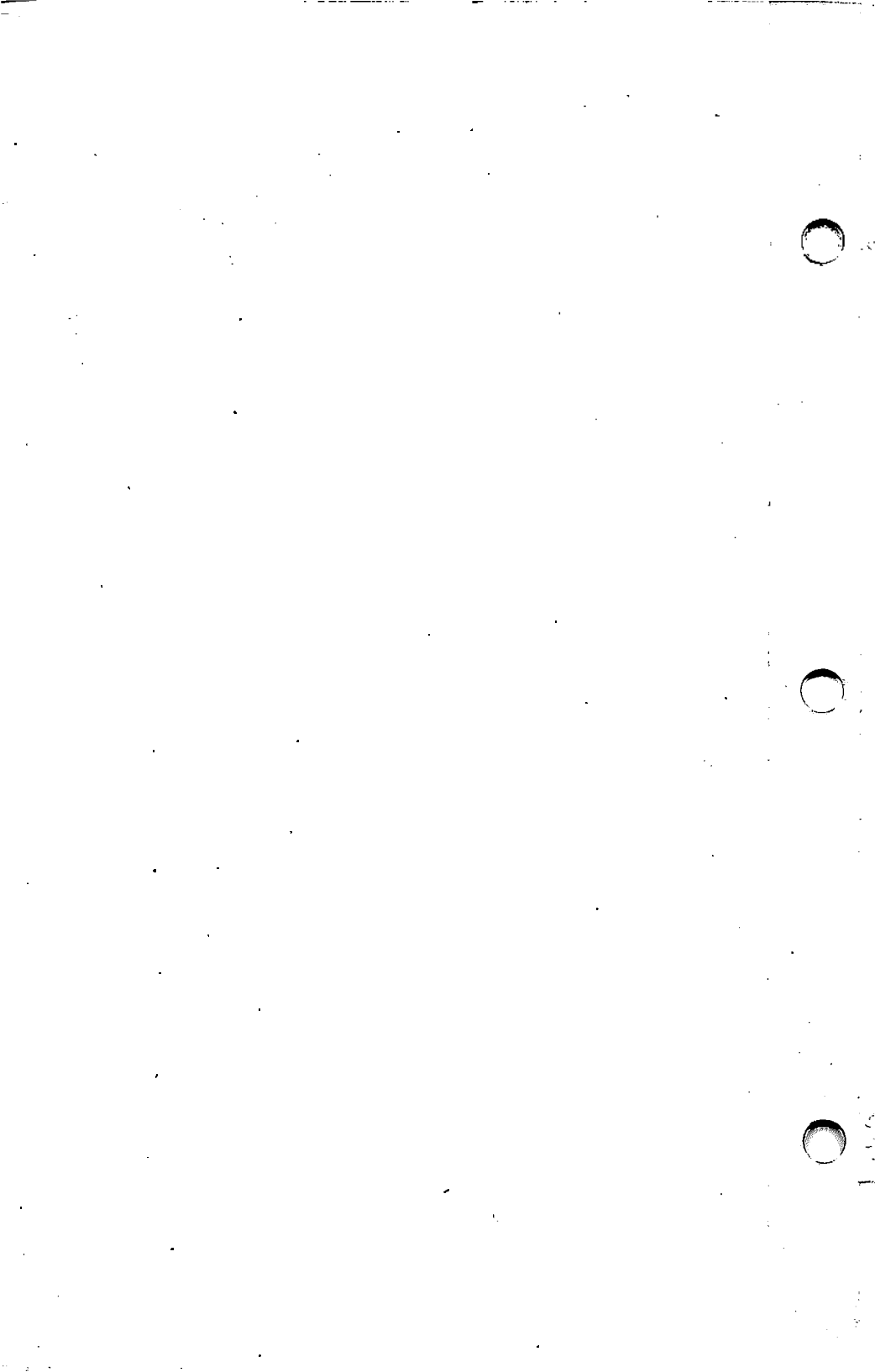


The logo features the letters 'QNX' in a bold, stylized font. The 'Q' and 'N' are white with blue outlines, while the 'X' is solid blue. To the left of the letters are several horizontal blue lines of varying thickness, suggesting motion or data flow.

QNX

**OPERATING
SYSTEM**



QNX™

Reference Guide

Version 2.1

Copyright © 1982,1988
Quantum Software Systems, Ltd.
ALL RIGHTS RESERVED.

Published by

Quantum Software Systems Ltd.
Kanata South Business Park
175 Terrence Mathews Crescent
Kanata, Ontario K2M 1W8
Canada
(613) 591-0931

© 1982,88 Quantum Software Systems Ltd.

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 Quantum Software Systems Ltd. Printed in Canada.

First Printing: January, 1988
Second Printing: March, 1988

QNX is a registered trademark of Quantum Software Systems Ltd.

QNX Manual

Page

1. Introduction	1
1.1 What You Were Shipped	1
1.2 Booting QNX	3
2. Installation Using The Install Command	5
3. Manual Installation	7
3.1 Background on Disks and disk drivers	7
3.1.1 XT Hard Disk Driver	9
3.1.2 AT Hard Disk Driver	10
3.1.3 PS2 Hard Disk Drivers (model 50 and up)	10
3.1.4 HP NIGHTHAWK Hard Disk Driver	11
3.1.5 BIOS Hard Disk Driver	11
3.2 Step-by-step Installation of Hard Disk	12
3.3 Creating A System Initialization File	20
3.4 Summary	22
3.5 Creating A Login Password File	23
3.6 Mounting a DOS or Second QNX Partition	26
3.7 More than One Physical Hard Disk	27
3.8 Changing Operating System Defaults	27
3.9 Removing the QNX loader	28
4. Network Installation	29
4.1 Hardware Installation	29
4.2 Software Installation	29
4.3 Summary	32
4.4 Starting the Poller	32
4.5 Recovery Without the Poller	33
4.6 Troubleshooting QNET Network Installation	34
4.6.1 Diagnostic Tips	35
4.6.2 Symptom Checklist	35
5. Terminals, Modems and Printers	37
5.1 Serial	37
5.1.1 I/O ports and Interrupts	37
5.1.2 Baud Rates and Parity	38
5.1.3 Connecting Cables (DTE vs DCE)	39
5.1.4 Setting QNX Line Editing Options	41
5.1.5 Flow Control	41
5.1.6 Logging In From A Terminal	42

5.1.7	Logging In From A Modem	42
5.2	Outgoing Calls	43
5.2.1	Problems	43
5.3	Parallel	43
5.4	Smartcards	44
6.	Devices	45
6.1	Terminal Input	45
6.1.1	Alt, Shift and Ctrl	45
6.1.2	Caps Lock, Num Lock	45
6.1.3	Type-Ahead	46
6.1.4	Line Editing	46
6.1.5	Compose Characters	48
6.1.6	Input Gate	49
6.1.7	Recalling Command Lines.	49
6.2	Terminal Output	50
6.2.1	Attributes	50
6.2.2	Output Escape Sequences	50
7.	Full Screen Consoles	55
7.1	Mounting Consoles	55
7.2	Switching Consoles	57
7.3	Initiating Commands in Other Consoles	58
7.4	Consoles and Graphics	58
8.	Files	61
8.1	File Names	61
8.2	Structure of Files	61
8.3	Directories	62
8.4	Pathnames	65
8.4.1	Specifying a Drive as Part of Your Pathname	66
8.5	Your Current Directory	67
8.5.1	Changing Your Current Directory	67
8.6	Moving Up the File Structure	68
8.7	File Attributes and Permissions	69
8.8	User Numbers	73
8.9	Device Names	73
8.10	Network Access to Files and Devices	75
8.10.1	Remote Search Order	75
8.10.2	Automatic Remote Searching	76
8.10.3	Remote Current Directory	77
8.10.4	Mounting A Remote Disk	77
8.10.5	Network Devices	78

8.10.6	Limiting Network Access	78
--------	-------------------------	----

9. The Command Interpreter (Shell) 79

9.1	Shell Prompt	79
9.2	Command Input/Output Redirection	80
9.3	Quoting	81
9.4	Filename Generation	82
9.5	Querying	83
9.6	Background Tasks	83
9.7	Multiple Commands On a Line	84
9.8	Pipes	84
9.9	Comment Lines	85
9.10	Executing Commands on Another Node	85
9.11	Command Files	86
9.11.1	Built-In Shell Variables	86
9.11.2	User Shell Variables	87
9.11.3	Executing Shell Commands	88
9.12	Local Shell Commands	89
9.13	Quick Reference to the Shell	90

10. Tasks 93

10.1	Introduction	93
10.2	System Tasks	94
10.3	Inter-task Communication	95
10.3.1	Messages	96
10.3.1a	Death of a Task	99
10.3.1b	Messages Across the Network	99
10.3.1c	Virtual Circuits	100
10.3.2	Ports	100
10.3.2a	Identification - single node only	101
10.3.2b	Semaphores	102
10.3.2c	Signals	102
10.3.3	Exceptions	103
10.4	Global Names	106
10.5	Task States	106
10.6	Task Ids	108
10.7	Task Hierarchy	108
10.8	Task Creation	109
10.9	Terminal Ownership	109

11. QUICS - The Quantum Update System 111

11.1	How To Phone Us	112
11.2	X25 Access	113

12. Tips on Using QNX	115
12.1 Memory Requirements	115
12.2 Enabling Colour	115
12.3 Disabling Colour	115
12.4 The Mount Command	115
12.5 Ramdisk	117
12.6 Shared Libraries	118
12.7 Operating On Groups Of Files	118
Character Set and Keyboard Codes	119

1. Introduction

BE SMART AND READ THIS

This chapter is written for the first time QNX user who wishes to install QNX on an IBM PC, AT, HP Vectra, PS/2 or compatible. The documentation will take you step by step through the installation procedure. If you run into problems not covered by the manual you may call the QNX technical support line.

Technical Support

(613) 591-0941 (9:30 am to 5:30 pm EST)

The installation involves the execution of a number of QNX utility commands. Over time you will become familiar with these commands but right now we will only cover the basics needed to get started. Although you can execute the commands directly, we have provided an install program which will invoke the necessary commands from a user friendly menu.

It is your choice whether to install the system manually or through the install command. Both are documented below (With Install - chapter 2, Manually - chapter 3), however the manual method contains much more technical detail. This detail is usually appreciated by the technical user and may be overpowering for the novice. We recommend that you use the install program and read through the manual installation which parallels the execution of the program. In any case, the first step is to read the rest of this chapter.

If you are upgrading DO NOT use the INSTALL command. It will destroy your existing QNX files. Use the documentation provided with your upgrade.

1.1 What You Were Shipped

QNX is provided on several floppy diskettes. Depending on the configuration you have ordered, you will have received some (or all) of the following diskettes:

	360K Diskettes	
<u>Disk</u>	<u>Directories</u>	<u>Contents</u>
1 QNX 2.1x Boot		
	/cmds	Essential commands
	/netboot	Operating system images

2	QNX 2.1x Boot Utilities	
	/cmds	Other commands
	/config	Configuration files
	/drivers	Disk and graphics drivers
	/expl	Explain files
3	Utilities 1	
	/cmds	Utility commands
4	Utilities 2	
	/cmds	Utility commands
5	"C" Compiler 1	
	/cmds	Compiler, Assembler and Linker ...
	/samples	Some C programs
6	"C" Compiler 2	
	/lib	System library
	/mathlib	Floating point library
	/mathlib8087	8087 floating point library

720K/1.2Meg Diskettes

Disk	<u>Directories</u>	<u>Contents</u>
-------------	---------------------------	------------------------

1	QNX 2.1x Boot	
	/cmds	Commands
	/netboot	Operating system images
	/config	Configuration files
	/drivers	Disk and graphics drivers
	/expl	Explain files
	/user/qnx	User directory
2	Utilities	
	/cmds	Utility commands
3	"C" Compiler	
	/cmds	Compiler, Assembler and Linker ...
	/lib	System library
	/mathlib	Floating point library
	/mathlib8087	8087 floating point library
	/samples	Some C programs

We always ship two versions of QNX. One which runs in real mode on PC's, AT's and PS/2's and one which runs in protected mode on AT's and PS/2's. There is also a special version for old style HP Vectras (earlier than Oct 87). You must request the special Vectra version. The standard processor *types* are:

PCAT - PC, AT, VECTRA, PS/2 or compatible running in real mode.
ATP - AT, VECTRA, PS/2 or compatible running in protected mode.

OR

HV - Early HP Vectra running in real mode.
HVP - Early HP Vectra running in protected mode.

The PCAT and HV boot run in real mode (like DOS) and limits you to 640K of memory. The ATP and HVP boot run in protected mode and allow up to 15 Megabytes of memory.

The same QNX utilities and applications will work on ALL of these versions of QNX. Also, any application developed by yourself or others should work equally well on each QNX version. The operating system is responsible for providing this *machine independence*.

1.2 Booting QNX

To boot QNX, place the disk labeled "QNX 2.1x Boot" in drive 1 and turn on the main power switch of your Personal Computer. If power was already switched on, turn it off for at least 10 seconds then turn it back on.

On a PC, QNX will prompt you for today's date. You should enter it in the form indicated by the prompt. For example, you could set the date and time to 12 October 1987 at 2:34 in the afternoon by typing:

12 oct 87 2 34 pm

On an AT or PS/2 the date will be read from the built-in clock/calendar.

QNX will now prompt you to login with the following message:

QNX Version 2.1x Release x node n
Copyright © Quantum Software Systems Ltd. 1983,1988
Login :

You should enter `qnx` followed by a carriage return at this point

```
Login: qnx  
$
```

which will log you in as a super-user. The dollar sign (\$) prompt indicates that QNX is waiting for a command. You will now have to configure QNX for your machine. If you have received a networked version of QNX, you should first configure a single machine. After you have gained experience with it, you may then configure the other machines in the network. These other machines will boot over the network. This is explained in more detail later on.

You should now go to chapter 2 (install program) or chapter 3 (manual installation).

2. Installation Using The Install Command

Do not read on until you have read chapter 1.

If you are upgrading DO NOT use the INSTALL command. It will destroy your existing QNX files. Use the documentation provided with your upgrade.

This install program should cover 95% of all installations. The program does not perform the installation itself but invokes a set of standard QNX utility commands to perform each step. These steps are described in more detail in chapter 3 which parallels the execution of the install program. You do not need to read it unless you want more detail on what is going on. This way you will learn a bit about QNX during the installation. Each utility is described in detail in the **Utilities** area of the manual (Utilities Tab).

At this point you should have booted QNX and have logged in as described in section 1.2. If you have booted from a 360K boot diskette, replace it with the boot utilities diskette. Then at the dollar sign prompt type the command

\$ install

and simply answer the questions presented to you. If you wish to install QNX on more than one hard disk partition or on more than one hard disk in the same machine you will have to read chapter 3. In this case you may still use the install program to create your first QNX partition on your first hard disk.

If you wish to share you disk with DOS we recommend that you install DOS first leaving room on the hard disk for QNX.



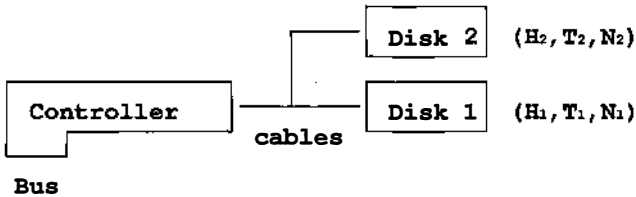
3. Manual Installation

Do not read on until you have read chapter 1.

This section describes in detail how to install QNX. If you consider yourself to be a novice you should probably refer to chapter 2 for a menu-driven installation. Even experienced users will like INSTALL.

The information in this section is useful and worth reading even if you do opt for the menu driven install program. Each utility command involved is described in greater detail in the Utilities area of the manual (Utilities Tab).

3.1 Background on Disks and disk drivers



Disk drives are available in various sizes. QNX needs to know the following information about a *drive*:

Number of heads	H
Number of tracks on each platter	T
Number of sectors on each track	N

The total size of the disk will be:

Total number of blocks is	$H * T * N$
Total number of bytes is	$H * T * N * 512$

Depending on the disk *controller*, one or more disk *drives* may be connected to it.

For QNX to communicate with a hard disk, an interface module called a **driver** must be properly mounted into the operating system. A driver is specific to a particular disk *controller* and knows the following information about that controller:

- I/O registers
- DMA Channels used
- Interrupt used

The **MOUNT** utility command is used to mount a driver. Each driver assumes a default value for the number of tracks, heads and sectors on your disk. For example, the XT driver assumes a 10 Meg disk while the AT driver uses the disk type defined by your AT setup program to determine the size of the disk. If you have a non-standard disk, the values for **h**, **t** and **n** can be overridden when you mount the driver.

The number of heads and tracks should be provided by the vendor when you receive your hard disk. Most drives contain 17 sectors per track. If you have an RLL drive it may contain 25 sectors per track. If you have an ESDI drive it may contain 34 or 35 sectors per track, although some ESDI controllers, for compatibility, pretend they have 17 sectors/track but twice the number of heads.

A **driver** treats each drive as a consecutive series of physical 512 byte blocks, starting at **one** and continuing to some number which is determined by the capacity of the drive ($H * T * N$). The drivers will normally map blocks through sectors, heads and then tracks with increasing block numbers. This minimizes head movement.

Here is a partial syntax of the **MOUNT** command regarding hard disks. Further information on this command can be found in the **Utilities** manual.

```
mount disk drive /drivers/driver_name pa=os_type  
          h=heads t=tracks n=sectors/track
```

NOTE: IF YOU MUST SPECIFY H, T or N YOU MUST SPECIFY ALL THREE PARAMETERS

In this command you have indicated 4 separate pieces of information.

drive - The number of the drive from QNX's point of view. QNX allows drive numbers from 1 to 8. Note that unlike DOS which uses letters for drives, QNX uses numbers. Although you may choose your own drive numbers, we recommend you adopt the following scheme:

<u>Disk</u>	<u>Use</u>
1	Floppy 1
2	Floppy 2 if present

continued...

- 3 Hard disk QNX partition #1
- 4 Hard disk partition #2
- 5 Ramdisk
- 6 ---
- 7 ---
- 8 ---

- driver_name*** - This identifies the software driver for a particular controller. (xt, at, ...).
- h, t and n*** - The physical structure of the hard disk. If these parameters are omitted, the driver will assume a default which may or may not be correct. Please refer to the documentation on each driver.
- os_type*** - The name of the partition to use. This may be the name **qnx**, **qny**, **qnz**, **dos**, or a number.

<u>Type</u>	<u>Description</u>
qnx	QNX partition (same as number 7)
qny	QNX partition (same as number 8)
qnz	QNX partition (same as number 9)
dos	DOS 2.1 partition (same as number 1)
1 ... 255	An explicit operating system <i>os_type</i>

When MOUNTING it is not necessary to specify the three key parameters of **H**, **T** and **N** unless the driver is unable to determine them. The capabilities of each driver we supply are described below.

3.1.1 XT Hard Disk Driver

A large number of drives now on the market boast an XT compatible hard disk controller. This controller **must** be compatible at the physical hardware level. Compatibility at the BIOS (ROM) level is not enough to use this driver. This driver assumes the old 10 Meg disk shipped by IBM on their old XT. If you have a 20 meg disk or one which is not identical to the default you must specify the proper number of tracks, heads and sectors.

	<u>Heads</u>	<u>Tracks</u>	<u>Sectors</u>	<u>SIZE</u>	
XT	4	306	17	10M	<--Default
mount disk 3 /drivers/disk.xt					
Other	2	612	17	10M	
mount disk 3 /drivers/disk.xt			t=612	h=2	n=17
Other	4	612	17	20M	
mount disk 3 /drivers/disk.xt			t=612	h=4	n=17

3.1.2 AT Hard Disk Driver

The AT driver uses the disk parameter table setup by the BIOS to determine the size of the disk. This means that you should NOT have to specify H, T or N when mounting the AT driver **UNLESS** you are using a drive other than that shipped by the AT manufacturer **AND** which does not correspond to any of the manufacturers' disk types as defined by the AT SETUP program.

You would mount the AT driver as follows. The second example is for a drive which does not have a supported disk type in the AT setup program provided with your machine. If you are using an unsupported drive, you **must** specify the number of tracks, heads and sectors.

```
mount disk 3 /drivers/disk.at
or
mount disk 3 /drivers/disk.at t=tracks h=heads n=sectors
```

3.1.3 PS2 Hard Disk Drivers

The PS/2 driver(s) use the disk parameter table setup by the BIOS to determine the size of the disk. This means that you should NOT have to specify H, T or N when mounting the PS/2 driver.

