, 132
 - one queue feeding multiple targets 123
 - feeding single target 122
 - output can be connected to multiple targets 121
 - priority, specifying 126
 - purpose of 121
 - registering name of spooler 126 – 127

queues (cont'd)

- specifying 119, 129
- writing directly to queue file 120, 131
- see also* spooling

QUICS

- conferencing system 187
- email 186
- free software 186
- hardware 187
- logging in 189
- phoning Quantum 188
- refresh policy 186
- reporting bugs 186
- using 189

- quoting special characters 14

R

Rambo mode (**chkfsys**) 175

raw disks

- browsing 174
- see also* disks

RCS

- branch 150, 154 – 155
- branchpoint 154
- checking in 147
- checking out 148
- ci** (check-in) command 147 – 150
- co** (check-out) command 147 – 148, 150
- commands (summary) 155
- defined 146
- delta nodes 154
- file format 152 – 154
- getting started 147
- head 155
- ident** command 151
- identification strings 150 – 151
- identifying revisions 147
- locking files 148, 150

RCS (cont'd)

- log messages 151
- rca** command 148
- RCS directory, creating 149
- RCS files, storing 149
- rcsdiff** command 148
- revision numbers, assigning 150
- revisions, identifying 147
- trunk 150, 154 – 155
- tutorial 147
- working file 147 – 149, 155
 - protecting 149

- read permission 66

reading

- input from a file 16

- readme files, location 27

realtime clock

- getting time from 39, 43
- see also* time zones

rebooting 9

- see also* booting

recalling commands 5

- received parity bits, suppressing checking of 81

recovering

- deleted files 176
- disks 165 – 183
- files 165 – 183
- lost files and directories 182 – 183
- lost+found** directory 182
- see also* backups; file maintenance utilities

redirecting

- input 16
- output 16

refresh policy 186

- remote execution of commands 17, 108

remote files

- accessing 25

removable disks

- backing up to or restoring from 162
- see also* disks

- removing
 - directories 29
 - files 29
 - renaming files 29
 - resolving access conflicts (RCS) 146
 - resources
 - access, controlling 62 – 70
 - access, restricting via a spooler 118
 - see also* access control utilities; permissions
 - restoring data
 - compressed data, problems recovering 162
 - floppies 160, 163
 - on another UNIX system 159, 163
 - removable disks 162
 - tape 161
 - taz/cpio* format 159
 - see also* archives; backups
 - restoring your filesystem
 - see* recovering
 - resuming output 8
 - revision control system
 - see* RCS
 - right arrow key 4
 - ringing a bell on a terminal 201
 - root
 - see* superuser
 - root block 167
 - creating 173
 - restoring 173
 - root directory 25
 - .bitmap* file 169
 - contents 168
 - creating 173
 - .lnodes* file 169
 - see also* directories
 - RS-232
 - cabling assignments 79
 - flow control between terminal and modem 81
 - serial protocol 80
 - session control 80
 - RS-232 (cont'd)
 - signal names 79
 - transmitted characters 80
 - see also* serial devices
 - Rubout key 4
- ## S
- SCAN mode 197
 - see also* *gnx* terminal
 - SCCS
 - see* RCS
 - screen fonts, changing 7
 - scripts (shell) 20
 - second filesystem, adding
 - see* DOS
 - see also* pathname space, partitioning
 - security 62 – 70
 - access control utilities 62
 - access to files 65
 - file permissions 65
 - group IDs 64
 - login* utility 62
 - newgrp* utility 63
 - passwd* utility 62
 - password database 69
 - password files, defaults 70
 - permissions 65
 - su* utility 62
 - user IDs 64
 - see also* password; permissions; resources
 - serial devices 77 – 89
 - connecting 82
 - hardware adapters 77 – 78
 - I/O addresses 77
 - hardware interrupt generated by each UART 77
 - modems, connecting 82 83
 - multi-port serial adapters 78
 - ports, configuring 81