The model 25 and 30 use the driver in the system BIOS.

```
Model 25, 30
mount disk 3 /drivers/disk.bios
```

There are two controllers available for the PS/2 models 50 and up. One interfaces to an ST506 type drive and one to an ESDI drive. The ESDI controller is not currently available for the model 50. These are high performance drivers which feature read-ahead and write-behind. You should read the technical note "Page

Caching Disk Drivers" in the technical notes section.

Model 50 and up

ST506 Drive

mount disk 3 /drivers/disk.ps2 *read technical note*

ESDI Drive (disks > 70 Megabyte)

mount disk 3 /drivers/disk.ps2esdi *read technical note*

3.1.4 HP Nighthawk Hard Disk Driver

When you purchase an HP vectra there is an optional hard disk and controller available for it. It is usually sold into process control markets and at this printing it had a part number of 45816A. Within HP, it is sometimes referred to as the "Nighthawk" hard disk controller.

You would mount the driver as follows.

mount disk 3 /drivers/disk.nighthawk

3.1.5 BIOS Hard Disk Driver

This driver uses the INT13 BIOS calls to perform hard disk I/O and will **only work in REAL MODE versions of QNX**. It will work on any controller card or machine which supports INT13 hard disk I/O. These cards usually contain a ROM which the BIOS finds and executes when power is turned on. This allows us to provide immediate support for the many new drives on the market. The BIOS driver reads the disk configuration table set up by the BIOS to determine the physical characteristics of the hard disk. This can be over-ridden (if required) by specifying tracks, heads and sectors on the mount command. This driver should only be used as a last resort.

This is a busy-wait driver and may cause a slight stutter effect in a multi-user environment. The BIOS driver **will not work in protected versions of QNX (ATP)** nor can it be used with the DOS emulator.

mount disk 3 /drivers/disk.bios

or

mount disk 3 /drivers/disk.bios t=tracks h=heads n=sectors

3.2 Step-by-step Installation of Hard Disk

At this point you should have booted QNX and have logged in as described in section 1.2. The procedure required to install QNX onto your hard disk is described here in a step-by-step fashion.

NOTE that any QNX command will print a short usage message when a '?' is specified as the only command line argument. If you wish to know more about the commands, please refer to the "Utilities" manual.

STEP 1 Creating a QNX hard disk partition

To create a hard disk partition on your disk, you must use the commands outlined below with the proper arguments.

Step 1a - Mounting a Hard Disk Driver

Quantum supplies a number of drivers for hard disks under the directory '/drivers' on your boot diskette. The following drivers are provided:

<u>File Name</u>	<u>Description</u>
/drivers/disk.xt	XT hard disk driver
/drivers/disk.at	AT hard disk driver
/drivers/disk.bios	BIOS hard disk driver
/drivers/disk.nighthawk	Hewlett-Packard nighthawk hard disk driver <i>A standard HP Vectra uses the AT driver</i>
/drivers/disk.ps2	PS2 ST506 hard disk driver
/drivers/disk.ps2esdi	PS2 ESDI hard disk driver

Once you have decided on the appropriate driver file, you must **mount** it into the system using the MOUNT command. Until the driver is mounted you will be unable to access your hard disk.

At this point you should mount the hard disk as one large volume. Once the driver is loaded you will be able to read and write to block 1 of the disk which contains the partition information. You will then use FDISK to set up a QNX partition and immediately remount the driver specifying that it constrain itself to block accesses within the created partition. Some examples follow:

XT with standard 10 Meg disk (default)
mount disk 3 /drivers/disk.xt

XT with 20 Meg Seagate ST255 disk
mount disk 3 /drivers/disk.xt h=4 n=17 t=612

AT with standard 20 Meg disk (disk type 2)
mount disk 3 /drivers/disk.at

AT with nonstandard 60 Meg disk (disk type unknown)
mount disk 3 /drivers/disk.at h=7 n=17 t=980

In the last example the user has installed a disk drive which does not match any of the defined disk types known by the AT SETUP command.

Step 1b - Partitioning Your Hard Disk

You will now have to partition all or part of your hard disk for the exclusive use of QNX. Partitioning allows different operating systems to share a single hard disk without conflicting with each other.

You should now run the FDISK command to set up a QNX partition on the newly mounted disk.

fdisk 3

Look at the bottom of your screen and verify that the size of your disk and the values for H=, T= and N= are reasonable. If they do not reflect what you believe to be the size of your disk, then either the disk type is incorrect (AT, PS/2), the default values are incorrect (XT) or the wrong values for **h**, **t** and **n** were provided to the MOUNT command.

There are 4 partition slots. Use the up/down cursor keys to select one that is not in use. Type the letter **c** to select the **change** option. You should now enter the OS type (7 for QNX) and the start and end cylinder to be used for the QNX partition. Enter a carriage return after entering each number. The QNX partition may be any size up to 1 terabyte which easily covers all disks on the market today.

If other partitions already exist, your QNX partition must NOT overlap them. If the DOS partition takes up the entire drive you will have to reboot DOS, backup all your DOS files and reduce the size of the DOS partition before creating a QNX partition. If there is room for a QNX partition, then after you have made your

changes you must type **s** to save them back to the disk. Then type **q** to quit. The following is a typically partitioned hard disk as seen in FDISK.

```

Ignore Next Prev Change Delete Mount Boot Unboot Save Quit

      OS      start      End      Number      Boot
      Cylinder Cylinder  Cylinders  Blocks
-->  1. dos ( 1)      1      350      350      23867
     2. qnx ( 7)    351      612      262      17816      *
     2. --- (---)  ---      ---      ---      ---
     4. --- (---)  ---      ---      ---      ---

Use up/down arrows to select partition.
Type the letter c to change/add a partition.
Type the letter s to save your changes.
Type the letter q to quit.

QNX is os type 7,8 or 9   DOS is os type 1 or 4   Unused is os type 0

First cylinder is 0   Last cylinder is 614

Disk is 21,411,840 bytes   T=615 H=4 N=17

```

Step 1c - Remount Disk to use the QNX Partition

Now that you have created a QNX partition you **MUST** re-issue the mount for drive 3, to restrict it to the new partition. This time we will not specify a driver, but use the **d=driver_num** option to tell mount to use the same driver as it is currently using for drive 3. For example:

mount disk 3 d=3 pa=qnx

mount disk 3 d=3 t=612 h=4 n=17 pa=qnx

You now have a fully functional disk driver mounted and can set up a QNX file structure.

This MOUNT with the `d=drive_num` and the previous MOUNT and FDISK commands need only be done this one time to set things up. From now on, each time you boot QNX you will have to mount the disk driver with the `pa=qnx` option and if necessary the `h=`, `t=` and `n=` options. For example:

```
mount disk 3 /drivers/disk.at pa=qnx
```

```
mount disk 3 /drivers/disk.xt t=612 h=4 n=17 pa=qnx
```

You will be shown a little later how to place this mount command in a system initialization file which is automatically executed each time you boot.

Step 1d - Format QNX partition (AT's only)

Unless you have mounted `/drivers/disk.at` you should skip this step.

Users installing QNX on an AT may gain a significant performance increase by reformatting the QNX partition with a different interleave than that used by DOS. This step is optional but highly recommended. Use the FDFORMAT utility with the `+hard` option to reformat the QNX partition.

```
fdformat 3 +hard - Format partition for drive 3 with  
the default interleave (s=6).
```

If you have a very fast machine (16 MHz 386) then you may wish to format with a smaller interleave (also called stagger).

```
fdformat 3 +hard s=5 - Format partition for drive 3 with  
an interleave of 5.
```

If you attempt to format a non-AT hard disk you will receive errors and the disk should not be affected.

STEP 2 Initializing the QNX File Structure

To initialize the file structure for your disk, you must use the commands outlined below with the proper arguments.

Step 2a - Create a QNX File System

Before you can use your QNX partition you must create an empty QNX file system on it. This is done with the DINIT command. The command takes the drive number of the disk and a flag to indicate that it is a hard disk. Type the following command:

```
dinit 3 +hard
```

The DINIT command is also used to initialize the file system on floppy diskettes and ramdisks. You will therefore use this command many times. The **+hard** option is necessary only when initializing a hard disk and protects you should you accidentally type in the wrong drive number when initializing a floppy.

Step 2b - Mark Tracks You know are Bad

If your hard disk was shipped with a sheet of paper containing a list of bad tracks then you may use the DMARK command to explicitly mark the blocks on these tracks as un-usable. If you are willing to trust the DCHECK command to find all bad blocks you may skip this step. (On occasion, where blocks are intermittently bad, DCHECK may miss them). The DMARK command takes a list of tracks and heads. Each pair is separated by a space and the track and head are separated by a comma. For example:

```
dmark 40,4 250,2 500,0 >3:/bad_blks - mark track 40 head 4
                                           250      2
                                           500      0
```

We have redirected its output to **3:/bad_blks** so that it can be used with the DCHECK utility.

Note that even if only a single block is bad, the entire track is sacrificed. This is because interleaving of sectors on the disk locates blocks in different places, beyond our control.

You now have enough commands on the hard disk to switch over to it. This is done with the SEARCH command which tells QNX which drives to search when looking for commands. Currently your search order is 1 (single floppy) or 1 2 (two floppies). Type the following command:

```
search 3
```

Step 3c - Copy Other Floppies to Hard Disk

You should now copy the files on your other floppy diskettes to the hard disk. Type the following command for each floppy diskette you received.

```
backup 1:/ 3:/ +all s=c      - utilities diskette
backup 1:/ 3:/ +all s=c      - c compiler diskette
...                          etc.
```

The backup command is the easiest way to move the files on any diskette you receive from Quantum to your hard disk. Pressing the up-arrow key recalls the last line and allows you to easily issue multiple BACKUP commands.

STEP 4 Making QNX Boot From Hard Disk

The following describe the steps you will need to take to get a QNX hard disk partition to boot from hard disk.

Step 4a - Set OS image file to boot

You will need to use the BOOT command to enable hard disk booting. Its syntax is of the form:

```
boot os_file_name [c=config_file] [d=disk_driver] [-pause] [+qnxloader]
```

os_file_name

If you have properly backed up the floppies, you now have at least two OS image files (*os_file_name*) under the `/netboot` directory on your disk. One is **os.2.10pcat** (REAL MODE), the other is **os.2.10atp** (PROTECTED MODE). If you have the special HP Vectra version, they will be **os.2.10hv** and **os.2.10hvp**. One of these needs to be specified as the first argument to the `BOOT` command. This mechanism allows you to easily change whether you want to boot real or protected mode. It is also handy if you download a beta version of an OS from QUICS, our online update system, and wish to try it out.

[*c=config_file*] and [-pause]

These are not described here. See the section called "**Changing Operating System Defaults**", as well as the "**Utilities**" manual for more details.

[*d=disk_driver*]

When QNX boots, it needs a hard disk driver in order to access the hard disk. Unlike DOS, it cannot use the BIOS driver which does not work in protected mode. You must specify which hard disk driver to bind into a hard disk boot. The **d=disk_driver** option creates a file called `/config/hdisk.cfg` which is loaded in from hard disk along with the operating system file *os_file_name*.

[+qnxloader]

To boot QNX you must create a hard disk boot loader. You have two choices. You can use the default loader used by DOS or you can use a special QNX loader. The former will always boot the active partition as set by the QNX or DOS `FDISK` program. The latter will default to the active partition (which can be DOS) but will pause for several seconds and allow you to override it with another partition number which you may specify by typing in a single digit from 1 to 4. This gives more flexibility. However, if you have non-standard hardware or are already using a special loader then the QNX loader may cause you problems. For 99.9% of all installations we recommend you use the QNX loader.

IF YOU DO NOT HAVE A BOOTABLE DOS PARTITION YOU SHOULD SPECIFY THE QNX LOADER

You need only specify the **d=disk_driver** and **+qnxloader** options the first time you configure for hard disk booting. Later, if you wish to boot a different operating system (OS), simply specify the *os_file_name* argument to the `BOOT` command and nothing else. Here are some examples:

- 1) Boot real mode on an XT, using the DOS loader.
boot /netboot/os.2.10pcat d=/drivers/disk.xt

- 2) Boot protected mode on an AT, using the QNX loader.
boot /netboot/os.2.10atp d=/drivers/disk.at +qnxloader
- 3) Specify a different OS to boot, leave the driver and loader intact.
boot /netboot/testos

NOTE: Regarding the `/netboot/os_file_name` and `/config/hdisk.cfg` files, you may copy a new file on top of them but NEVER remove them (FREL, RM, ...) and then create a new one even if they have the same name. The BOOT command saves away the absolute start blocks of each of these files which will change once the file is removed. The BOOT command will always reset the proper starting block.

Step 4b - Make partition bootable

Your last step is to make the QNX partition the active, bootable partition. Type in the FDISK 3 command again, use the arrow keys to point to the QNX partition and type the letter 'b' for boot, (this will put a '*' under the "boot" heading), the letter 's' for save and finally the letter 'q' for quit.

fdisk 3

Select QNX with up/down arrows

Type the letter b

Type the letter s

Type the letter q

- **Make bootable.**

- **Save.**

- **Quit.**

You should now be able to boot from hard disk. Open the door to your floppy and simultaneously press the following 4 keys:

CTRL ALT SHIFT DEL

3.3 Creating A System Initialization File

Each time QNX is booted it will execute the commands in a file called

`/config/sys.init.nnnn` - Specific to node *nnnn*, initial choice.

or

`/config/sys.init` - Default.

where *nnnn* is the node number of the node which is booting (if you do not have a networking card, QNX will look for "sys.init.0"). If that file does not exist then it will try `/config/sys.init` (no node postfix).

config

sys.init sys.init.0 sys.init.1 sys.init.3
default

The message which greets you each time you log in is currently in the **sys.init** file. As you become acquainted with QNX, you will probably wish to add commands to this file which will customize QNX to your needs. This file typically contains commands to mount special disk drivers, set terminal options and perhaps read the date from a clock/calendar card. We will quickly illustrate a typical system initialization file. These files can be created, and may be modified with the editor (ED). You may want to refer to the section called "Tips on Using QNX".

```
rtc at
mount bmcache d=3
mount cache d=3 s=32k
mount xcache s=4k
mount float
mount lib /drivers/glib.ega

mount console $con2
cp /expl/logo $con
clock a=f104 &
```

Set QNX's date. (AT's only)
Mount a bitmap cache
Mount a directory cache
Mount an extent header cache
Mount a floating point library
Mount graphics and 43 line console library
Mount a second virtual console
Print QNX logo
Put a clock in top right corner

3.4 Summary

The following chart summarizes the steps necessary to install QNX on an AT hard disk.

```
| Mount the driver as disk 3  
|  
| mount disk 3 /drivers/disk.at  
|  
| Create a QNX partition  
|  
| fdisk 3  
|  
| Re-mount the QNX partition  
|  
| mount disk 3 d=3 pa=qnx  
|  
| Format QNX partition (AT's only)  
|  
| fdformat 3 +hard  
|  
| Initialize the hard disk  
|  
| dinit 3 +hard  
|  
| Check and mark bad blocks  
|  
| dmark 3 track,head >3:/bad_blks  
| dcheck 3 +mark  
|  
| Backup the contents of your BOOT floppy  
|  
| backup 1:/ 3:/ +all s=c      BOOT disk in drive 1  
|  
| Change your search order  
|  
| search 3  
|  
| Backup the remaining diskettes  
|  
| backup 1:/ 3:/ +all s=c      Repeat for each floppy  
|  
| Set OS image file to boot and hard disk driver to use  
|
```

```
boot 3:/netboot/osname d=/drivers/disk.at
```

```
| Make QNX partition bootable
```

```
fdisk 3
```

3.5 Creating A Login Password File

QNX has the capability of supporting more than one user at a time, many of which may be located at remotely attached terminals. For this reason you may wish to restrict access to the system and its files by creating a file of allowable userids and passwords. This capability is activated by the command

```
"passon"
```

NOTE: Passwords are automatically turned on when a machine is booted from the network. This means you will **have to** create a password file if you are booting from the network.

After prompting for the login and password

```
Login: john
```

```
Password:
```

QNX attempts to open the file `"/config/pass"`. If it does not exist then access will be denied.

Users are identified by a userid and a group and member number which determines their permissions. QNX supports up to 256 groups each with a maximum of 256 user numbers ranging from 0 to 255. Group number 255 is privileged. For example, user number 0.21 has unrestricted access to all his own files but may only access the files of other users according to the permission fields of the file. This field is displayed by the FILES command when invoked with the `+verbose` option.

```
files +v
```

The default permission assigned to a file when it is created is "read". This allows other users to read it, but they may not write to it, append to it or delete it. If desired, you may remove the read permission by using the CHATTR command as follows.

```
chattr filename p=-r
```

The file "/config/pass" is created and updated using the editor (ED). This file should be created by the super-user (member of group 255) and should have read permission removed from it. It consists of a five line entry for each userid. All lines except the USERID line should be indented by ONE tab.

```
USERID
tab  PASSWORD
tab  USER NUMBER (Group.Member)
tab  LOGIN DIRECTORY
tab  LOGIN COMMAND
```

The USERID is the character string which must be entered in response to the "Login:" prompt. It may contain any characters including blanks and control characters.

The PASSWORD is the character string which must be entered in response to the "Password:" prompt. If this field is left blank, then this userid does not have a password and may be logged in without one. The password may also contain any characters including blanks and control characters. NOTE: the leading *tab* is still needed even if the password itself is absent.

The USER NUMBER is a pair of numbers between 0 and 255. The first number is the group, the second is the member of that group. These numbers identify the user to the system and determines his/her file access capabilities. Each userid will typically be assigned a unique user number. However, you may assign more than one userid the same user number if you wish them to have the same file capabilities.

The LOGIN DIRECTORY is the name of the directory to place the user at, after successfully logging in.

Finally, the LOGIN COMMAND is any QNX command which will be executed upon logging in. In a student environment this should be kept consistent for all students. Executing a user profile like "ec user.init" is a good choice.

Typical user.init

```
fortune
ap waiting list alarm
umail waiting
path !/cmds!/cmds1/!!
```


An exclamation mark (!) at the beginning of the LOGIN command will transfer directly to that command. This allows custom applications to be called directly when the user logs in without allowing them to return to a shell when the program terminates.

The following example illustrates a simple four user file.

```
superman
tab    clark kent
tab    255.255
tab    /user/superman
tab    ls
jasmith
tab    qwerty
tab    0.1
tab    /user/jasmith
tab    ec user.init
jqpublic
tab    0.2
tab    /user/jqpublic
dbase
tab    abc
tab    1.0
tab    /dbase/data
tab    dbase
```

superman has been assigned group number 255 and can do anything. Userid **jqpublic** has no password, and may be logged into by anyone. It also has no initial command to execute.

In an environment where there are many users it is convenient to assign userids based upon first and second initial followed by the last name. Mr. Jim A. Smith would be **jasmith** as in the example above.

The password file must be accessible to all users when they login. In a network, it is usually placed on the boot server node. Keep in mind that once this file exists then you must live by its login rules. Should you corrupt it you may not be able to log into the system. It should be pointed out that this security system is only effective once QNX has been booted, the disk containing the password file mounted, and the PASSON command executed (unless you booted from network). For this reason, access to the system unit should be restricted.

By convention, QNX places user directories under the directory "/user". There will typically be one directory for each user. The directory "/user" should be owned by the super user and only he may create new user directories directly under it. Each sub-directory under "/user" should be owned by the user assigned to it. The following procedure for adding a new user may be followed. In the example we will add a new user with name "jasmith" and user number 0.12.

1. Login to the super user.
2. Create the new user directory under "/user".

```
mkdir /user/jasmith
```

3. Assign the new user a number and give him ownership.

```
chattr /user/jasmith g=0 m=12
```

4. Read the file "/config/pass" into the editor and add a password entry for the new user.

```
jasmith
tab   xyz
tab   0.12
tab   /user/jasmith
tab   ec user.init
```

5. Write the file back and you should have successfully added a new user to the system.

3.6 Mounting a DOS or Second QNX Partition

You can mount more than one hard disk partition at a time. When you mount the second or third partitions you replace the name of the driver with a *d=drive* option. The *drive* is the drive number of an already mounted partition. The partition to mount is selected using the *pa=os type* option. For example, the following will mount two QNX partitions and one DOS partition on an AT.

```
mount disk 3 /drivers/disk.at pa=qnx
mount disk 4 d=3 pa=qny
mount disk 6 d=3 pa=dos
```

The DOS partition can only be accessed by the DOS emulator or DFS package.

We have reserved the *os_type* numbers 7, 8 and 9 for QNX. When you mount a disk you may either specify its number or one of the following symbolic names.

pa=qnx *same as* pa=7
pa=qny *same as* pa=8
pa=qnz *same as* pa=9
pa=dos *same as* pa=1 or pa=4

3.7 More than One Physical Hard Disk

Most of the hard disk controllers can support more than 1 physical hard disk. To mount a second disk use the *d=* and *p=* option on the mount command. For example, assume that an AT controller has two 20 Meg disks attached to it, and both have QNX partitions on them.

```
mount disk 3 /drivers/disk.at pa=qnx - disk 1
mount disk 4 d=3 p=2 pa=qnx - disk 2
```

The *p=* option specifies which *physical drive* to use. The default is to use physical drive 1, which is why it was not required in the initial installation.

3.8 Changing Operating System Defaults

The BOOT command allows you to specify a configuration file (*[c=config_file]*) which lets you change some of the operating system defaults such as the number of open files supported.

A *config_file* is created and maintained by the OSCONFIG command. It is described in detail in the "Utilities" manual. It takes a file name as an argument. For example:

```
osconfig 3:/config/sys.cfg
```

would create a configuration file for a single non-networked machine. It does not become active until you use the BOOT command to set it. For example:

```
boot 3:/netboot/os.2.10pcat c=3:/config/sys.cfg
```

You may select any file name, however, when booting over the network QNX will automatically look for */config/sys.cfg.nn* with *nn* replaced by your node number. For this reason we recommend you adopt this convention, even on a single machine, although on a single machine you may omit the ".*nn*" extension.

`/config/sys.cfg` - Single machine.
`/config/sys.cfg.nn` - Network machine.

NOTE: When booting from hard disk you should not remove the os filename, the driver filename or the config file name.

`/netboot/osname`
`/config/hdisk.cfg`
`/config/sys.cfg`

You may copy a new file on top of them but NEVER remove them (FREL, RM, ...) and then create a new one even if they have the same name. The BOOT command saves away the absolute start blocks of each of these files which will change once the file is removed.

3.9 Removing the QNX loader

The bootstrap loader created by the [+qnxloader] option of the BOOT command is compatible with that created by the DOS FDISK utility when a hard disk is partitioned for the first time. However, some compatible PC's or AT's may have a different bootstrap loader placed in block 1 of the disk.

If you have used the [+qnxloader] option with the BOOT command, and now find that DOS no longer boots and you wish it to, you can use the QNX FDISK command to remove the special signature (55AA) from the boot block. This will enable the FDISK command under DOS or QNX to re-write a new loader. For example:

fdisk 3 +r

The +remove option to FDISK should rarely be used and is not documented elsewhere.

4. Network Installation

Do not attempt network installation until you have read chapter 1 and completed the single machine installation in chapter 2 or 3.

Now that you are successfully running QNX on one machine you can start to connect several machines in a local area network. You will need a local area network card for each machine in the network and a version of QNX configured for the proper number of computers (nodes).

4.1 Hardware Installation

Please refer to the documentation which was provided with the networking card. It should have been supplied as an insert to be added to the standard manual.

4.2 Software Installation

Your first machine (boot server) must boot from disk. Each other machine in the network may optionally boot from disk or over the network. It is possible to configure work stations without any disks what-so-ever, using only remote disk storage.

If you have three or more networking cards we recommend that you first attempt to connect only two machines together.

In the following sections you will want to place the indicated commands in the system initialization file of the boot server. More information on the commands may be found in the "Utilities" manual.

STEP 1 Selecting Node Numbers

From QNX's point of view, each card must have a unique network address. QNX node id's may lie between 1 and 255. You should configure your node id's to start

at 1 and attempt to setup the node id's to be consecutive. Most users select node 1 to be their main machine (boot server).



STEP 2 Setting a Remote Search Order

When a machine boots over the network it will have its search order set to the machine it booted from. It will also inherit the date. Since all commands and files are coming from the boot machine you must define a remote search order for the boot server using the SEARCH command.

```
search 3 +remote
```

You should place the remote search command in the system initialization file for the boot server (eg. /config/sys.init.1).

STEP 3 Allowing Non-super-user Access

If you wish non-super-users to use the hard disk you must use the NACC command to allow network access. This command takes a list of drive numbers and device names to allow/disallow. To allow read/write on disk 3 enter the command.

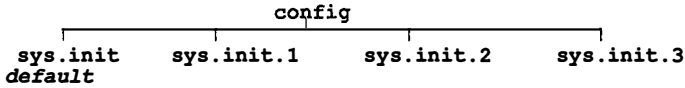
```
nacc 3 +read +write
```

Since the search order of a node which is booting over the network will be to the boot server, you should place the network access command in the system initialization file for the boot server. Otherwise machines will hang when they boot because they can't find an appropriate `sys.init.nn` file in their search order.

STEP 4 Creating System Initialization Files

As each machine in the network boots it will attempt to execute the file

`/config/sys.init.nn` where *nn* is the node number of the machine. If that file does not exist then it will try `/config/sys.init` (no node postfix).



We recommend that `sys.init` be a relatively harmless set of generic commands similar to the one shipped on your boot diskette. This allows you to quickly boot a new machine on the network. Once that machine is booting successfully you can then create a custom `sys.init.nn` for it.

See the description on "Creating a System Initialization File" in chapter 3.

STEP 5 Create a Password File

Since any machine which has booted over the network from a boot server always has passwords enabled, it is necessary that a password file `/config/pass` exists somewhere in the search order.

See the description on "Creating a Password File" in chapter 3.

STEP 6 Starting the Netboot Task

Before a machine (boot server) can boot other machines, you must run a background task which accepts boot requests from the network and downloads the contents of operating system image files (*os_file_names*) to the requesting machines. This task is called NETBOOT and should be started by typing its name followed by an ampersand (&) on the command line.

netboot &

You will probably wish to place this command near the end of the system initialization file for the machine acting as the boot server.

QNX operating system boot image files (*os_file_names*) are kept under the directory `/netboot`. To prevent confusion, the names of the files in this directory have been chosen to reflect the operating system they represent.

os. n.n r type

where **n.n** - is the version number (2.1)
r - is the release number
single digit 0 .. 9
type - is a hardware machine type:
pcat - IBM PC, AT, PS/2 or compatible
atp - IBM AT, PS/2 or compatible running in protected mode

The following are typical boot filenames.

os.2.10pcat - PC or AT, real mode, release 0.
os.2.10atp - AT, protected mode, release 0.

4.3 Summary

The following chart summarizes the steps necessary to install QNX on a network.

```
| Install Networking Cards  
| Set a Remote Search Order  
search 3 +remote  
| Allowing Non-super-user Access  
nacc 3 +read +write  
| Create System Initialization Files /config/sys.init.nn  
| Create a Password File If One Does not Exist  
| Start the Netboot Task  
netboot &
```

4.4 Starting the Poller

A poller is a program which continually sends poll messages to all active nodes in the network to verify their sanity. This poll message contains a list of all nodes which the poller believes are functional. When a node receives a poll message, it updates its list of UP nodes and replies with a positive acknowledgement. If the

poller does **not** receive a positive acknowledgement back after several poll cycles, then that node is removed from the **UP** list. This updated list will then be sent to all nodes during the next poll cycle.

When a node detects the transition from **UP** to **DOWN** on a poll cycle it will automatically close all virtual circuits set up between itself and that node. This has the effect of informing the associated QNX tasks in communication with tasks on that node that the remote task has died. They will then reclaim resources (close files, release memory, ...) which may have been tied up by that remote task.

The poller is not necessary for the network to function. Its purpose is to provide network wide resource reclamation in the event of a catastrophic error (machine crash, power failure, ...) at a time when that machine has allocated resources across the network. The most common example would be the opening of files on a disk on another node. Turning the power off on a machine which is sitting idle would not require the poller.

Any node in a QNX network is capable of being a poller, however **you should only run ONE poller in the network at a time.**

```
poll & - start poller
```

You may wish to place the command

```
alive +newboot
```

in the system initialization file of each node which boots over the network. This will cause all nodes in the network to close all virtual circuits open to your node then mark your node as **UP**. It is effectively like a crash followed by an instant boot. The **POLL** command is documented in the **Utilities** section of this manual.

4.5 Recovery Without the Poller

If you elect not to run the poller, or if the poller node crashes you may close all virtual circuits from your node to the crashed node using the **KILL_VCS** command. This command takes a single argument which is the node which crashed. The command

```
kill_vcs 3
```

tells the node on which the command is executed that node 3 has crashed.

You may tell your own network administrator that a node is up or down using the ALIVE utility. Your local network administrator will refuse to attempt to set up a virtual circuit to any node it believes is down.

```
alive +2 +4 -3    - set status of node
                  2 and 4 as UP
                  3 as DOWN
```

Both the ALIVE and POLLER command contain many options which you should read about in the Utilities section of this manual.

4.6 Troubleshooting QNET Network Installation

During your first attempt at setting up the network, you may encounter some difficulty. The following are some tips which can aid you in diagnosing problems:

For purposes of example, we will assume the boot server is node 1. This means:

- That the operating system image files (*os_file_names*) are in the /netboot directory somewhere in its search order (most probably on its hard disk).
- You have a remote search order defined, given proper network access, and have at least a "/config/sys.init" file.
- The NETBOOT task is running in the background.

Let's describe what should happen when you try to boot node 2:

- The "Node 2" message appears in the center of the screen on node 2.
- A flashing '.' appears just to the right of the "Node 2".
(while it flashes, an operating system image is being downloaded)
- The screen will clear.
- Either [1]/config/sys.init.2 will be executed on node 2, or if it was not found, [1]/config/sys.init will run.
- The "Login:" prompt will appear.

4.6.1 Diagnostic Tips

Although you may not want to do the following all the time, you may need to use them as indicators during diagnostic sessions:

- 1) If you run the NETBOOT command (on the boot server) with the verbose option, it will indicate on the boot server's console when a boot request has occurred, by which node, for which operating system image, and also if NET-BOOT believes it was able to transmit it properly. For example:

```
netboot +v &
```

- 2) When you first try to get a node to boot, the contents of the `/config/sys.init.nn` file should be kept very simple. Try using the `sys.init` shipped on the QNX boot diskette or a one liner like:

```
type "I love QNX"
```

- 3) Later, when creating more complex `/config/sys.init.nn` files, place the word **verbose** as the first command to execute. This will echo each command to the console as it executes. If the booting process stops somewhere during the execution of this file, then you will see which command is at fault.
- 4) If you suspect a particular card is not functioning properly, replace it with another (if you can) and observe if the behavior remains the same or not. In this situation you should also remove any non-essential cards from your computer in case there is a conflict which you might not be aware of. In some cases you may even wish to test the card in another computer.
- 5) If you are setting up a card on an existing network, you can use the TSK and ALIVE (if the poller is running) commands on the other nodes to see if they can communicate with this node.

4.6.2 Symptom Checklist

SYMPTOM: The machine boots from disk instead from the network.

REASON: The card is not programmed correctly. You should select the "Boot from Network" option.

SYMPTOM: The screen does not clear, and the "Node *nn*" never comes up.

REASON: The networking card was not found during the power-up ROM scan performed by the BIOS. You should try another card if you can, or perhaps another machine. You may need need to upgrade the ROM BIOS in your computer.

SYMPTOM: The "Node *nn*" comes up, but is wrong.

REASON: The card is not programmed correctly.

SYMPTOM: The "Node *nn*" is OK, but the '.' never comes up.

REASON: Run NETBOOT in verbose mode to see if the boot server gets the request and if it thinks it was sent properly (indicated by an "OK").If NETBOOT cannot find the operating system image file, "Boot file not found" is printed on the console of the booting node.

If NETBOOT does think the OS was sent properly, then the interrupt configuration on the card may be a problem. Try programming the card to use a different interrupt.

SYMPTOM: The "Node *nn*" comes up, the '.' flashes, the screen clears, but the machine hangs.

REASON: Since the screen cleared, the operating system was most probably downloaded. If your characters are echoed back to you when you type at the keyboard, then the system is alive, but most likely hung either trying to find the `"/config/sys.init.nn"` file, or if it found it, then it might have hung on a command within it.

If it hung trying to find it, then verify that the boot server has properly defined the **remote search order** and given **network access** for the drive on which the system initialization file resides.

If it hung during execution of the system initialization file, try putting the **"verbose"** command at the beginning of it to have the commands echoed as they execute.

Another possibility is the wrong operating system for the type of machine has been downloaded.

SYMPTOM: You can't login.

REASON: This is probably because the password file `"/config/pass"` cannot be found. Either it does not exist, or if one does exist, it is not in your current search order.

5. Terminals, Modems and Printers

This chapter will cover serial and parallel ports on your computer. There is also a special discussion on smart serial cards. You may wish to refer to the Technical Note called "Operating System Limits".

5.1 Serial

Terminals, modems and serial printers are connected to your computer using a serial port. This serial connection is referred to as an RS-232C asynchronous link. Some computers have 1 or 2 ports built-in while others require you to plug in an additional adapter card. Most manuals refer to the first two serial ports as COM1 and COM2. There are a number of multi-port cards on the market which have as many as 8 serial ports on one card. With QNX, these cards allow you to support many users on a single machine. You can also have one or more tasks monitor these lines for use in a process control application.

This section deals with "dumb" serial cards. Smartcards are described at the end of this technical note.

5.1.1 I/O ports and Interrupts

After booting, QNX scans for serial ports at the following IO port addresses in the following order.

3F8 (*com1*)
2F8 (*com2*)
280 288 290 298 2A0 2A8 2B0 2B8 3E8 2E8

Note: This is the default list and may be changed by running the OSCONFIG program.

QNX will assign device numbers and names to each port found as follows.

1 st - \$mdm
2 nd - \$term1
3 rd - \$term2
... - ...

Use the MOUNT command with no arguments to see the names and tty numbers recognized by the system.

QNX does NOT poll serial lines but runs them in interrupt mode. COM1 (port 3F8) uses interrupt 4 while COM2 (port 2F8) uses interrupt 3. A multi-port card generates a single interrupt for all (4 to 8) ports. You can usually select which interrupt is generated. QNX has built in support for most multi-port serial cards. The technical support group can advise you on cards which are in use by existing QNX customers.

Note that some multi-function cards have two serial ports on one card. These are usually COM1 and COM2 and each port uses its own interrupt (4 or 3). They do not share a single interrupt like a 4 or 8 port serial card.

In a PC/AT each card must be assigned a unique interrupt, they cannot be shared. In a PS/2, interrupt sharing is allowed.

By default, QNX only enables interrupt 4. To use a serial adapter which has been configured for interrupt 3, you must explicitly turn on interrupt 3 with the STTY command.

```
stty inton=3
```

If you wish to use an interrupt other than 3 or 4 you must tell QNX to map that interrupt to the serial handler by copying vector 4 into the vector selected. To configure a card on the AT to use interrupt 5 you would execute the command:

```
stty intcp=4,5 inton=5
```

If a 4 or 8 port serial card is installed, try to configure it to respond to I/O address 280h. The 4/8 port card will only generate one interrupt for all ports.

If you run across a card which uses I/O ports other than those scanned by QNX, you can change QNX's list by running the OSCONFIG command described in the QNX manual.

5.1.2 Baud Rates and Parity

When QNX boots, it sets the baud rate and parity on all serial ports to 1200 baud, 8 bits and no parity. You will have to set the baud rate of your serial card to match that set by the terminal, modem or serial printer. For terminals and printers we recommend 9600 baud if possible. Modems will typically be 300, 1200 or 2400 baud. You must also set the number of data bits to 7 or 8 and the parity to one of even, odd, mark, space or none.

Data Bits	Parity	
7	even	
7	odd	
7	mark	
7	space	
8	none	<i>Allows 8 bit data transmission</i>

These parameters are changed using the STTY command. For example:

```
stty baud=2400 >$mdm
stty baud=9600 >$term1
stty baud=9600 par=even bits=7 >$term2
```

5.1.3 Connecting Cables (DTE vs DCE)

Most serial cards have been configured as a DTE (Data Terminal Equipment). This means that the electrical interface thinks that it is a terminal and it can not be directly connected to another terminal. It was meant to connect to a modem or a printer. If you wish to connect a terminal to a DTE serial port you must make a special cable that makes your port look like a DCE (Data Communication Equipment). Some cards have options to allow this. Most of the single port cards do not. If the card does not have this option then you must connect what is sometimes called a NULL modem in the serial line between your PC and the terminal.

A NULL MODEM is a cable which interchanges some of the lines. You can make a NULL modem for a 25 pin serial connector as follows. Remember that to connect a modem or a printer you can use a straight through cable.

The signals on the following pins must be exchanged on one of the two connectors to make a NULL MODEM:

pins 2 &3	Transmit and Receive data
pins 4 &5	Clear-to-send and Request-to-send
pins 6 &20	Data-set-ready and Data-terminal-ready

The remaining pins should be connected directly from one connector to the other. NOTE that in most instances, not all 25 pins need to be connected. For most RS-232-C terminals (NOT Current Loop), only pins 2 through 8 and pin 20 need to be connected.

25 PIN to 25 PIN

Terminal	PC	25 PIN	PC	Modem
1 -----	1	ground	1 -----	1
3 <-----	2	tx data	2 ----->	2
2 ----->	3	rx data	3 <-----	3
5 <-----	4	rts	4 ----->	4
4 ----->	5	cts	5 <-----	5
20 ----->	6	dsr	6 <-----	6
7 -----	7	ground	7 -----	7
8 ----->	8	carrier	8 <-----	8
6 <-----	20	dtr	20 ----->	20

NULL MODEM CABLE

STRAIGHT THROUGH CABLE

There are a number of serial cards on the market that have a 9 pin connector. The pin assignments are as follows.

Pin	Signal
1	carrier
2	rx data
3	tx data
4	dtr
5	ground
6	dsr
7	rts
8	cts
9	ring

If you must connect a serial port to a device which provides only tx data, rx data and ground, it is a good practice to jumper the handshake lines as follows:

ground	1	_____
ground	7	_____
tx data	2	_____
rx data	3	_____
rts	4	_____
cts	5	_____
dsr	6	_____
carrier	8	_____
dtr	20	_____

This jumpering will only allow software flow control to be used.

5.1.4 Setting QNX Line Editing Options

QNX can be configured to perform line editing on input and the expansion of certain characters on output. The default settings are suitable for connecting a terminal to QNX and having a user login.

If you wish to connect a serial printer you should disable line editing and echoing of input characters.

```
stty -edit -echo >$printer_device_name
```

You may also wish to disable one or more character expansions using the following options to STTY:

-etab	Expand tabs to spaces set every 4.
-ers	Expand a record separator to a carriage return linefeed.
-edel	Expand a rubout to a space, backspace, space.

Many programs, such as LIST, will automatically reset and restore these expansions as needed. If you leave them enabled this has the advantage of allowing you to directly copy text files to the printer and have them come out looking OK.

QNX separates text lines in files with a single record separator character (hex 1e). If you do not have the ERS option enabled then this character will not be expanded into a carriage return linefeed resulting in output which is difficult to read.

When connecting a modem for outgoing calls only, you should disable both edit and echo as with a printer. If you wish to support incoming calls, then leave the options in their default state for a terminal. You may wish to use the COMM program for incoming calls. It is described later on in this technical note.

5.1.5 Flow Control

QNX supports both software and hardware flow control on input or output. This can be explained with an example.

Assume that a serial printer is connected which can print at 120 cps. A QNX program can send data to the printer at 9600 baud which is about 960 cps. To prevent loss of data the printer will tell QNX to stop sending just before its input buffer is full. It can do this in one of two ways. With software flow control it will send an XOFF (Ctrl s) character telling QNX to stop sending. When it is ready for

more data it will send an XON (Ctrl q) telling QNX to resume output. With hardware flow control it disables one or both of the hardware signal lines called CTS (clear to send) and DSR (data set ready). When it is ready for more data these lines are re-enabled. Most terminals and printers can be set to use either hardware or software flow control. On a terminal, software flow control is nice in that you can type Ctrl s at the keyboard to stop QNX output and then a Ctrl q to resume. This is the default set on the console/keyboard. With a printer there is a small danger that you might get an XOFF but not an XON. This can be fixed by turning the printer off and on line (the printer should send an XON when it goes online) or by telling QNX to continue anyway using the STTY command.

stty -paged >device_name

When QNX is blocked on output because of flow control, it is referred to as "paged".

Input flow control is the same process, but in the opposite direction. QNX will send an XOFF/XON (software) or toggle RTS/DTR (hardware) to prevent its input buffer from overflowing if data is received faster than a QNX application can process it.

The defaults at power up are software flow control on output only. You can change these using the following STTY options.

+ -hflow	- Select software or hardware flow control.
+ -iflow	- Enable/disable input flow control.
+ -oflow	- Enable/disable output flow control.