- serial devices (cont'd)
 - printers, connecting 84
 - RS-232 serial protocol 79
 - single-port serial adapters 77
 - terminals, connecting 85
 - troubleshooting 89
 - typical installation 78
 - see also* modems; RS-232; terminals
- serial driver
 - starting 40, 76
- serial ports
 - baud rate, specifying 40, 82
 - configuring 81
 - data bits 81
 - interrupts, sharing 78
 - parity bits 81
 - stop bits 81
- server node, selecting & configuring
 - see* boot server
- services *see* images; system initialization file
- setgid 68
 - changing 67
- setting up user accounts 61 – 73
- setuid 68
 - changing 67
- shell
 - command basics 11
 - command lists 20
 - command syntax 12
 - command usage, displaying 18
 - filename expansion 19
 - metacharacters 15 – 16, 19
 - quoting special characters 14
 - recalling commands 5
 - redirecting input and output 16
 - remote execution 17
 - scripts 20
 - special characters 14
 - wildcards 19
 - see also* commands
- sinit utility 36
- software flow control
 - enabling & locking 85
- software refresh policy 186
- software, free 186
- sorting files 29
- source code control system
 - see* RCS
- spatch utility 174, 177, 182
- special characters
 - quoting 14
- spooler
 - accessing 135 – 138
 - default spooler 135 – 136
 - defined 118
 - registering name 126 – 127
- spooling
 - accounting information, recording 126, 131
 - architecture of spooler 121 – 123
 - attribute keys
 - adding custom keys 128
 - referencing as metavariables 128
 - attributes of queues & targets, specifying 125
 - canceling jobs 120
 - chaining queues 124, 131
 - control programs
 - effect on copy-out 128
 - purpose of 121
 - specifying 126
 - copy-in filters 121, 130
 - bypassing 124
 - default 128
 - specifying 126
 - vs copy-out filters 133
 - copy-out filters 121, 130
 - default 128
 - possible effects 130
 - specifying 126
 - default queue 135 – 137
 - default spooler 135 – 136

spooling (cont'd)

- despooled job, chaining to queue 126
- despooling, specifying wait before 126
- error message, mailing to user 131, 134
- `/etc/config/lpsrvr` file 119, 124
- filters 121, 128 – 129
- global keywords 127
- input errors 131
- jobs
 - querying 120
 - removing 120
 - selected on queue priority & age of job 122
 - submitting 119, 136
- keys
 - adding custom keys 128
 - global 127
 - referencing as metavariables 128
- `lp` utility 119
- `lpc` utility 120
- `LPDEST` environment variable 135 – 136
- `lpq` utility 120
- `lprm` utility 120
- `LPSRV` environment variable 135, 138
- `lpsrvr` utility 119
- mailing error message to user 131, 134
- metavariables 128
 - quoting 130
 - referencing keys as 128
- output devices
 - can't be hardcoded in copy-out 130
 - reinitializing between jobs 121
 - specifying 126
- output errors 131
- `PRINTER` environment variable 136
- querying jobs 120
- queues
 - accessing 135 – 138
 - chaining 124, 131
 - controlling 120

spooling (cont'd)

- queues
 - default queue 135 – 137
 - defined in `lpsrvr` file 124
 - describing 126
 - located in `/dev/spool` directory 118
 - maximum no. of jobs 126
 - minimum no. of jobs before flushing 126
 - multiple queues feeding
 - multiple targets 123
 - multiple queues feeding
 - single target 122, 132
 - one queue feeding multiple targets 123
 - one queue feeding single target 122
 - output can be connected to multiple targets 121
 - priority specifying 126
 - purpose of 121
 - specifying 119, 125, 129
 - writing directly to 120, 131
- spooling (cont'd)
 - registering name of spooler 126 – 127
 - removing jobs 120
 - server 119
 - starting 119
 - setup file 124 – 128
 - examples 132 – 134
 - included in distribution disks 119
 - node-specific files 119
 - syntax 124
 - starting spooler 119
 - submitting jobs 119, 136
 - targets
 - command executed when target abandons 126, 131
 - command executed when target completes 126
 - defined in `lpsrvr` file 124
 - describing 126
 - only means of accessing a device 130

- spooling (cont'd)
 - targets
 - output can be connected to
 - only one device 121
 - purpose of 121
 - selected by availability 123
 - specifying 125 – 126, 129
 - temporary directory for spool files 126 – 127
 - utilities 119
- standard input
 - redirecting 16
- standard output
 - appending to a file 16
 - pipng to another command 16
 - redirecting 16
- standard time zone, defining 204
 - see also* time zones
- stop bits 81
- string patterns, searching for 29
- su** utility 62
- submitting spool jobs 119
 - see also* spooling
- suffixes (filenames) 24
- superuser
 - /etc/shadow** file, reading 70
 - access control utilities, running with
 - superuser permissions 68
 - accessing files regardless of permissions 65
 - accounting file access 71
 - accounts, adding 62
 - Dosfaya** file ownership 101
 - mode bits of any file, changing 67
 - password file, editing 64
 - spool jobs, removing 120
- suspending output 8
- symbolic links
 - using to partition pathname space 95
- symbols, international and mathematical
 - see* characters
- syntax (commands) 12
- sysinit** file
 - see* system initialization file
- system debugger
 - invoking & disabling 8
- system initialization file 36 – 41
 - altaysinit** 38
 - base-level services 39
 - clock
 - getting time from 39
 - starting 42
 - common commands, placing in separate file 39
 - console driver, starting 39
 - consoles, starting login 41
 - cron server, selecting machine as 42
 - Device Manager, starting 39
 - DOS filesystem, starting 42
 - environment variables, defining 41
 - first command in file 39
 - floating-point emulator 40
 - floppy driver, starting 41
 - hard disk booting 39
 - keyboards, non-USA 42
 - last command in file 41
 - login**, starting on consoles 41
 - name server, selecting machine as 40
 - Network Manager and drivers, starting 55
 - node mappings, loading 40
 - optional services 41
 - parallel driver, starting 40
 - realtime clock, getting time from 39
 - serial driver, starting 40
 - sysinit** 38
 - sysinit.1**, created by **install** 37
 - sysinit.node** 37 – 38
 - copying to **altaysinit** before
 - changing 38
 - time zone, defining 39
 - workstations, customizing 37
- system won't boot 177 – 181