5.1.6 Logging In From A Terminal

To login on a terminal, you must ensure that the parity and baud rates are set correctly, then type a Ctrl-z. If you do not get a login prompt, try typing a Ctrl-x (delete any garbage in the input buffer) and a Ctrl-q (unpage incase a Ctrl-s has paged output). If this fails, then type some letters. If they fail to echo, you may not have properly configured your serial port(s).

5.1.7 Logging In From A Modem

You must configure your modem to auto-answer mode and simply follow the procedure for logging in to a terminal. While this works it has two major disadvantages.

- 1. If you hang up, or the line drops, you will not be logged off.**
- 2. You must always access the modem at the correct baud rate.**

With 300/1200/2400 baud modems popular, you may not know which baud rate to select.

To solve these problems, you may wish to run the COMM command. This program will determine and set the baud rate and parity for the remote user. The COMM command is described in the utilities section of your QNX manual. It supports both dumb modems and Hayes compatible modems. When carrier is lost, COMM will kill off your program, effectively logging you off. To detect loss of carrier it is important that the modem raise and lower the carrier detect line (pin 8) and that your modem cable connect that line.

5.2 Outgoing Calls

To place an outgoing call you will use the QTALK program. Please refer to the QTALK documentation for details. If you require a more advanced terminal emulator you may wish to purchase an extremely slick product called QTERM.

5.2.1 Problems

Most problems connecting serial ports can be traced to one of the following:

1. **Interrupts on the serial card are not configured properly or two cards are sharing an interrupt.**
2. **The I/O port on the serial card is not configured properly or two cards are sharing an I/O port.**
3. **The cable is incorrect. You may need a straight through cable or one that crosses some of the signal lines.**
4. **The baud rate is not correct.**
`stty baud=baud_rate >$device_name`
5. **You have not enabled interrupts if the card uses an interrupt other than 4. The symptom of this is a single character being echoed.**
`stty inon=int_num`

5.3 Parallel

Connecting a parallel printer to QNX is very easy. You do not need to worry about interrupts, flow control or special cables as you do for serial connections. QNX supports up to 2 parallel ports which are called

\$lpt, \$lpt2

It is rare to see more than one or two in a system.

5.4 Smartcards

A smartcard is a multi-port serial card with an onboard processor and a large amount of memory for buffering incoming and outgoing data. There are no interface standards for smartcards, as there are for dumb cards, and you must run a custom driver to use these cards. These cards may be more expensive than dumb serial cards. They should be considered when:

- You have more than one channel of input data at 9600 baud.
- You have input data which cannot be flow controlled and comes in bursts greater than 250 characters.
- You are performing nearly continuous output at 9600 baud and must maintain that data rate on several channels.
- You need to support more than 10 serial ports. The smartcards typically allow up to 4, 8-port cards in a system, for a total of 32 possible serial ports.

If you are only connecting terminals to QNX, then you probably do not need a smartcard. If you are unsure, contact QNX technical support for some guide lines as well as an up-to-date list of supported smartcards. The smartcard drivers are available from the manufactures (the best choice) or from QUICS (the Quantum update system) for download. In many cases, full source code for the driver is also available. The driver runs as a background task. A typical invocation of a driver would look like this:

```
smartcard a=d000 i=5 p=300 &
```

```
Start driver to talk to a card with:  
dual ported memory at segment d000  
interrupt 5  
I/O port 300
```

Wait a few seconds for the card to be initialized and type the MOUNT command with no arguments to see the new set of device names.

6. Devices

QNX provides a sophisticated terminal handler for all character devices attached to it (keyboard, display, serial communication ports). This chapter is a summary suitable for end users.

6.1 Terminal Input

The following information concerns itself with the default options set up for both the console keyboard and attached terminals in a multi-tasking system. Due to keyboard limitations of many terminals, some keys and options will apply to the console keyboard only.

6.1.1 Alt, Shift and Ctrl

These three keys do not generate data themselves, but modify the data generated by other keys.

The shift key is used to type upper case letters and the symbols shown on the upper positions of the keys with dual markings.

The Ctrl key is used to generate the control codes defined in the ascii character set.

The Alt/Compose key, when used as a shift, will turn on the eighth data bit for any key combination pressed. When typed as a data key it sets up a two character compose sequence. Compose characters allow you to enter extended codes for foreign language characters and special symbols. This is described later in this section. This key is only available on the console keyboard and special QNX compatible terminals.

The entire character set for the console keyboard is contained in appendix A. The first table will apply for most terminals. The second table is specific to the console keyboard and special QNX compatible terminals.

Users with a foreign language keyboard on their PC should refer to the KEYBOARD command in the utilities section of this manual.

6.1.2 Caps Lock, Num Lock

These three keys are used to select a particular input mode.

The Caps Lock key converts all lower case letters to capital letters until you press it again. This key is available on most terminals.

The Num Lock key lets you type the numbers on the keypad without having to shift. If you do use the shift key in this mode, it will allow you to access the scroll (arrow) codes on the bottom of the keys. This key exists on PC keyboards which share the keypad for numbers and cursor keys. Like the Caps Lock key, Num Lock key is a toggle. Typing the key a second time will turn off Num lock mode.

If you run the CLOCK program, it will display the state of the Caps Lock, Num Lock and Alt key in the upper right hand corner of your console display.

6.1.3 Type-Ahead

The QNX Operating System provides full type-ahead with echo and line editing. As long as QNX is functional, your input keys will be seen and remembered. As a general rule, whenever you enter a line, there will be a program waiting for input which will consume it. You act as the producer and the program acts as the consumer of input lines. If there is not a program waiting on the keyboard, you may still continue to type input lines which will be saved until a program (usually the command interpreter(SH)) asks for them. For example, if you wanted to execute 4 programs, and each program took between 1 and 5 minutes, you could type all four commands one after another and go for a coffee while they finish.

The operating system has a limit of 254 characters in the type-ahead buffer. QNX will stop echoing your characters if this limit is exceeded. This can be mistaken for a software crash. If your keyboard does not echo, first try typing a Ctrl-x to delete the last line in the input buffer. If you get echo but no command output try typing a Ctrl-q to cancel output flow control. This can occur if you type or your terminal sends a ctrl-s. If this does not help, refer to the chapters on "Connecting Terminals To QNX" and "Tips On Using QNX".

6.1.4 Line Editing

Your keyboard/terminal can operate in one of two modes called EDIT mode and RAW mode. In RAW mode, every character typed is immediately given to the program asking for input. The program itself acts on each character received. The full screen editor runs in this mode. In EDIT mode, when you enter characters on a line, they are not passed to any program waiting for input until you type the carriage return key. This allows you to correct any typing mistakes you may make. The keyboard (terminal) input routine examines each character you type and recognizes several keys as special. The command line interpreter (SH) runs in EDIT mode. The following keys are special in EDIT mode. They are summarized in appendix A.

RUBOUT (PC: back-arrow TERMINAL: Del or Rubout)

This key deletes the last character typed. The cursor will move one position to the left, erasing the character at that position. Only characters which are typed by the user will be erased. The cursor will not back up over a system prompt.

CANCEL (Ctrl-x)

This key deletes the entire line.

LEFT/RIGHT (PC: left, right arrows TERMINAL: backspace, n/a)

These two keys allow you to move your cursor back and forth over any text on your line, changing it if desired. On a terminal these keys must be defined using the STTY command since there is no clear standard. Most terminals support a backspace key but some may not support a non-destructive forward space.

DELETE (PC: Del key TERMINAL: Ctrl-k)

This key deletes the character at the current cursor position. The rest of the line will slide over to fill the gap. On a terminal this key must be defined using the STTY command.

INSERT (PC: Ins key TERMINAL: Ctrl-n)

This key toggles insert mode. In insert mode any characters that you enter will be inserted before the character at the cursor. Insert mode is always turned off when you enter a CARRIAGE RETURN a BREAK or a CANCEL. On a terminal this key must be defined using the STTY command.

TAB (tab key or Ctrl-i)

This key will enter a tab character in your buffer, and echo the required number of spaces to move you to the next tab stop. In QNX, tab stops are set every four columns with the first stop set in column five. If you try to RUBOUT a tab character, you will only remove one displayed space on your screen even though the TAB may have been displayed as several spaces. The TAB character in your buffer will have been deleted. It is your display which expanded the TAB but failed to un-expand it on the RUBOUT.

CARRIAGE-RETURN

This key causes the entered line to be made available to the user program. Carriage returns are automatically mapped into RECORD SEPARATORS (hex 1e) by default. QNX separates lines by a single record separator character in text files. This differs from DOS which separates lines with two characters (carriage return and linefeed) and Unix which uses a single linefeed.

PAUSE (Ctrl-s)

Suspend further system output to the terminal (until Ctrl-q).

CONTINUE (Ctrl-q)

Continue allowing any system output to be displayed.

BREAK (PC: Ctrl-Break TERMINAL: Ctrl-c or BREAK key)

This key halts execution of the currently running program which is associated with the terminal. Any opened files will be closed, and all memory will be returned to

the operating system. Note that some programs (such as the full screen editor) are capable of trapping BREAK, preventing program termination via this key. .

ESCAPE (Ctrl-z)

This key suspends (but does not kill) all running tasks associated with this terminal and creates a new shell allowing you to enter commands. This key allows you to escape from any program which has not defined its input stream to be RAW. When you terminate the shell with a Ctrl-d you will return to the program you suspended. Do not use Ctrl-z as a break even though the immediate effect (you get a \$ prompt) is similar. You will eventually run out of task entries or memory since the suspended programs remain around. Programs which do run unedited (ED) may have been designed to look for the Ctrl-z and invoke a shell.

REBOOT (PC: Ctrl-Alt-Shift-Del - 4 keys held down)

This combination will cause QNX to reboot itself. Your open files are NOT closed, and any which were open for write will be left busy. You can get rid of them by using the ZAP command. This method of restart should only be used when you have no running programs or in the rare event that QNX should "*check out*" (CRASH).

DEBUG (PC: Ctrl-Alt-Esc - 3 keys held down)

This combination will suspend execution of your program and invoke the low level assembly debugger if it has been loaded. If the debugger has not been loaded then this key combination is ignored.

6.1.5 Compose Characters

The Alt/Compose key on your PC is a dual function key. When used as a shift it will turn on the 8th data bit of the shifted character. For example

a - generates hex code 61
Alt a - generates hex code E1

If it is depressed then released, WITHOUT typing any other key, it sets up for a two character compose sequence. The next two characters are combined into a single 8 bit character code. Appendix A contains a list of compose key translations. The following brief table lists a few.

Alt (release) p i --> pi symbol
Alt (release) + - --> plus or minus symbol
Alt (release) e ' --> e accent grave symbol
Alt (release) 1 2 --> hex 12 character
Alt (release) b a --> hex ba character

The last two illustrate the inputting of exact hex values.

If a program wishes to accept compose characters and differentiate them from function and cursor keys it should enable option EFUNC. This will precede all function and cursor keys with a hex FF. This is necessary since the codes returned for function and cursor keys overlap the composed characters. Programmers should refer to the documentation on the GET_OPTION library routine in the C compiler manual.

6.1.6 Input Gate

A unique feature of the QNX terminal handling software is termed the INPUT GATE, which prevents output from being mixed with user input on the display.

If the INPUT GATE option is enabled, the operating system will stop all program output to the display while you are entering a line of text. When the line is ended with a carriage return, or the user deletes the entire line (via CANCEL or RUBOUT), system output to the terminal will again be allowed. This option may be turned off using the STTY command.

stty -igate

Programmers should note that this option is ignored when you turn off the EDIT option. The option is meant for the console and users connecting to terminals. It is not used for serial ports configured as raw communication lines.

6.1.7 Recalling Command Lines

Due to QNX's circular 256 character buffering scheme on input, it is possible to back up or move forward in the buffer recalling previously entered lines. The number of lines you may back up is determined by the length of each input line. Editing a line you have backed up to may overwrite part of the following line. You should think of the up and down arrow as a means of rolling backward and forward through your typed lines. On a terminal you will have to define these keys using the STTY command.

RECALL PREVIOUS LINE (PC: up arrow TERMINAL: Ctrl-u)

This key will redisplay the previously entered line, allowing you to edit it if necessary. Typing it again will recall the line previous to this and so forth. Warning: Attempting to recall a line after a command which runs without line editing (Full Screen Editor, Talk) may result in garbage being displayed.

RECALL NEXT LINE (PC: down arrow TERMINAL: Linefeed or Ctrl-j)

This key is the opposite of the up arrow and moves you forward through your type ahead buffer. These keys must be defined using the STTY command if they are to be used on an attached terminal.

6.2 Terminal Output

6.2.1 Attributes

Your console display supports several special attributes.

- inverse video
- underline
- blinking
- intensify
- colour

These attributes may be accessed in QNX by sending special control character sequences to the display. These sequences apply to the console only. It is unlikely that an attached terminal on QNX will support the same escape sequences to access their special attributes. QNX's escape sequences were chosen to provide a user friendly interface, not for compatibility with any manufacturer's line of terminal equipment.

Users interested in writing programs which are terminal independent should read the documentation on the TCAP utility and library.

Not all machines which QNX supports will have the same set of display attributes. QNX will map and support these attributes in a machine independent manner. Selecting an attribute which does not exist will have no effect.

6.2.2 Output Escape Sequences

The QNX video display driver treats the Ascii ESC code (hex character 1b) as a special control code. The character following an ESC is examined, and if recognized will cause some control function to be performed. An unrecognized character following an ESC will be passed to the display unmodified, so to print a real ESC character on the screen, you may send two ESC codes.

Following is a list of the ESC sequences which are recognized by the QNX display driver. Note that due to hardware limitations some combinations may not be possible. In particular inverse and underline are mutually exclusive on the IBM monochrome display. Since the QNX shell echoes each received character you can experiment with these sequences by simply typing them in. The most commonly typed sequence is a Ctrl-I which will clear the screen.

Esc (Turn on inverse video mode. This will also swap the foreground and background colours set by the ESC @ sequence
Esc)	Turn off inverse video mode
Esc [Turn on underline mode
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 A	Move cursor UP (no wrap)
Esc B	Move cursor DOWN (no wrap)
Esc C	Move cursor RIGHT (no wrap)
Esc D	Move cursor LEFT (no wrap)
Esc E	Insert line
Esc F	Delete line
Esc H	Move cursor HOME
Esc I	Reverse linefeed
Esc J	Clear to end of screen
Esc K	Clear to end of line
Esc 1	Switch to console 1
Esc 2	Switch to console 2
	etc...
Esc 8	Switch to console 8
Esc S	Save colour and attribute information
Esc R	Restore colour and attribute information
Esc @ <i>fc bc</i>	Define the foreground and background colour of displayed characters. Both <i>fc</i> and <i>bc</i> represent a single digit taken from the colour table below.
Esc ! <i>fc bc</i>	Sets the fill colour which is used for the bottom line when scrolling and the entire screen when clearing via form feed. <i>fc</i> is the colour of the cursor, <i>bc</i> is the fill colour
Esc = <i>r c</i>	Position cursor at the screen coordinates (<i>r</i> , <i>c</i>) which are given by the two characters following the '='. <i>R</i> is the row (0-24) and <i>c</i> is the column (0-79). Both <i>r</i> and <i>c</i> are offset by hexadecimal 20 (a space). Therefore, <i>rc</i> = <SPACE><SPACE> is row zero, column zero
Esc Y <i>r c</i>	Same as ESC =

As an example, the sequence

Esc@71Esc!71

will set your character and fill attribute to white on-blue.

COLOUR CODES

<u>digit</u>	<u>colour</u>	<u>digit</u>	<u>colour</u>
0	black	4	red
1	blue	5	magenta
2	green	6	brown
3	cyan	7	white

You should set your colour preference using ESC @ and ESC ! in your sys.init file and follow it with an ESC S to save it away. All QNX applications which play with the colour will print an ESC R to restore the saved colour before exiting.

In addition, the following characters will cause some special action to be performed, rather than printing a character on the display...

CR	Carriage-return. Move cursor to column 0 of current line
LF	Line-feed. Move cursor down 1 position, and scroll the screen up 1 row if necessary.
FF	Form-feed. Clear the display to the background attribute defined by ESC ! sequence. Cursor returns to top left corner (0,0).
BS	Back-space. Move the cursor left 1 position. Wrap around to the previous line if necessary.
BELL	Bell. Generate a tone for a fraction of a second.
HOME	Move cursor to top left corner of screen.
up-arrow	Move cursor up one position. Wrap around to bottom if necessary.
down-arrow	Move cursor down one position. Wrap around to top if necessary.
left-arrow	Move cursor left one position. Wrap to end of previous line end if necessary.
right-arrow	Move cursor right one position. Wrap to beginning of next line if necessary.

7. Full Screen Consoles

Virtual consoles give you the ability to switch between several tasks, each running on a different virtual console. For example, a developer may choose to edit programs in one console, compile programs in another console and execute them in yet a third console. Multiple consoles offer a definite advantage over running programs in background in that you may examine your output and gain control at any time. An advanced work station might run a VT100 emulator, 3278 emulator and another QNX application in three consoles, all concurrently, with the ability to rapidly switch between applications.

QNX can support several virtual consoles on your PC's physical screen. Each virtual console is one full screen and may be thought of as a console to which you may attach your keyboard. Your keyboard is always attached to the console which is being displayed. The other consoles may still receive output data from tasks.

7.1 Mounting Consoles

When QNX boots, it assumes only one console. You must explicitly MOUNT every other console you wish to use.

By default, QNX has a built-in Console Driver which will work with CGA and EGA in text mode. Other Console/Graphics Libraries are available which support text consoles in 43x80 mode, these are:

<code>/drivers/glib.ega</code>	-	IBM Enhanced Graphics Adapter.
<code>/drivers/glib.vga</code>	-	IBM Video Graphics Array.

If you wish to use consoles in these non-standard text modes you must mount the proper Console/Graphics Library **before you mount any of your extra consoles.** If you mount them afterwards you will be limited to a screen size of 25 by 80.

The operating system allocates system memory to store each of the consoles that have been mounted when they are not displayed. **When you mount your first extra console,** QNX will query the Console/Graphics Library which is mounted at that time, to find out what maximum amount of space is required to store a console (for example, the default 25x80 needs 4K, while 43x80 requires 6.8K). Once the buffer size has been determined, it is memorized and used for all other subsequent console mounts.

QNX supports many standard devices. To determine exactly how many devices your current version supports you should refer to the Technical Note called "Operating System Limits". Each device has two names. A \$ttypn which identifies device *nm* and a symbolic name like \$mdm which can also be used to reference the device. For example, \$tty3 is usually associated with \$mdm. The following illustrates how the device entries are intended to be used with physical devices:

0 \$con	- keyboard	7 \$term4	- serial 5
1 \$lpt	- parallel 1	8 \$term5	
2 \$lpt1	- parallel 2	9 \$term6	
3 \$mdm	- serial 1	10 \$term7	
4 \$term1		11 \$term8	
5 \$term2		12 \$term9	- serial 10
6 \$term3	- serial 4		

In practice, not all devices are used. These unused device entries may be converted into virtual consoles using the MOUNT command.

For example, the following commands will create three new consoles for a total of four including \$con. The device number of the device entry to re-use may be given (eg. 2 for \$lpt2, 5 for \$term2, etc) followed by a new name to assign to the console. If you do not specify the device, then a free one will be selected for you. If you specify a device, be **VERY** careful to specify only **unused** device numbers. Typing the MOUNT command with no arguments will display the devices in your system.

mount

The new name may be up to five characters in length.

Device not specified

mount console \$con1
 mount console \$con2
 mount console \$con3

Device specified

mount console 2 \$con1
 mount console 11 \$con2
 mount console 12 \$con3

You will wish to place MOUNT CONSOLE commands in your system initialization file so they will be executed automatically each time you boot.

7.2 Switching Consoles

Holding down the Ctrl key, the Alt key and then pressing the large plus key on the keypad will step you to the next console while a Ctrl-Alt minus (keypad) will step you to the previous console.

People may enter a particular console by holding down the Ctrl key, the Alt key and then typing the console number (1 through 8).

Programs may switch the displayed console by writing an escape sequence to any of the consoles:

ESC 1 - displays the first console \$con
ESC 2 - displays the second console
ESC 8 - displays the eighth console
etc ...

NOTE: This second scheme may seem to work if typed in by users, but data characters are left in the device buffer, which usually confuses the application that reads them.

At the time of this printing, certain EGA video cards on the market perform a bit of magic allowing them to auto-switch between several graphics modes. They also allow you to connect a monochrome or CGA monitor to the controller while in EGA compatibility mode. The card does all the gray-scale interpretation required to make it work. They implement this cleverness by generating an I/O check which generates a NMI (non-maskable-interrupt) which they vector into their ROM BIOS on the card. This trick does NOT work in protected mode and will cause the operating system to fault, usually the first time you attempt to switch consoles.

Fortunately, you can usually disable the INTELLIGENT capability of these cards and make them BE a MDA, CGA, HGA or EGA controller connected to the appropriate monitor type.

Any program may be executed in the active (displayed) console and will behave as expected. However, some programs which write directly to the screen memory or use the BIOS calls (int 10h) may cause problems if you switch to another console while they are executing. The rule of thumb to use in these cases is never switch to another console unless you are sure these programs are not going to try to print to the screen while you are in the other console. Programs which use the QNX I/O facilities or the high speed video interface will not have a problem. This includes all Quantum utilities and products.

The stty command supports a +hold option which will hold a task from running unless it is the active console. This may be useful for a console in graphics mode where applications always write directly to the screen memory. Programmers can control this option from within their programs.

7.3 Initiating Commands in Other Consoles

Typing a Ctrl-z in a new console will cause the login prompt to be displayed just as it would for any terminal.

Occasionally it may be necessary or convenient to automatically initiate a command in the other console. This can be accomplished using the ONTTY command.

```
mount console 2 con1
```

```
ontty 2 login
```

OR

```
ontty 2 application
```

7.4 Consoles and Graphics

The default Console Driver which is part of the operating system, does not support any graphics. However, as mentioned above, there are several Console/Graphics Libraries which can be mounted to replace and/or augment its capabilities. These support graphics capabilities through the compiler library interface.

These allow you to have **1 console in graphics mode at any one time**, and some allow you to switch consoles when in graphics mode, while others prevent it:

- /drivers/glib.hga** - **Hercules Graphics Adapter.**
 - Text mode [25 by 80 monochrome].
 - Graphics mode [720 by 348 monochrome].

- /drivers/glib.cga** - **IBM Colour Graphics Adapter.**
 - Text mode [25 by 80 with 16 colours].
 - Graphics mode [320 by 200 with 4 colours].
[640 by 200 with 2 colours].

- /drivers/glib.ega** - **IBM Enhanced Graphics Adapter. ***
 - All modes from CGA Library.
 - Text mode [43 by 80 with 16 colours].

- Graphics mode [640 by 350 with 16 colours].

/drivers/glib.vga

- **IBM Video Graphics Array. ***
 - All modes from EGA Library.
 - Text mode [43 by 80 with 16 colours].
 - Text mode [50 by 80 with 16 colours].
 - Graphics mode [320 by 200 with 256 colours].
 - Graphics mode [640 by 480 with 16 colours].

*** Cannot switch consoles when in graphics mode.**

For more information see the Technical Note called "Console Shared Library".

8. Files

8.1 File Names

Like most operating systems, QNX stores information on disks in files. You identify a file by its name. On QNX, a filename may be from 1 to 16 characters chosen from the set of:

- the letters of the alphabet
- the numbers 0 through 9
- the period '.' and the underscore '_'
- hex characters 80 through AF (foreign characters)

Upper and lower case letters are distinct. The file "Report" does not have the same name as the file "report". Some examples of good and bad filenames follow:

John Doe	<i>Bad: contains a space</i>
27-d	<i>Bad: contains a dash</i>
John Doe	<i>Ok</i>
12.3.7	<i>Ok</i>
.	<i>Ok</i>
test.c	<i>Ok</i>

Unlike some operating systems, the period '.' is not treated as an extension separator by the QNX file system. It is just another possible character making up a filename. To improve clarity, the user may adopt the convention of suffixing filenames with an extension (a period followed by a few characters). The extension is NOT enforced by the file system, only the user and perhaps the commands that operate on files. In particular, the C compiler assumes that a C program that it has been asked to compile ends in ".c". If asked to generate an object file it creates a file with the ".c" replaced by a ".o".

8.2 Structure of Files

A file is regarded by QNX as a featureless, randomly addressable sequence of bytes. No other structure of files is assumed by the operating system. Commands which operate on files will in general impose their own structure. For example, commands which operate on text files (p, ed, size etc...) assume that a file consists of variable length lines, with each line terminated by an ascii RECORD SEPARATOR character (hexadecimal value 1E).

A file may vary in size from 1 to 1,099,511,627,771 bytes (about 1 terabyte). A file cannot be larger than the disk which contains it.

The smallest unit of storage allocation for a file is one 512 byte disk sector. Files need not be contiguous or preallocated to their maximum foreseeable size. QNX will enlarge a file's size as required and if possible, allocate contiguous space for it.

There is a file called "bitmap" on each disk which contains the sector allocation of the disk. You may display this file in hexadecimal format using the "dump" command or pictorially using the query command.

```
dump 3:/bitmap
OR
query 3
query 3 +d
```

The total number of files a user may have on a disk is only bounded by available disk storage.

8.3 Directories

There is a type of file maintained by the operating system called a *directory*. Directories contain the mapping between the names of files and the files themselves and thus impose a *structure* on the file system as a whole. Each user has a directory of files and is free to create subdirectories to contain groups of files which are related. A directory may be read by the user, but may only be written by privileged programs and as such, is controlled by the operating system. A directory may contain a maximum of 32,000 files, however, if it contains subdirectories then each subdirectory may also contain 32,000 files. This rule is then applied to the subdirectories. For all practical purposes the number of files you create will only be limited by the amount of disk space you have available to save their data.

The structure of the file system is hierarchical and is often called "tree structured" since diagrams depicting it look like inverted trees with their root at the top. To clarify this, let's consider an example where a user (gord) has three directories; one for C programs, one for Basic programs and one for business documentation. His file structure would look like Diagram 3.1.

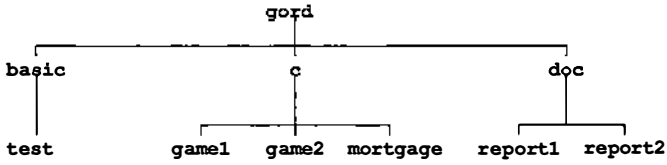


Diagram 3.1

The "leaves" (test.c game1, etc..) are *files* and the internal nodes (gord, c, basic, and doc) are *directories*. Let's complicate our example and assume the user wants to separate his C game programs from his C business programs. The user also generates a series of reports each month and wants each of the monthly reports kept separate. This may be achieved by the structure in diagram 3.2.

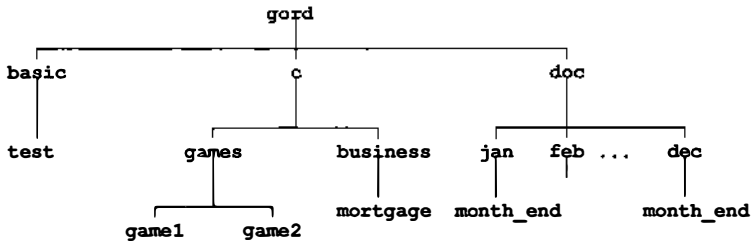


Diagram 3.2

Finally, let's also subdivide the directory doc into years (diagram 3.3).

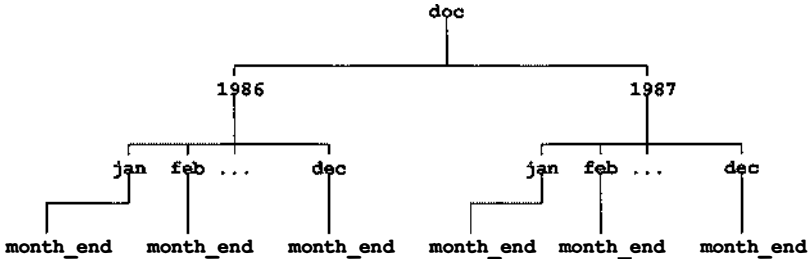


Diagram 3.3

Note that you can use the same filename for different files under different directories. The file system allows you to structure your files in a natural hierarchical arrangement. To achieve this with a flat file system would be very awkward.

The previous examples have dealt with userid "gord". Since a disk may contain many userids, "gord" is in fact just a directory under another directory called "user". A three user disk is illustrated in Diagram 3.4.

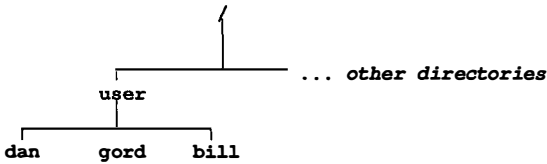


Diagram 3.4

The '/' at the top is a very special directory called the ROOT directory and each disk has one. Your Boot diskette contains directories under the root as illustrated in Diagram 3.5.

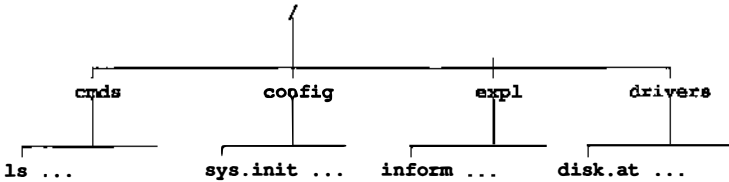


Diagram 3.5

Executable commands are kept under the directory "cmds" and configuration files are kept under the directory "config". Explain files are kept under the directory "expl". Disk drivers are under "drivers".

8.4 Pathnames

When you specify a file to QNX, you must present it in a form which uniquely identifies it within the hierarchical directory structure. This form is called a *pathname* and consists of a sequence of directory names separated by slashes, '/', and ending in a filename. For example the file "ls" under the directory "cmds" could be written as the pathname

`/cmds/ls`

Likewise, the file "month_end" under the directory "1987" under the directory "jan" in diagram 3.3 could be written as the pathname

`/user/gord/doc/1987/jan/month_end`

The leading slash indicates to QNX that it should begin its search at the ROOT of the file system on your disks. In the "month_end" example, it will start its search on the first drive in your search order and look for the directory "user". If it fails to find it, it will then look on the next drive and so on for each drive in your search order. Once it locates "user" at the ROOT of a disk, it will look in that directory for the directory "gord", and if successful, it will look in directory "gord" for directory "doc" and so on until it finally locates the file "month_end".

If you are opening a file for write, once the system has matched a directory on a disk (ie: "user" in this case), it will not restart its search on another drive if it fails to find a subdirectory or the file on its search down the tree. This behavior is only important if you have two disks with a directory by the same name at the ROOT. QNX solves this ambiguity by always stopping its search on the first disk on which it finds a match at the ROOT. This is not true for files being opened for read, read/write or execute. In these cases, the search will continue across all drives looking for an exact match.

The default search order of your drives depends on how you boot.

Boot	Search Path
floppy	1st floppy, 2nd floppy if present
hard disk	hard disk only
network	node from which you booted.

You may change this using the SEARCH command. For example if you had a ramdisk as drive five and a hard disk as drive three you could tell QNX to search drive five first by issuing the command:

```
search 5 3
```

Refer to the SEARCH command in the utilities section of your binder for more information.

8.4.1 Specifying a Drive as Part of Your Pathname

You can override QNX's sequential search of your disk drives and lock onto a particular drive by prefixing your filename with a drive number followed by a colon (:). As an example

```
backup 2:/user 1:/user
```

would invoke the backup command with the directory "user" on drive 2 as the source and the directory "user" on drive 1 as the destination. The command

```
backup 1:/ 2:/ +all
```

would back up all structure on the disk in drive 1 onto the disk in drive 2. Finally, the command

```
dump 2:/bitmap
```

would only look on drive 2 for the file "bitmap" under the ROOT.

8.5 Your Current Directory

The previous pathnames all began with a slash and cause a search to be made starting at the ROOT of the disk. If you leave off the leading slash, the search will be made starting at your current directory. When you log in, your current directory is set to point to the directory you logged into. For example, if you logged into "gord" and gord had a file named "costs", you could name it via its full pathname starting from the root

```
/user/gord/costs
```

or its relative pathname starting at your current directory

```
costs
```

When QNX memorizes your current directory, it does more than just remember a pathname. It remembers the drive and block number on the disk of your current directory. In a QNX network the node number is also remembered. This results in very fast access to files accessed by this method. In practice you will find that 90% of your pathnames will be relative, NOT absolute from the ROOT.

If at any time you need to specify your correct directory in a utility (such as BACKUP), it may be indicated with a null string (""). This is also true when opening the current directory from a program.

```
backup "" /dir - backup current directory to /dir
```

8.5.1 Changing Your Current Directory

QNX provides a command to allow you to change your current directory to point to any directory on a disk. The command is

```
cd pathname
```

where "cd" stands for "change directory". The pathname should end in a directory NOT a filename. For example, lets say you wanted to work on your 1987 month end report for January. Using the structure illustrated in Diagram 3.3 you could reference the file "month_end" by

```
/user/gord/doc/1987/jan/month_end
```

or if you had logged into userid "gord", you could access it relative to your current directory by

doc/1987/jan/month_end

If you intend on typing this pathname more than once you should probably change your current directory to

cd doc/1987/jan

and reference the file as

month_end

At any time you may return to the directory you logged into by executing the CD command with no argument.

cd - will return to login directory

You can print your current directory using the PWD command.

pwd - will print current directory on your screen

8.6 Moving Up the File Structure

QNX allows you to move up the file structure by preceding your pathname with up-arrows (^). Each up-arrow moves you up one directory level. If your current directory was at

/user/gord/doc/1987/jan

you could reference your February **month_end** file as

^feb/month_end

and your 1986 jan **month_end** file as

^^1986/jan/month_end

Finally, you could "cd" yourself to "gord" with

cd ^^^

which will move you up three levels.

8.7 File Attributes and Permissions

Each QNX file has an attribute field and two permission fields associated with it. Each field acts as a mask or a filter limiting access to a file.

The *attribute* field limits the access modes which may be applied to the file by any user INCLUDING the OWNER of the file, the GROUP LEADER and the SUPER USER.

The *group permission* field limits the access modes which may be applied to the file by users which are in your group.

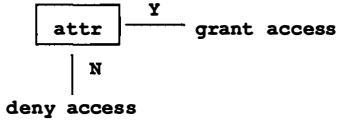
The *other permission* field limits the access modes which may be applied to the file by other users.

Changing permissions allows you to control access to include

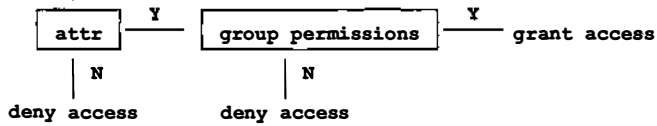
- yourself only
- users which are in your group
- users which are not in your group
- all users

Opening a file performs the attribute and permission checks. If any check fails, then the open will fail with a "permission denied" violation. The following diagrams illustrate the sequence of checks made.

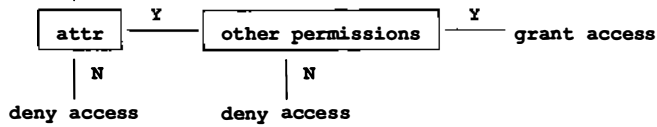
ACCESS BY
OWNER OF FILE



ACCESS BY
USER IN SAME GROUP



ACCESS BY
USER IN ANOTHER GROUP



The attributes/permissions of a file are:

- READ** - You may open the file for read.
- WRITE** - You may open the file for write. Previous contents are lost. If you do not write to the file it will be removed when you close it. This also gives delete capability. Files may be opened for read/write in which case the previous contents are not affected by the open operation.
- APPEND** - You may open the file for append. Any new information you write will not overwrite any current information but will be added to the end.
- EXECUTE** - You may open the file for execute. This allows you to execute the file as a load module or a command file without the sh. All QNX commands have the attribute of execute.
- MODIFY** - You may modify the attributes/permissions of the file.

When a new file is created it will by default have the attributes and permissions of:

attributes: READ WRITE APPEND MODIFY
permissions: READ

In addition, if the file is created by the linker and is executable, then both the attribute and permission field will also have EXECUTE capability set.

The access capabilities on a directory are quite different from those on files. Directories are maintained by the operating system and a few specialized privileged commands. A user should never be allowed to write to a directory, and executing a directory is absurd. The following access capabilities exist for directories:

- READ** - You may open the directory for read. This allows you to discover the names of the files it contains. For example, the 'ls' command opens the indicated directory for read to list its contents.
- CREATE** - You may create new files under this directory.
- BLOCK** - You may not pass through this directory when matching a pathname.
- MODIFY** - You may modify the attributes/permissions of the directory.
- DIRECTORY** - This bit indicates that the file is a directory. It is controlled by the operating system and a few privileged commands.

When a new directory is created it will by default have the attributes and permissions of:

attributes: READ CREATE MODIFY DIRECTORY
permissions: READ DIRECTORY

The uses of these access capabilities are best explained by some examples.

1. As a user you do not want other users creating new files under your directories. Once created they could remove write permission and you the owner of the directory could not remove them. You can stop this by ensuring that CREATE permission does not exist on your directories. This is the default. Shared directories like '/tmp' must be available for creating new files by all users and should therefore have CREATE permission.
2. You do not want users to be able to access any of your files regardless of the permissions on the files. Simply ensure that BLOCK permission is set on the directory and users will be blocked from accessing all files (or sub-directories).
3. You do not want users to see the names of your files. However, you do want one or two users or special commands to be able to access files they know the name of. Simply remove READ and BLOCK from the directory. Adding BLOCK would not only prevent users from seeing your files, but they could not even access them if they guessed their names.
4. You wish all members of your group to be able to read and write a file you own. Simple add READ and WRITE group permission to the file.

The capability of MODIFY is the most powerful since it allows you to change the other capabilities of the file. You must have MODIFY to use the CHATTR com-

mand on a file. Note that if you remove MODIFY from the attributes, then the file will be fixed for all time at its current capability set. For this reason the CHATTR command takes care never to remove MODIFY unless explicitly told to do so!

8.8 User Numbers

The *owner* of a file is indicated by a group number and a member number between 0 and 255 inclusive. All members of group 255 are SUPER USERS and are privileged in the sense that they are not restricted by the permissions field of any file, only the attributes. They can therefore read/write/execute/append/modify other users files, regardless of the file's permissions. Likewise, member 255 of each group is a GROUP LEADER and may access the files of all members in that group regardless of the permissions. All other users are constrained to access privileges determined by both the attribute and permission fields of any files they do not own. The owner of a file as well as its permissions are listed by the FILES command with the +v (verbose) option.

You will be assigned the group number of the super user if there is no password file. If a password file exists (see section 'Using QNX') then you will be assigned a group and member number from that file when you login.

In a multi-user system, all system directories should be owned by the super-user. This includes the directory "/user". However, each directory under "/user" should be owned by the user assigned to it. Since the super-user owns "/user", only he may create new user directories under it. To change the ownership, the g= and m= option of CHATTR should be used. For example

```
mkdir /user/jsmith
chattr /user/jsmith g=2 m=21
```

will give ownership of directory /user/jsmith to user number 2.21.

If you wish to restrict access of potentially dangerous commands, you should remove general execute and read permission from them. For example

```
chattr p=-er /cmds/mount
```

would prevent anyone but a super-user from executing the mount command, assuming that /cmds/mount is owned by user 255.255.

8.9 Device Names

Until now a pathname has referred to devices which are capable of maintaining a directory structure, the most common being a disk drive. QNX also supports devices which contain no internal structure and communicate a character at a time.

These are referred to as character devices and consist of

- terminals
- line printers
- modems and serial communication lines

On the Personal Computer, the keyboard and display will normally act as your terminal. The display is not strictly a character device, but for convenience it is treated that way by the software.

Some devices are read or write-only while other devices may be read from and written to. Your display is an output device and your keyboard is an input device. QNX pairs closely related devices like this and refers to them by a single name. To identify a character device you start your *pathname* with a dollar '\$'. The devices supported by QNX are

- \$con** - Your keyboard/display (console)
- \$lpt** - Your line printer
- \$lpt2** - A second line printer
- \$mdm** - First serial port (COM 1)
- \$term1** - Second serial port (COM 2)
- \$term2** - Third serial port
- ...
- \$null** - Returns end of file on input and throws characters away on output.

These may also be referred to as \$tty0, \$tty1 ... etc. There may be more or less devices configured depending on the version of QNX you have running on a particular machine. The MOUNT command with no arguments will list those devices you have configured and are installed.

mount

QNX goes to some length to treat character devices and files as similarly as possible. This is called device independent I/O. What this means is that any command which expects to read or write to a device may also read or write to a file without modification. The reverse is also true. As an example the CP (*copy*) command is typically used to copy files.

cp *source_file destination_file*

Either or both pathnames may be a character device.

`cp source_file $lpt`

Anyplace where a *pathname* is acceptable, you may usually enter a device name. A device name is considered a subset of the class of pathnames.