T

- tape
 - backing up to or restoring from 161
 - tape** utility 161, 163
 - see also* archives; backups
- tar** utility 159, 163
- targets (spooling)
 - command executed when target completes 126
 - control programs 121
 - specifying 126
 - defined in **lp~~s~~rvx** file 124
 - describing 126
 - only means of accessing a device 130
 - output can be connected to only one device 121
 - output device, specifying 126
 - purpose of 121
 - selected by availability 123
 - specifying 125 – 126, 129
 - see also* spooling
- technotes, location 28
- temporary files
 - default location 28
- termdef** utility 89
- terminal initialization utility (**tinit**) 7, 41, 86
- terminals
 - capabilities 88
 - communication, ending 80
 - configuring for users 86
 - connecting 85
 - custom apps, launching 86
 - daemon, starting 41
 - defining type 88
 - DTR line, dropping 80
 - escape sequences (**qnx** terminal) 199
 - flow control between modems and terminals 81
 - flow control, enabling & locking 85
 - full-screen programs, **terminfo** database 88
 - terminals (cont'd)
 - hardwired terminals 88
 - keyboard limitations of attached terminals 5
 - launching custom apps 86
 - login** 86
 - login**, automated 86
 - qnx** terminal 191
 - qtalk** terminal emulator 87
 - querying for type 89
 - supported terminals 88
 - terminfo** database 88
 - tinit** utility 7, 41, 86
 - troubleshooting 89
 - type, defining 88
 - types supported 88
 - see also* consoles; **qnx** terminal; serial devices
 - terminating a process 8
 - terminfo** database 88
 - terminfo** files, accessing and creating 88
 - text file formats
 - DOS versus QNX 98
 - text files
 - revising (RCS) 146
 - see also* files
 - textto** utility 98
 - third-party applications
 - installing 35
 - time zones 42, 203 – 205
 - coordinated universal time (UTC) 42
 - date and time, establishing 43
 - daylight savings time, changing to or from 203, 205
 - daylight savings zone, defining 204
 - east of Prime Meridian 204
 - establishing 39, 42 – 43
 - Greenwich Mean Time 42
 - not setting, effects of 43
 - standard zone, defining 204
 - time, establishing 43

time zones (cont'd)
 transferring files to a different time zone 43
 TZ environment variable 39, 43, 203 – 204
 UTC 42
 west of Prime Meridian 204
 tinit utility 7, 41, 86 – 87
 Token Ring drivers/cards
 booting from 104
 physical node ID, determining 51
 see also network
 tone, generating on console 201
 transmission of data, slowing down 81
 troubleshooting
 network 59
 serial devices 89
 typing, correcting errors 4
 TZ environment variable 39, 43, 203
 contents of 204
 see also time zones

U

UARTs 77
 hardware interrupts generated by 77
 umask permission bit mask 67
 see also permissions
 unmasking nodes 111
 see also network
 up arrow key 5
 update system
 see QUICS
 usage messages 18
 use command 18
 user accounts
 see accounts
 user IDs 64
 changing 67
 effective user IDs 64
 effective user IDs, changing 64

user IDs (cont'd)
 ranges, determining 62
 see also security
 username
 entering at login 62
 UTC
 see time zones
 utilities
 common file management utilities 29
 logging info about users 70
 usage messages, displaying 18
 see also commands

V

variables, defining 41
 version control
 see RCS
 virtual consoles
 switching 6
 see also consoles
 vol utility 41, 159, 163
 volume labels (DOS) 100

W

warm boot 9
 wildcards 19
 working directory
 see current working directory
 working files (RCS)
 see RCS
 workstation
 booting over the network 44, 58, 109, 111, 144
 communicating with server via essential
 driver 109
 configuring 58 – 59
 customizing via `sysinit.node` file 37

workstation (cont'd)

defined 53

image loaded when booting from network 53

non-USA keyboard, specifying 42

see also network

write permission 66

Z**zap utility** 174 – 176