8.10 Network Access to Files and Devices

The QNX network links all machines together. Each node (machine) contains a file system which handles all file requests to disks attached to it. These requests may come from local tasks or remote tasks running on other nodes. The network extends the syntax for a pathname to include the node number on which the file or device exists. The node number is enclosed in square brackets and precedes the pathname. For example, the P command

`p 3:/user/dtdodge/data`

would look on disk drive 3 on my node while

`p [4]3:/user/dtdodge/data`

would look on disk drive 3 of node 4. This represents a totally specified pathname. It should be noted that there are no artificial restrictions placed on the indicated drive. Drive 3 may be a floppy, hard disk or ramdisk. If a driver exists, it may be an optical disk or x.25 linking servicing the entire network.

A missing node number will always default to your local node.

8.10.1 Remote Search Order

If the drive is omitted then

`p /user/dtdodge/data`

would use the search order on my node to search for the file on my drives. When a node is specified

`p [4]/user/dtdodge/data`

then the search order on node 4 is used and a search is made across its drives. Each node maintains two search orders, one for *local* requests and one for *remote* requests.

<code>search 5 3</code>	- search drive 5 then drive 3
<code>search 3 +remote</code>	- search drive 3 for remote access

In the above example, assume drive 3 is a hard disk and drive 5 is a ramdisk. The local users will search their ramdisk then their hard disk. A user on another node will only search the hard disk.

8.10.2 Automatic Remote Searching

You may specify a node in place of a drive to the SEARCH command. The command

```
search 5 [4]
```

would search drive 5 (probably a ramdisk) then search on node 4 using its *remote* search order. This would be a typical search order for a node without its own hard disk. Node 4 would probably contain a hard disk and be thought of as a file server. It is possible for node 4 to also specify a node in its search command.

```
search 3 [5] +remote
```

This would cause another indirection to node 5 and a search through its drives using its remote search order. If node 5 contains a node in its search order it is skipped. Only two levels of indirection are allowed. The second level allows a means of redirecting all file system requests to another node in one operation. Assume that node 1 was a file server and node 32 is a backup file server. The normal remote search order for node 1 might be

```
search 3 +remote
```

All requests could be referred to node 32 with the command

```
search [32] +remote
```

The following diagram illustrates a complex search across several nodes.

```
NODE 1
search 3 [2] [3] 1      - local search

NODE 2
search 2 3 +remote     - remote search

NODE 3
search 2 [4] +remote   - remote search

NODE 4
search 2 +remote       - remote search
```

An access to `/path` on node 1 would attempt to open the following files:

```
3:/path
[2]2:/path
[2]3:/path
[3]2:/path
[4]2:/path
1:/path
```

8.10.3 Remote Current Directory

Your current directory may be on any drive on any node in the network. Simply precede the CD command with a node number

```
cd [4]/user/dtdodge
```

The PWD command will print out the pathname of your complete current directory, indicating both the node and drive.

```
pwd
```

8.10.4 Mounting A Remote Disk

There are a small subset of QNX commands and library routines which take a drive number as an argument. Their syntax does not allow for the specification of a node number. To overcome this restriction the MOUNT command allows you to mount a virtual disk. The command

```
mount remdisk 7 n=4 d=3
```

will map all requests to disk 7 into node 4 disk 3. You may remap a virtual drive at any time.

```
mount remdisk 7 n=4 d=3
dcheck 7
mount remdisk 7 n=6 d=1
dcheck 7
```

The concept of virtual disks is in fact the *only* means of remote access offered by most other networking systems. You might wish to consider it as an alternative to node numbers in your search command. The following will give you your own ramdisk and equivalence virtual drive 3 to the hardisk on node 1.

```
mount remdisk 3 n=1 d=3      - remote hardisk
mount ramdisk 5 s=64k       - local ramdisk
search 5 3
```

8.10.5 Network Devices

A device name may be prefixed by a node number in the same manner as a path-name.

```
cp file $lpt                - local line printer
cp file [4]$lpt             - line printer on node 4
```

8.10.6 Limiting Network Access

Now that we have given you the syntax and mechanism for completely generalized access to any device in the system, it becomes necessary to **restrict** access. You may not wish to share your local ramdisk and floppy with another work station. Likewise, you may consider a dot matrix line printer connected to your computer as your exclusive property. The NACC command can be used to control every device attached to your machine. The default is NOT to grant access. You may enable READ and or WRITE permission as desired. For example

```
nacc 2 $mdm +read +write    - read and write for drive 2
                             and $mdm
nacc 3 +read                 - read access only for drive 3
nacc CPU +write              - allow remote task creates
```

The last example allows other nodes to create and execute programs on your node. This very advanced feature separates QNX as a true fifth generation operating system. Please refer to the documentation on the NACC command for further information.

9. The Command Interpreter (Shell)

The command interpreter, referred to as the "shell", acts as the interface between the user and QNX. To the shell, a command is a sequence of words separated by spaces.

command arg1 arg2 ... argn

The first word is treated as the name of an executable file (the command name) and all other words are considered as arguments to the command. Upper and lower case letters are different. If the command name begins with a slash then a search is made for the pathname exactly as specified, otherwise, the string "/cmds/" is prefixed to the pathname first. If after prefixing with "/cmds/" the pathname can not be found then the prefix is removed and a search is made under your current directory. For example, the command

p data

would cause a search for the executable file

/cmds/p

and only if that failed would it look for

p

under your current directory.

This default search path may be changed by the PATH command which is described later in this chapter.

The arguments are collected by the shell and passed to the command as strings.

9.1 Shell Prompt

The shell will prompt for input with a character followed by a space. The character prompt is different for the three classes of QNX users.

\$ - Super User
- Group Leader
% - Regular User

The PROMPTT shell command may be used to precede the prompt with your tty number. This is useful on the console with multiple consoles.

9.2 Command Input/Output Redirection

When a command begins execution, it opens three files referred to as its standard input, standard output and standard error output. By default these all refer to your terminal. The shell is able to change these default assignments by recognizing the constructs:

```
>pathname      - redirect standard output to pathname
>*pathname     - redirect error output to pathname
>>pathname     - redirect and append standard output to pathname
>>*pathname    - redirect and append error output to pathname
<pathname     - redirect input from pathname
```

As an example, the command "ls" ordinarily lists on your terminal the names of the files in your current directory. The command

```
ls >my_files
```

would redirect the output to the file "my_files". Likewise

```
ls >$lpt
```

would print them on the line printer. An interesting example of redirecting your standard input allows you to invoke the line editor with a script of editor commands.

```
led file <script_file
```

Although the redirection character and following pathname appear to be an argument to the command, they are in fact interpreted completely by the shell and not passed to the command at all. Thus no special coding is required within the command to handle input/output redirection. The redirection applies only to the command executed and not subsequently executed commands.

Programs which are started in the */config/sys.init.nn* files have their standard input set to \$null. This means that programs which read data from standard input will get EOF. If you wish to start a task which can read from standard input you must redirect it. For example:

```
myprog < $tty0
```


You can also use the ONTTY command:

```
ontty $tty0 myprog
ontty $tty5 my_other_prog
```

C programmers may note that redirected I/O pathnames are passed to their program as argv[argc] thru argv[argc + 2]. The current path is in argv[argc + 3]. A value of zero indicates no redirection.

9.3 Quoting

Since arguments are separated by spaces it is not directly possible to pass spaces as arguments or a single argument containing a space to a command. To overcome this limitation the shell allows you to enclose in double quotes any argument which you want passed to a command as is. Some examples of quoting follow:

```
cmd " "           - invoke command with one argument
                  consisting of a space
cmd ""           - invoke command with one argument
                  consisting of a null string
cmd ">file"       - invoke command with one argument
                  consisting of the string ">file"
cmd "abc def" ghi - invoke command with two arguments
                  consisting of "abc def" and "ghi".
```

In the third example the '>' is protected from the shell and cmd will not have its output redirected. In the fourth example the string "abc def" with an embedded space is passed as one argument, not two. In all cases the double quotes are stripped off the argument before passing it to the command.

There are cases where you may only wish to protect one character (perhaps a leading '>' or '<' on an argument) or the quote character itself. The shell understands the backslash character as a character which protects the character following it from the shell. The backslash character is NOT passed as part of the argument. You may pass a backslash to a command by entering two of them side by side. The first simply protects the second. Therefore the cmd line

```
cmd \"hello" \\ >file
```

would pass three arguments to cmd.

```
"hello"  
 \  
>file
```

A backslash does not lose its special significance when in double quotes.

9.4 Filename Generation

The shell treats the characters star (*) and question mark (?) special in the context of a filename. They are often referred to as global filename characters. A ? in a filename indicates that any character may occupy that position. A * in a filename indicates that zero or more characters may occupy that position. A search is made of all filenames in the current directory. Any files which match will replace the filename which contains the * and or ?. It is often convenient to think of the * and ? forming a pattern which may match zero or more filenames. Since the pattern may be expanded into many filenames there is the possibility that you may overflow the 512 byte maximum command line. If this happens you will get the message LINE TOO LONG. The following examples will illustrate a few simple expansions.

```
type *      - list all files in current directory  
type *.c   - list files which end in '.c'  
type a*    - list files which start with an 'a'  
type *a*   - list files which contain an 'a'  
type *.*   - list files with a one character dot extension  
type p=*   - type the string 'p=*'
```

The last example illustrates that a filename will only be expanded if the characters around it are valid filename characters. The equals (=) is an invalid filename character, so no expansion was attempted. This check allows commands like WS, FILES etc to accept an argument of the form

p=pattern

without having to worry about the shell interpreting and expanding the pattern itself. Note that the shell can not handle an embedded * in a pathname.

type /dir/* - not supported

You should use extreme caution when using *. The innocent looking command

frel *

will release ALL files at your current directory level!

9.5 Querying

When a command is followed by a question mark '?', it indicates that the command should explain itself.

In C programs, the command is informed by setting the number of arguments passed to it to zero. Each command normally receives at least one argument which is the command name itself.

All commands supplied by QNX have adopted this convention. The command

ls ?

will inform the LS command to print a terse usage message. This will often save you from pulling out your utilities manual when you are unsure of a command's syntax. The ? should be the last character on a line.

9.6 Background Tasks

In QNX, the ampersand '&' character is used to run commands in background (deferred). The task will be detached from your console and both the new command and your command shell will run concurrently. The task-id of the new task will be displayed on your console (this can be disabled) followed by the \$, # or % prompt from the shell indicating that you may continue to execute more commands. If you do not redirect the standard input for the new command it will be changed to the device \$null, NOT your keyboard. The new task created will be immune from keyboard breaks. If you wish to explicitly kill it before it terminates, you may use the BREAK, KILL or SLAY command. For example

slay dragon

will kill the task with task name "dragon". If you are not the SUPER USER you may only kill tasks which you yourself have created.

The priority of the new task may be decreased by following the '&' with a minus '-' sign. This is often used for programs you do not want to interfere with your keyboard response. This is especially true when you are in the full screen editor.

cc cprog &-

The LIST command is an excellent example of a command which can be run background to increase your productivity. Issuing more than one LIST command (from one or more terminals) will result in the requests being queued up waiting for the printer. The following would queue up 3 list requests.

```
list prog.c &  
list x=myfiles &  
list *.c &
```

In a network system or even a large single machine multi-user system you should consider using the spooler instead of direct calls to list.

9.7 Multiple Commands On a Line

You can place several commands on a line by separating them with a semicolon (;). For example.

```
ls ; task
```

If you wish to pass a real semicolon to a program you must quote it with double quotes or a backslash.

9.8 Pipes

The or-bar (|) character is used to set up pipes between commands. This connects the standard output of one command into the standard input of the next command. The effect of the command

```
files | sort f=1
```

is the same as

```
files >file  
sort <file f=1
```

The commands are connected by a pipe, implemented in QNX as a temporary file. The file used will be placed under the directory '/tmp'. When the shell dies it will remove any pipe files it may have created. You may change the names of the temporary files used with the 'defpipe' command. For example, the command:

defpipe # $\$$

will use a temporary file under the current directory with the name of the task number of the shell. A simpler example

defpipe my_pipe

will use temporary files

my_pipe1 and my_pipe2

at the current directory level. Placing the temporary files on RAMDISK has the advantage of speed.

Commands which are designed to read their standard input, transform the data in some way, then write to their standard output are often referred to as *filters*. You may connect several of these commands together via pipes in which each command performs one operation on the data stream.

9.9 Comment Lines

Any input line to the shell which begins with a control character or a double quote will be ignored. These lines may be used for comments.

9.10 Executing Commands on Another Node

If a line is preceded with a node number in square brackets, the shell will attempt to run the following command on the indicated node. The default standard input, output and error will all be directed back to your terminal. The closing square bracket must be followed by a space to prevent confusion with a pathname.

- | | |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------|
| [4] task | - Run the task command on node 4 |
| [4]task | - Load the task command from node 4 and run it on this node |
| [2] [4]task | - Load the task command from node 4 and run in on node 2 |
| [2] cc [5]/dir/prog | - Compile on node 2 the program 'dir/prog.c' found on node 5. |
| [1] list data | - Run the list command on node 1. |
| [1] ontty 3 comm | - Run comm on node 1 and have it attach to \$tty3 on that node. |
| [1] ontty 3 db_admin & | - Run db_admin on node 1 in background. All messages will go to \$tty3 on that node. No vc's back to originating node. |

[1] login

- Create LOGIN on node 1.

The last example would allow you to effectively login to node 1 as a virtual console. All subsequently executed commands would now be run on node 1. Typing a Ctrl-d would log you off and return you back to the shell on your current node. You may only create tasks on other nodes if

- You are the Network Super User (255.255).
- OR - That node has CPU write set using the NACC command.
- OR - The command is owned by the Super User and has been patched to have overriding remote create.

Make sure you have NACC permission to write to your own console from the remote node!

9.11 Command Files

The command interpreter normally takes its input from the terminal. In the case of a command file it will take its input from a file containing QNX commands. During the processing of a command file a question mark (?) at the start of a command line will stop execution of the command file if that command returns a non-zero status. A command file can call other command files.

9.11.1 Built-In Shell Variables

The shell has a number of built in variables which may be referenced as #c where c is a single character. The #c sequence is replaced by a character string. If there is no #c replacement then then #c is replaced with the null string. All built-in shell variables with the exception of #u are expanded as numbers in the current base. If the base is 16 they are expanded as 4 hexadecimal digits and if the base is 10 they are expanded as a 5 digit unsigned number. The default base is 16. For example:

<u>Base 16</u>	<u>Base 10</u>
0000	00000
000c	00012
8000	32768

You may turn the leading zeros on or off using the "zeros on/off" command.

The built in variables are:

- ## - number of arguments
- ##% - task number of shell

#\$	- task id of shell
#?	- exit status of last command
#&	- task id of last background task
#c	- cyclic number which changes each second
#g	- group number of user
#k	- accept a line from the keyboard
#m	- member number of user
#n	- node number
#t	- tty number
#u	- userid of user
#v	- qnx version number
#0	- name of shell command
#1 to #9	- arguments passed to shell command
#*	- all arguments #1 ... #nn

9.11.2 User Shell Variables

The shell supports 10 integer and 10 string variables which may be read and modified by the user. Integer variables by default expand as numbers in the current base (same manner as built in variables) and string variables expand as text strings. You can force an integer variable to expand in any base by placing a 't' or 'x' in the name as follows.

#i0	- expand i0 using current base
#it0	- expand i0 using base 10
#ix0	- expand i0 using base 16

You can perform arithmetic on the integer variables. They are modified using the SETVAR shell command and expanded using the following # notation:

#i0 to #i9	- integer user variable
#s0 to #s9	- string user variable

For example, the following line would read a line of text from the terminal and save it in a string variable.

```
setvar = s0 #k
```

Assignment and arithmetic is also straight forward.

setvar = i0 123	- set i0 to 123 using current base
setvar = i0 0t123	- set i0 to 123 using base 10
setvar = i0 0x123	- set i0 to 123 using base 16
setvar + i1 #i0	- add i0 to i1 updating i1

Please refer to the SETVAR command in the UTILITIES section of the manual under the SH utility.

9.11.3 Executing Shell Commands

Command files may be executed explicitly using the SH command

```
sh cmd_file arg1 arg2 ... arg9
```

or implicitly by placing execute permission on the file and typing its name as a command.

```
chattr a+=e p+=e cmd_file          - need only do this once  
cmd_file arg1 arg2 ... arg9
```

The arguments arg1, ...arg9 may be referenced as #1, ...#9 in the command file. For example a command file called **rename**, containing the line

```
chattr n=#2 #1
```

would be invoked by either

```
rename newname oldname  
OR  
sh /cmds/rename newname oldname
```

The first example requires that the file have execute permission.

```
chattr /cmds/rename a+=e p+=e
```


9.12 Local Shell Commands

The shell examines each command entered and interprets a few special ones itself, rather than passing them off to the operating system to execute. These commands either modify the shell's interface or are inconvenient to implement as regular system commands. Each command is described in the "Utilities" section of this manual. The following list summarizes each command's function.

back	- suppress background tid
base <i>base</i>	- base for number expansions
break <i>task_id</i>	- break a task
cd [<i>directory</i>]	- change directory
debug [<i>text</i>]	- debug a command
defpipe <i>path</i>	- pipe temp files
ec <i>shell_file</i>	- execute sh file
else	- conditional
endif	- end a block if
exit [<i>status</i>]	- exit shell with status
goto <i>label</i>	- transfer control
if <i>test cmd</i>	- conditional
kill <i>task_id ...</i>	- kill a task
ontty <i>tty cmd</i>	- create task on another tty
passon	- password protection
path <i>searchpath</i>	- change command search path
pause	- pause for carriage return
pri [+ -]<i>number</i>	- set priority
promptt	- display tty number
setvar <i>op var val</i>	- set variable
sharoff	- don't share code segments
sharon	- share code segments
shift	- shift # args down one
stype [<i>text</i>]*	- type arguments
sysup	- set system up flag
then	- conditional
trap [<i>label</i>]	- trap keyboard breaks
type [<i>text</i>]*	- type arguments
verbose	- toggle verbose mode
zeros on/off	- toggle verbose mode

9.13 Quick Reference to the Shell

The syntax of a shell command line is

[?] [!] [node] command arg1 arg2 ...

The **?** option will abort the shell on a bad exit status. The **!** option will transform into the command instead of creating an extra shell for its execution. If you do this in a shell file, no commands beyond the one transformed into will be executed.

SHELL VARIABLES

All number expansions are 4 digit leading zero hexadecimal by default

- ## - number of arguments
- #! - last exit status
- #\$ - shell task id
- ##% - shell task number
- ##& - last background task id
- ##0-9 - argument 0 to 9
- ##c - cyclic number
- ##g - group number
- ##k - take 1 line from keyboard
- ##m - member number
- ##n - node number
- ##t - tty number
- ##u - user name
- ##v - qnx version number
- ##* - ALL arguments
- ##i0-9 - integer variable
- ##s0-9 - string variable

LABEL

:name

IO REDIRECTION

- < - standard input
- > - standard output
- >> - standard output (append)
- >*> - standard error
- >>*> - standard error (append)
- | - pipe (uses tmp files)
- ;- - cmd separator

Leading TABS and SPACES are ignored and may be used for indentation

USER COMMANDS

- base *base*
- break *task id*
- cd *path*
- debug [*command*]
- defpipe *path*
- ec *command_file*
- else
- endif
- exit [*hex number*]
- goto *label*
- if +f *filename [cmd | then]*
- if +d *dirname [cmd | then]*
- if +m *filename [cmd | then]*
- if +a *nodeid [cmd | then]*
- if eq *str pat [cmd | then]*
- if ne *str pat [cmd | then]*
- if lt *str str [cmd | then]*
- if ge *str str [cmd | then]*
- kill *task id*
- ontty *tynum command*
- passon
- path *!path!path!....*
- pause
- pri [*+|-*]number
- prompt
- setvar = *var value*
- setvar | *var value*
- setvar + *var value*
- setvar - *var value*
- setvar x *var value*
- setvar / *var value*
- setvar % *var value*
- shift
- stype [*text*]*
- sysup [*text*]*
- type [*text*]*
- trap [*label*]
- verbose

10. Tasks

This section is an overview of the concept of multi-tasking in the QNX operating system. The discussion starts off very simply and progresses to the more complex issues and mechanisms within the operating system.

10.1 Introduction

It is by no means necessary to understand the underlying principles on which QNX is based in order to use QNX. For most users in a multi-user system, QNX will simply be an environment in which to run an application program on the same personal computer at the same time as someone else. How this is accomplished need not be known. However, a basic understanding is a definite asset.

In its simplest form, a task may be thought of as a program which accomplishes a particular task. In a multi-tasking operating system it is possible for several different tasks (programs) to be executing (running) at the same time. The operating system accomplishes this by sharing the computer among the various tasks competing for it. In a single tasking system the computer spends most of its time waiting on the user. At the speed at which most users enter characters, the computer could easily have read a thousand times that many characters. It is this speed that allows a multi-tasking system to appear to run more than one program at once. In fact the system takes turns running the various programs. The rules governing which task to run and when are complex but can be summarized as follows.

1. No task which is blocked (typically for input or output) will run. This means that a user program waiting for input will NOT compete with other tasks which are ready to run.
2. No task is allowed to run continuously for more than a fraction of a second (set by SLICE utility) if there is another task waiting. This prevents one task which is performing long complex calculations from locking all others out. Unlike point 1 above, the other tasks will be affected since they may now have to wait a fraction of a second for their turn to execute. In a multi-user system this is seen as a slight delay in response to commands. Should an appreciable number of tasks all consume their maximum time limit then response may suffer noticeably. In QNX, if eight users run a program which takes several seconds of dedicated processor time, then each program will appear to take eight times as long. Typically this does not occur because most tasks frequently request input/output.
3. Tasks may be assigned priorities such that higher priority tasks will never relinquish the processor to lower priority tasks. The time sharing of point 2 only applies to tasks at the same priority.

10.2 System Tasks

When QNX is booted, five system tasks are created. These tasks run at a priority which is higher than any user created tasks. Each system task performs a well defined function as follows.

TASK ADMINISTRATOR

This task is responsible for creating and destroying tasks. It is also responsible for allocating memory for the tasks. This task runs at priority one, which is the highest task priority in the system.

FILE SYSTEM ADMINISTRATOR

This task is responsible for the file system. It handles all requests to open, close, read and write to files. This task runs at priority three.

DEVICE ADMINISTRATOR

This task is responsible for all character devices attached to the system. This includes your terminal, the line printer, etc. It handles all requests to open, close, read and write to devices. This task runs at priority two.

IDLE ADMINISTRATOR

This task simply consumes any spare processor time. Since this task runs at the lowest possible priority in the system (15), it only runs when QNX has nothing else to do. It therefore does not affect system performance.

NETWORK ADMINISTRATOR

This task is responsible for communication over the local area network. It handles all data requests which must be transmitted over the network. This task runs at priority 3. If you do not have a networking card installed, this task is not needed and will terminate itself.

There are a other tasks which may be created after the system is running. Although not essential for the system to run, they provide valuable services. They are described below.

TIMER ADMINISTRATOR

This task is responsible for running a wakeup service. Any task can request to go to sleep for a specified period of time and this task will wake it up when that period elapses. It is also capable of signaling ports, setting exceptions and forcing a task ready. C programmers are referred to the file `"/lib/timer.h"` and the library routine `set_timer()`. The timer administrator may be started by typing (or placing in your `"/config/sys.init"` file) the command.

```
timer &
```

SPOOLDEV ADMINISTRATOR

This task creates virtual devices which spool information away to disk. This allows several people to use the device at the same time. When they close the device, the file is submitted to a real device in a first come first served order.

LOCKER ADMINISTRATOR

This task supports record locking and concurrent file sharing. The record locking is compatible with that found in Unix System V.

QUEUE ADMINISTRATOR

This task supports named queues which supplement QNX's normal methods of intertask messaging.

User tasks request services from these system tasks by sending messages to them. Upon sending a message the user task blocks waiting for the system task to receive the message, process it, then reply with some result. System tasks NEVER send to user tasks. The life of a system task can be summarized as follows.

- Block awaiting a message.
- When a message arrives, process the request and if the request can be satisfied reply indicating the result.
- If the request can not be satisfied immediately, then remember it and wait for another message. When the request can be satisfied, often as the result of another received message, reply to the waiting task at that point.

When running in a network, these messages may originate from other nodes on the network. The receiving task need not differentiate between local and remote messages.

10.3 Inter-task Communication

QNX supports three types of inter-task communication.

1. Messages

send (tid, snd_msg, rep_msg, size)
receive (tid, rcv_msg, size)
reply (tid, rep_msg, size)
read_msg (tid, rcv_msg, size)
write_msg (tid, rep_msg, size)
vc_create (nid, tid, size)

2. Ports

attach (port)
detach (port)
signal (port)
csignal (port)

3. Exceptions
 set_exception (tid, sys_exc, usr_exc)

10.3.1 Messages

A message consists of a sequence of data bytes from 0 to 65,535 bytes in length. Messages between nodes on a network are limited to the size of the virtual circuit buffer which was created for this communication.

The contents of a message will usually fit a predefined structure agreed upon by both the sender of a message and the receiver. A receiving task may receive more than one type of message, with each message having a different structure. In this case the first byte of the message will usually indicate its type. Based upon this message type the receiver may then apply the proper structure for extracting the various fields within the message. The operating system does NOT check for consistency in message types between tasks. It considers a message to be a sequence of data bytes and does not examine or assume any structure on the message itself. This view is consistent with that taken by the file system concerning the contents of files. The convention of using the first byte of the message to indicate message type may (or may not) be adopted by the communicating tasks themselves.

When a task SENDS a message to another task it will block until the receiving task RECEIVES the message and then REPLIES. When a task does a RECEIVE waiting for a message it will block until one is SENT to it. The operation of REPLY does not block. This presents two scenarios depending upon whether the send occurs before or after the receive.

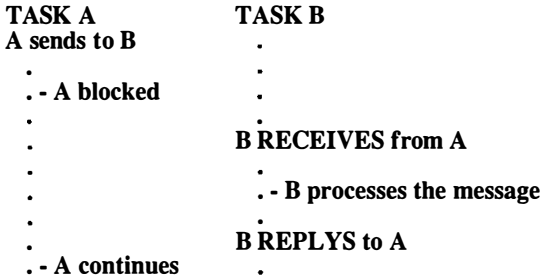


Diagram 10.1

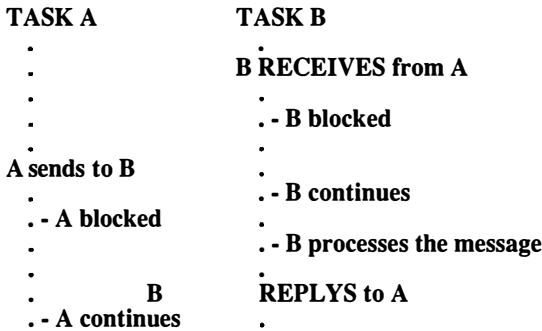


Diagram 10.2

A physical transfer occurs at the point of the receive in Diagram 10.1 and at the point of the send in Diagram 10.2. This can be thought of as the synchronized connection point between the two tasks. Messages implement a totally synchronized communication.

During the REPLY operation the receiver may also reply with 0 to 65,535 bytes of information back to the sender. The transfer occurs immediately and does not block since the sender is already blocked awaiting a reply.

There are two types of receives, general and specific. A general receive may be satisfied by any sending task while a specific receive will only be satisfied by the one task indicated in the receive. All operating systems tasks perform general

receives since they do not know beforehand the identity of the user tasks sending to them. Sending to a task which does not exist or dies before replying will unblock the sending task. Likewise, performing a specific receive on a task which does not exist or dies before sending will unblock the receiver.

Should more than one task send to the same receiving task, they will each block and queue up on the receiving task. As the task performs RECEIVE operations it will receive the messages in the time order that they were sent except in the case of a specific receive. In this case it will receive from that task only, regardless of its position in the queue.

Should two tasks send to each other at the same time, they will both block on each other resulting in a deadlock situation. This can be easily avoided by careful design and by following these guide lines.

1. **If possible, design your programs such that one task is a producer (sender) and the other a consumer (receiver).**
2. **If two tasks MUST send to each other, insist on a very rigid protocol stating when each may send and the other receive. A much better solution would be to create a third task which only performs receives, and have the other tasks send to it. The middle task would then reply appropriately to each task. This middle task is often referred to as an *agent*. The QUEUE manager is an example of a generalized agent task.**

The queuing of sending tasks provides a mechanism for controlled, sequential access to a resource. Complex issues of synchronization, critical sections and semaphores are handled in a clear and uniform manner. For example, the classic problem of providing secure, controlled access to a database from multiple tasks (users) may be greatly simplified by placing a task between the files making up the database and the requesting tasks. The database administrator would now receive synchronized sequential requests to act upon the database. It would provide the intelligent interface to the file system. The file system need not contain complex record locking facilities. These are provided by the administrator task in a manner which is in harmony with the structure of the data. The task can also resolve questions of access permissions, access statistics, access billing etc., a job which should not be performed by the file system or operating system.

The following example is perhaps the most common occurrence in the QNX operating system.

1. User task A sends a message to the device administrator requesting a line of input. At this point task A is SEND blocked and will not compete for the processor.
2. The device administrator receives the message from task A and starts to run. Just before it received the message it would probably have been RECEIVE blocked awaiting a message.
3. Task A goes from the SEND blocked state to a REPLY blocked state. It's message has been received but it is still blocked awaiting a reply.
4. The device administrator checks to see if there is a line waiting from the terminal indicated in the message. Lets assume that there is not. The device administrator will then remember task A's request and again RECEIVE block waiting for another message.
5. Sometime later, a user types carriage return on the terminal Task A has requested a line from. The driver associated with that device sends a signal to the device administrator saying a line is available.
6. The device administrator unblocks, and replies to task A with a message containing the line. It then RECEIVE blocks waiting for another message.
7. Task A unblocks and starts running with a message containing the line.

10.3.1a Death of a Task

When a task dies (for any reason) it sends a message to each QNX administrator informing it of it's death. This allows the administrator to clean up any resources allocated by the task before its death. Users who write their own system tasks may request that they also receive this message when tasks die.

10.3.1b Messages Across the Network

The QNX network extends the range of the message primitives across a local area network. It is this capability which truly separates QNX from other operating systems which claim local area network support. QNX integrates the local area network right into the heart of the operating system, at the fundamental level of inter-task communication.

In the above example, TASK A sent a message to the device administrator to get a line of input from a device. There is no need for the two tasks to be on the same node. This means that a task on one node may communicate directly with any device on any node in the system. It just has to send to the appropriate device administrator task. This also applies to the file system administrator and the task administrator.

To open a file, a user task sends an open message followed by read and write messages to the file system task. The user task may therefore access any floppy, hard or ramdisk on any node by sending messages to the file system task on any node.

To create a new task, a user task sends a create message to the task administrator. By sending this message to the task administrator on another node you may execute remote task creations across the network.

10.3.1c Virtual Circuits

In these examples, the receiving task did not have to perform any special action to receive or reply to messages from remote tasks. It is the responsibility of the sending task to set up the communication path to the receiver. This is accomplished by a QNX primitive which establishes a VIRTUAL CIRCUIT between the sending and receiving task. This primitive

```
vid = vc_create(node_id, task_id, message_size)
```

returns the task-id of a *virtual* task which represents the task on the far node. There is a virtual task created at each node. You may now refer to the vid as though it were a local task, sending to it, receiving from it, etc... The operating system will ensure that all requests will be transparently sent over the network.

The virtual circuit provides the necessary mapping between two tasks. It also ensures that resources can be cleanly recovered across the network when a task dies. For example, assume that TASK A is a very nosy task and opened files on a large number of different nodes and being an irresponsible task, killed itself without closing any of its files. Upon the task's death, the local task administrator would examine the virtual circuits set up by TASK A and send control messages to the destination nodes killing the virtual tasks at the other end. The death of these virtual tasks will be perceived as the death of TASK A to the remote tasks. They will then clean up in the same manner as the death of a local task, closing off TASK A's files. Without the virtual circuit, the death of any task would have to be sent to every task on every node in the network just to be safe! This action would bring a local area network to its knees.

If the poller is running and a node crashes, all nodes will be informed. They will check if any of their tasks have a virtual circuit setup to the crashed node. If they do, the local virtual task is killed and again the local tasks will perceive the death of the remote task on the crashed node.

10.3.2 Ports

Ports provide the important capabilities of

- interrupt handler communications
- simple non-blocking communication

as well as the less important capabilities (due to names and messages) of

- identification of unrelated tasks on the same node
- semaphores

10.3.2a Identification - single node only

Knowing the identity of another task gives a task the ability to communicate with it. The task identities of the system tasks are fixed and thus known to all tasks. When a task creates a new task (son), both the father and the son are informed of each others task identities. In each of these cases the knowledge of identity enables communication by messages. The above message primitives do not satisfy the requirements of communication between unrelated tasks.

A *port* is a globally known name to which a task may attach in order to communicate with unknown tasks. QNX provides a minimum of 32 ports which are identified by number. The first 16 are reserved by the operating system of which the first eight are privileged. A task may attach, detach or obtain the identity of a task connected to a port. The following table summarizes the the action of the attach and detach primitives on a port. For each primitive the port may be free or already attached by another task.

	PORT IS FREE	PORT ALREADY ATTACHED
ATTACH	return zero	return task identity of attached task
DETACH	return zero	return task identity of attached task

Attaching to a free port claims the port, while detaching from an owned port frees it. Detaching from a port you do not own will return a value as above but it will not detach another task. It can therefore be used to obtain the identity of a task connected to a port.

In the database example above, the database administrator would immediately attach to an agreed upon port. All tasks wishing to communicate with it would detach that port to obtain the task identity to which they must send.

The use of ports for name serving is only suitable on a single machine. Under the network version of QNX the section on NAMES provides a better mechanism for identification which will work on a single machine as well as a network.

The major purpose of ports is to allow interrupt handlers to communicate with tasks. The interrupt handler signals a port which a task is attached to. The signal will wake the task up allowing it to process the interrupt.

10.3.2b Semaphores

The operations of ATTACH and DETACH implement a semaphore which may be used to gain controlled access to a resource. QNX's LIST command currently uses port 16 to gain access to the line printer. In a multi-tasking and multi-user system it is quite possible for several instances of the LIST command to be invoked concurrently. If they each wrote indiscriminately to the printer, you would end up with an intermixing of output. Instead, the LIST command attempts to attach to port 16 before opening the line printer for output. If successful, the LIST command continues and starts outputting to the printer. If unsuccessful, then the printer is already in use by another task. In this case the LIST command sends a message to the LIST task which owns the port. Since LIST never does a receive, the sending LIST will block. When the attached LIST finishes and dies, it is detached from the port and all tasks send-blocked on it are unblocked. The sending LIST(s) will then again attempt to attach to the port. The code in LIST to implement this consists of 2 lines of C code.


These semaphores will only work on a single machine and not across a local area network.

10.3.2c Signals

A task may SIGNAL a port resulting in that port sending a message to the task attached to it. This form of communication is special in that

- 1. The task performing the signal does not block.**
- 2. The message sent by the port contains no data.**
- 3. The signal may be performed by an interrupt handler.**
- 4. Messages sent by ports are queued ahead of regular messages and will be received first even if sent second.**

The first point permits a form of non-blocking communication. The task associated with the port will receive the message when it does a RECEIVE operation from anyone or an AWAIT operation on a specific port. The task identity returned by the receive will be that of the port and is clearly distinguishable from any task identity. If a task sends multiple signals before they are received, they will all be remembered. The conditional signal (CSIGNAL) will not signal a port if there is already a pending signal.




The identity of the port which sent the signal is the *only* information which the attached task receives. It is not even aware of the task which performed the signal. This is much more limiting than the SEND-primitive but adequately serves the purpose of informing the task that a particular event has occurred. For example, if a task was waiting upon the event of a game paddle switch being closed it could attach itself to a port and wait for a signal from that port. The signal would indicate the event. No extra data is necessary.

Point three allows interrupt handlers to inform tasks of external hardware events. The game paddle example above would signal the event within the interrupt handler causing the task responsible for that piece of hardware to unblock and service it. The device administrator and file system administrator are both informed by signals when a line is available or a disk operation has completed.

The final point simply gives communication by signals priority over that by messages. When used in conjunction with signals from interrupt handlers, this gives hardware events priority over software requests from other tasks.

10.3.3 Exceptions



So far, all communication between tasks has been synchronous with the receiver. The receiver explicitly had to perform a receive operation and thus was *expecting* a communication. Exceptions on the other hand are asynchronous with the receiving task. They are usually generated by some abnormal event and may occur at any time during the task execution. The best example is the exception associated with a keyboard BREAK. When you run a program, you may type a keyboard BREAK (Ctrl Scroll Lock on PC keyboard) at *any* time. This has the effect of setting an exception on the task associated with the keyboard. There is **NO** way for the task to know if or when the exception will occur. If the task has not protected itself against this exception, then the default action is to kill the task.

There are a total of 32 exceptions which you may set on the task. These correspond to bit positions within two 16 bit words. The operating system reserves the first 16 exceptions. User tasks may freely choose meanings for the second 16 exceptions. Those currently defined by the operating system are

<u>EXCEPTION</u>	<u>HEXADECIMAL VALUE</u>
modem hangup	0001
keyboard break	0002
quit	0004
communication error	0010
missing shared library	0040
floating point error	0080
kill	0100
memory violation	0400
privileged	0800
alarm clock	2000
task termination	4000
divide by zero	8000

The 'kill' and 'task term' exceptions may not be protected against and always results in the task being killed. The 'task termination' exception is the means by which a task terminates itself (normally or via an explicitly coded abort).

Certain tasks, like the editor, do not want to be destroyed by exceptions like BREAK. Therefore, the operating system provides the ability for a task to either ignore exceptions or execute a user provided routine to handle them. In the case of the editor, it terminates the current operation and returns to command level. Note that due to the asynchronous nature of an exception it is necessary for the editor to protect itself by ignoring exceptions during certain critical operations. In these cases the exception is left pending until the editor re-allows it at which point the exception occurs. What the editor has done is to force synchronization of the exception during these critical places.

The capabilities of user tasks handling exceptions is best described using diagram 10.3. For simplicity we will assume a single exception being set (one bit set in the exception word).

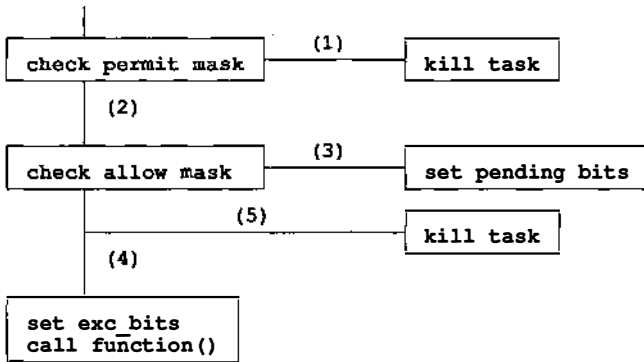


Diagram 10.3

The exception fields of PERMIT, ALLOW, PENDING and FUNCTION are all maintained in the user task's address space and may be manipulated directly. From within a C program they are referenced as

```

extern unsigned Exc_permit[2], Exc_allow[2];
extern unsigned Exc_pending[2], Exc_bits[2];

```

The PERMIT field may be thought of as a bit mask against which an exception is applied. If an exception bit is set where a permit bit is not, then the task is not permitted to handle that exception and is killed (1). Once the exception makes it past this first mask, it is applied to the second mask called ALLOW (2). If this bit is clear then the task is not allowing this exception at this time and the exception bit is placed in the PENDING field (3). No other action occurs. The task may examine the pending field at any time in order to see if any permitted but disallowed exceptions have occurred. This allows the task to poll the bits at its convenience. If all bits are disallowed then the bits can be treated as a 16 bit number which can be set by another task. Finally, if the bit in allow had been set, then the function whose address is in the FUNCTION() field would be invoked (4). In this case if there is not a function set up to handle the exception, the task is killed (5).

It is possible for multiple exceptions to be set upon a task by setting more than one bit in the indicated exception. The above explanation operates on all 32 bits in parallel. Killing the task (no permit) followed by invoking the exception function (an allow) take precedence when considering multiple exceptions. The operating system tasks do not set multiple exceptions on a task.

The following table summarizes the actions for a single exception, permit and allow, bit path.

PERMIT	ALLOW	ACTION
clear	clear	kill task
clear	set	kill task
set	clear	set proper pending bit
set	set	invoke exception function

If a task sets an exception on a virtual task (vc_create), the operating system will forward the request to the node which contains the real task. The real task will have the exception placed upon it.

A word of warning... If an allowed exception occurs, the task will become UNBLOCKED before executing the exception function. This has the unfortunate side effect of causing errors if the program was waiting for input from the keyboard for example. User programs which handle exceptions should be prepared to handle errors at any point in their program. Please refer to the documentation on the C library routine EXC_HANDLER.

10.4 Global Names

As stated earlier, for a task to communicate, it needs to know the task id of the task it wishes to communicate with. QNX offers a mechanism where a task may attach a symbolic name of up to 8 characters which may be queried by other tasks. Associated with the name will be the node number and task id of the attaching task. A task may attach its name as local to a single machine or global across the entire network.

The first type of attaching is handled by a table in the local task administrator. The second type is handled with the assistance of a server task which must be running on a node in the network. This is described in the technical note "Attaching and Locating Task Names" in the C binder.

Attaching a name is similar to the attach and detach primitives on ports.

10.5 Task States

The previous sections have talked about tasks blocking. In QNX a task is always in one of six states.

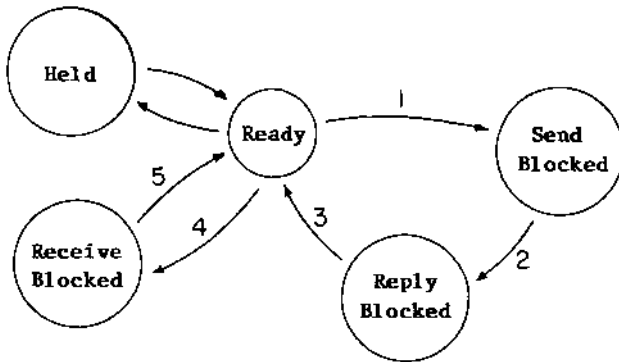
DEAD	- Task does not exist or is being killed
READY	- Task is ready to run
SEND_BLOCKED	- Task has done a SEND which

- RECEIVE_BLOCKED** - has not been received
- REPLY_BLOCKED** - Task has done a RECEIVE with no waiting send
- HELD** - Task has done a SEND which has been received but not replied to
- NETWORK_BLOCKED** - Task is ready but has been explicitly held
- Task is blocked on a network request

The three blocked states are all tied to the message communication primitives. The state of HELD indicates that the task is ready to run (not blocked) but is being prevented from running due to a HOLD operation on it. Performing a hold on a blocked task does not change its state. However, as soon as the task unblocks, its state will change to HELD, not READY.

The highest priority task that is ready will be the task which is running. If there is more than one task ready at the highest priority level, then they will be time-sliced every fraction of a second.

The possible state changes are indicated graphically in diagram 10.4. The arrows indicate a state transition caused by the indicated reason.



- 1. Task sends message
- 2. Target task receives message
- 3. Target task replies
- 4. Task waits for message
- 5. Message received

Diagram 10.4

10.6 Task Ids

A task id consists of a 16 bit word in which the lower 8 bits is a task number and the upper seven bits is a version number. Each time a task is created, it is assigned the task number of a dead task and the version number of that task is incremented by one. The version number ensures that a particular task *number* may be reused 128 times before reusing a given task *name*. The telephone company follows a similar policy in not immediately re-using canceled numbers. By holding off on reassignment, there is less possibility of someone phoning the old number and getting an answer. Instead they get a number out of service. Similarly, in QNX, a send to an old task will usually fail. The top bit indicates whether the task is local or remote (virtual task).

The task number must lie in the range of 1 to 254. The maximum number of tasks allowed and other system information may be obtained by using the TSK command.

tsk info

The return values of system calls which return task names is summarized in the following table.

<u>HEXADECIMAL TASK NAME</u>	<u>MEANING</u>
0000	Task does not exist
0100 to ff00	Valid port name
xx01 to xxfe	Valid task name
fff	System call failed

where **xx** is any combination (00 to ff) and
the top bit indicates if the task is
0 - local (real) task
1 - remote (virtual) task

Examples

0500 - port five
0508 - local task (number 8, 5th incarnation)
8728 - virtual task (number 28, 7th incarnation)

10.7 Task Hierarchy

Tasks running in the system have a hierarchical (tree) structure which very closely parallels that of the file system. Each task has one father and zero or more sons and brothers. Two examples are shown in diagram 10.5.

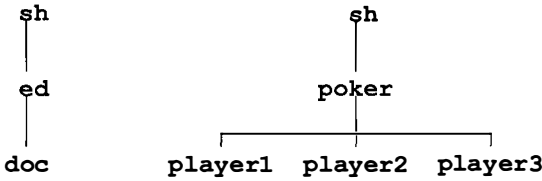


Diagram 10.5

When a task dies, all its children are also killed. This may seem harsh, but it is necessary to maintain a consistent task hierarchy. Imagine removing a directory without first removing the files under it!

10.8 Task Creation

When a task creates a son task, there are three major options available to it.

1. **It may wait upon the sons death before continuing to execute itself.**
2. **It may continue executing in parallel with the son (concurrent).**
3. **It may continue to execute in parallel with the new task but not preserve the father son relationship (background). As a result the death of the father task will NOT kill the new son task.**

The command interpreter (shell) by default always chooses option 1 unless you explicitly tell it to run the task background (option 3). Option 2 is not a common occurrence. One might imagine a large application program which consists of one father task which creates several son tasks, all performing a separate function, and running concurrently. Killing the father will remove (kill) all sons.

In addition the father task may also set the priority of the son at creation.

10.9 Terminal Ownership

At any one time, only one task is associated with a terminal. This will be the most recent task to open the terminal for read. Besides receiving all input the task will also catch keyboard BREAKS. Each open stacks on top of the last open. Only the last open on the top of the stack is active. Closing a terminal will remove it from the stack. This may move a previous open to the top of the stack making it active

again. If a task which is not on top of the stack attempts to read from a terminal, the request will be queued until that task becomes active. A remote task may take active control of the terminal by opening it in the same manner as a local task. This scheme has no ambiguity.

When a task is created it may optionally be "handed" down responsibility for handling breaks on the device. It is possible for a son task to process data from a terminal, but have breaks still be set on the father task.

11. QUICS - The Quantum Update System

The Quantum update system, called Quics, is a computer which runs 24 hours/day which you may call (using a modem) to:

- **Download fixes and changes to the operating system, utilities and libraries.**
- **Download a variety of free software from various sources.**
- **Send us electronic mail.**
- **Engage in a conference with other QNX users on a large variety of topics.**

The first point allows us to provide you with a level of customer service which is second to none in this industry. 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 on line, the most current versions of all programs including the operating system itself for download. Along with these we provide a list of all changes made to the system. You may wish to sign on once a month and make a log of this list.

We also maintain a growing list of free software which may be of interest to you. It includes games, nifty utilities and a megabytes of source provided by ourselves and our customers. This source can be an invaluable aid in helping you to understand some of the intricacies of QNX. For example, the QUEUE manager shipped in binary form with your system is available in full source under the free software. The same is true for all disk and graphics drivers.

If you are not in a hurry for an answer you can send us electronic mail which we will answer within a day or two.

Lastly, if you would like to comment or ask a question which you feel other users may be interested in, then you will want to use our conference system. It has a command structure which is very similar to that used by Byte magazine's conferencing system. All the Quantum developers are members of the conferencing system as well as most of our more active developers. It is not unusual to pose a question and have several comments on it within an hour or so. There are topics on real time, hardware, utilities, c compiler, ... and so on. You do not need to be a QNX customer to use the conferencing system. It can be accessed from a terminal or a terminal emulation program running under DOS.

The machine you are calling is node 1 of a large QNX network. It is our boot node and the node from which most of our users get all their commands. While signed on, you are in fact sharing the machine with quite a few users. At the time of this printing the boot node is an 8 Mhz AT with 3 modem lines and an X.25 line into the Datapac public network which supports 4 (expandable to 32) additional users.

All three modems will support both 300 and 1200 baud service. We are always upgrading our equipment and some (if not all) of the modems will support 2400 baud and we are looking to provide 9600 baud service as well. The 3 modems are on a hunt group with the primary number being.

(613) 591-0934 • First number of the hunt group.

An up to date configuration on the number of modems, their types and phone numbers is maintained on the update system. You can read this when you give us a call.

As stated above, we are also connected via an X.25 link to the Datapac public network. An X.25 card plugs into our AT and interfaces with X.25 software which runs under QNX. The X.25 link can support several user calls at one time. To connect with us via X.25 you will have to make a pre-paid call or register with us for an X.25 account number allowing you to make collect calls. Our network address is

3020 85701416
└──┬── Our datapac address.
└──┬── Datapac's network code.

11.1 How To Phone Us

To call us you will have to use one of QNX's terminal emulation programs.

- qtalk • A simple terminal program which comes with the system.**
- qterm • A very slick, full featured terminal program which is separately purchased.**

Since QTALK is shipped with every system we will describe how to contact us using it. We will assume that your modem is connected to the QNX device \$mdm. If it is not, you will have to specify to QTALK which tty device it is on.

Step 1 Set your baud rate to conform to your modem as follows:

```
stty baud=2400 par=none bits=8 >$mdm
stty baud=1200 par=none bits=8 >$mdm ( QNX default at boot )
stty baud=300 par=none bits=8 >$mdm
```


Step 2 Start QTALK and cause your modem to dial us. If you are in north america and have a hayes compatible modem you can use QTALKS's dialing directory capability. We provide an entry with our phone number and the name "quics". Refer to the documentation on QTALK in the utilities section of the operating system manual for more information.

`qtalk quics [m=modem_device]` • Hayes compatible modem.
`qtalk [m=modem_device]` • Other modem.

The option `m=modem_device` is only necessary if your modem is not connected to the device `$mdm`.

Step 3 When the word **CONNECT** is printed, wait one second. If you do not get a login message type the following 5 characters

`ab..carriage-return`
 └ this is the <enter> key

and repeat if necessary. You should get a screen of information telling you what to do next. One of your options will allow you to download an up-to-date manual on how to use Quics, the update system.

11.2 X25 Access

Quics is also connected to the Datapac packet network via an X.25 link. Customers may access the system by using a public dial-up port. Instead of placing a long distance phone call directly to one of Quantum's modem lines, you place a local call to a modem line on a packet network. Once connected to the packet network you will have to place a call to Quantum's X.25 link. This is accomplished by entering a multi-digit number (referred to as a DNA, destination network address) to identify who you wish to call. Once a successful call is made, you will be connected to Quantum's update computer and all data will be routed through the packet network. Each country tends to have its own packet network with gateways connecting the many networks. Due to agreed upon standards, you should be able to access our update computer from nearly any country/network in the world. The standards, unfortunately, do not apply to the user interface presented to the user when he wishes to place his call. At this point, you are communicating with the network (*this is similar to the method by which you communicate with a hayes modem when you wish to place a call*) and each network has its own command language. Once the call has been established you are then communicating with QNX and the network should pass all data transparently. There are two major advantages of accessing the update system via X.25.

First, the networks communicate at very high speed and pass data to each other in a completely error-free manner. The only place a communication error is possible is between your modem and the network's modem over the phone line on which you called. This will typically be a local phone call. Those of you who have tried to access the update computer directly using long distance may have discovered that the lines are often very noisy. It seems to depend on the time of day, phase of the moon and how badly you need the data. Our European customers have found the long distance line pretty much unusable. Our X.25 link has changed all this and for the first time we have a global update system.

Second, access via the packet networks is often cheaper than the cost of a long distance phone call. This is especially true when you consider the re-transmission time caused by data errors. The cost is based upon both connection time and the amount of data transferred. Each network has it's own rate schedule with some networks being more expensive than others. The different networks have agreed upon inter-network costs in much the same fashion as the international phone companies.

You still have to use a QTALK or QTERM to contact us as described above, however, you will call your local public network and then use it's commands to place a call to us.

Contact Quantum's marketing department for an application for an X.25 account which will allow you to place collect calls.

12. Tips on Using QNX

This section consists of a few tips in getting started on QNX.

12.1 Memory Requirements

To boot QNX you need a minimum of 256K of memory. This can be expanded to 640K in real mode and 15 Megabytes in protected mode.

12.2 Enabling Colour

You may change the colour of your text and screen by issuing escape sequences to your screen. As an example, the sequence

Esc@71Esc!71 (Esc is the single Esc key)

will set your character and fill attribute to white on blue.

COLOUR CODES

<u>digit</u>	<u>colour</u>	<u>digit</u>	<u>colour</u>
0	black	4	red
1	blue	5	magenta
2	green	6	brown
3	cyan	7	white

You should set your colour preference using ESC @ and ESC ! in your `sys.init` file and follow it with an ESC S to save it away. All QNX applications which play with the colour will print an ESC R to restore the saved colour before exiting.

type **Esc@71Esc!71EscS** (Esc is the single Esc key)

12.3 Disabling Colour

If you have a colour card, but a black and white monitor, you may wish to suppress all colour since it makes text very hard to read. You can disable colour using the `stty +nocolour` command.

12.4 The Mount Command

The MOUNT command allows you to dynamically reconfigure QNX. You may

1. Install custom disk drives.
2. Install consoles.

3. Install shared libraries.
4. Install disk caches.

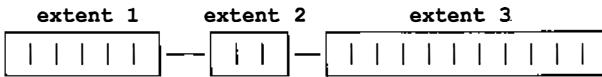
By default QNX assumes a 360K (40 track) floppy disk drive for drive one and the same for drive two if it exists. On machines like the AT, QNX will automatically configure itself to support the 1.2M high capacity drives. On a PC, users may have an 80 track floppy for drive 2. To configure your 80 track drive, you would issue the command

```
mount disk 2 t=80 n=9 h=2
```

If you can spare the memory, a great improvement in hard disk access time can be made by installing a disk cache. For example, we could add a cache to disk 3 by issuing:

```
mount cache d=3 s=32k
```

If you are going to accessing large files in a random access mode (data bases), then a tremendous improvement in performance can be had by mounting an XCACHE. A file in QNX consists of one or more extents. Each extent is a contiguous series of blocks within the file.



An XCACHE caches the information on how these extents are connected. This saves you from having to chain through extents when seeking within a file. Each extent connection requires 20 bytes. This cache is shared by all disks.

```
mount xcache s=10k - enough for 512 extents
```

Finally, you can speed up disk writes by mounting a bitmap cache. This cache takes up very little room and yet it can add significant performance when growing files.

```
mount bmcache d=3
```

Virtual consoles an extremely useful extension to QNX and can be added to the system (at the expense of another device entry which is hopefully not required) by a command such as:

mount console con1

Shared libraries such as graphics libraries or floating point math libraries can be installed with commands such as:

```
mount lib /drivers/glib.ega  
mount lib /config/qdb.slib
```

The use of shared libraries allows application programs which run under QNX to be independent of hardware.

12.5 Ramdisk

One form of the mount command allows you to create a virtual disk in memory. This disk behaves exactly like a real disk in all respects except speed where it is much faster. To create a 200K ramdisk as drive five issue the command

```
mount ramdisk 5 s=200k
```

The ramdisk loads a driver into memory. As a result, the above command would require just a little bit more than 200K of memory to create the ram disk. You can only mount a **single** ramdisk.

You must now initialize the disk using the dinit command as follows

```
dinit 5
```

QNX will not look on the ramdisk until you include it in its search order using the search command

```
search 5 3
```

In this example we have indicated that we wish to always look on the ramdisk first, followed by drive 3. We will not look on drive 1 unless that drive number is explicitly prefixed to the file name (1:/).

Once the ramdisk has been initialized you may create directories and files on it. Creating the directory **/tmp** on it gives you a place to put all temporary files you wish to go away when you power off. The compiler places its temporary files under **/tmp** as well.

```
mkdir 5:/tmp  
chattr 5:/tmp p=+c
```

Don't forget to add general create permission on the /tmp directory if other users are going to be using temporary files. Ramdisks may be reinitialized by using DINIT and starting again! If you have sufficient memory, you may wish to create a larger ramdisk and place a /cmds directory on it. Copy down the editor and its macro file as well as any other commonly used commands.



```
cp 3:/cmds/ed 5:/cmds/ed
cp 3:/cmds/ed.macros 5:/cmds/ed.macros
... any other commonly used commands ...
```

Ramdisk is also convenient for data logging over a high speed communication line. The pause going to a disk is eliminated.

Once they get used to the speed, most users find it hard to work without a ramdisk! A ramdisk can be created from your sys.init.*nn* file.

12.6 Shared Libraries

Rather than duplicate code in many applications, or increase the size of the operating system, QNX has adopted the technique of dynamically mounting software libraries of code which can be *shared*. This technique can also be used provide machine independent interfaces to hardware. For example, a different Graphics library may be mounted for each type of graphics card. The user's application program does not change. It makes calls into the shared library to accomplish a function. The standard shared libraries are.



/config/float.slib	- software floating point
/config/float8087.slib	- 8087 floating point
/drivers/glib.ega	- Console/Graphics for ega card

12.7 Operating On Groups Of Files

QNX provides two very useful commands which allows you to execute a command on a group of files. They are the EO (execute on) and WS (walk structure) commands. These routines may seem confusing to the novice, however, they will quickly become indispensable as you become familiar with the system. Please keep them in mind and refer to the documentation in the utilities manual.



Appendix A

Character Set and Keyboard Codes

DECIMAL VALUE	HEXA DECIMAL VALUE	0	16	32	48	64	80	96	112
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	[k	{
12	C	♀	└	,	<	L	\	l	
13	D	♪	↔	-	=	M]	m	}
14	E	♪	▲	.	>	N	^	n	~
15	F	☀	▼	/	?	O	_	o	△

Standard Console Character Set

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	à	⌌	⌌	μ	÷	
7	7	ç	ù	ó	⌌	⌌	τ	≈	
8	8	ê	ÿ	ì	⌌	⌌	ϕ	°	
9	9	ë	Ö	⌌	⌌	⌌	θ	•	
10	A	è	Ü	⌌	⌌	⌌	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'	

Standard Console Character Set

ASCII Code to ASCII Character Table

Upper 4 bits

	0	1	2	3	4	5	6	7
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	DEL

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 z
ETX - Ctrl c	VT - Ctrl k	DC3 - Ctrl s	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 _

QNX Keyboard Codes

Upper 4 bits

	8	9	A	B	C	D	E	F
0	Shift TAB	Shift F6	Home	Ctrl Home	Alt Home			Alt P
1	F1	Shift F7	up	Ctrl up	Alt up	Alt F1	Alt a	Alt q
2	F2	Shift F8	PgUp	Ctrl PgUp	Alt PgUp	Alt F2	Alt b	Alt r
3	F3	Shift F9	minus	Ctrl minus	Alt minus	Alt F3	Alt c	Alt s
4	F4	Shift F10	left	Ctrl left	Alt left	Alt F4	Alt d	Alt t
5	F5	Ctrl F1	five	Ctrl five	Alt five	Alt F5	Alt e	Alt u
6	F6	Ctrl F2	right	Ctrl right	Alt right	Alt F6	Alt f	Alt v
7	F7	Ctrl F3	plus	Ctrl plus	Alt plus	Alt F7	Alt g	Alt w
8	F8	Ctrl F4	End	Ctrl End	Alt End	Alt F8	Alt h	Alt x
9	F9	Ctrl F5	down	Ctrl down	Alt down	Alt F9	Alt i	Alt y
A	F10	Ctrl F6	PgDn	Ctrl PgDn	Alt PgDn	Alt F10	Alt j	Alt z
B	Shift F1	Ctrl F7	Ins	Ctrl Ins	Alt Ins	Shift F11	Alt k	
C	Shift F2	Ctrl F8	Del	Ctrl Del	Alt Del	Shift F12	Alt l	
D	Shift F3	Ctrl F9	PrtSc	Ctrl PrtSc	Alt PrtSc	SysRq	Alt m	
E	Shift F4	Ctrl F10	F11	Ctrl F11	Alt F11		Alt n	
F	Shift F5	Ctrl TAB	F12	Ctrl F12	Alt F12		Alt o	

Note:

The above codes are preceded by a hex FF code, if the option EXPAND_FUNC is turned on, and the indicated function keys are pressed.

The above codes can also be generated as a *compose* character, which results when the ALT key is pressed and released, then 2 more characters are typed. *compose* characters are NEVER preceded with an FF code.

Alt C , Ç	Alt E ' É	Alt a ' á	Alt g a ∞
Alt u " ü	Alt A e 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 _ a	Alt g u μ
Alt c , ç	Alt u ` ù	Alt o _ o	Alt g t τ
Alt e ^ è	Alt y " ÿ	Alt ? ? ÿ	Alt o ø
Alt e " ë	Alt O " Ö	Alt + _ ⌈	Alt o - θ
Alt e ` è	Alt U " Ü	Alt _ + ⌋	Alt g w Ω
Alt i " ï	Alt c ç	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 t R	Alt < < «	Alt E E E
Alt A o Å	Alt f - f	Alt > > »	Alt U U U

Alt = = ≡
Alt + - ±
Alt > = ≧
Alt < = ≦
Alt - : ÷
Alt ~ ~ ≈
Alt o . •
Alt . . •
Alt s q √
Alt g n n
Alt ^ 2 2

Release Alt before typing 2 character
input sequences

This page left blank

This page left blank

- AT disk 10
- Alive 33
- Argv 81
- Attach 101
- Attributes 69
- BIOS disk 11
- Background 83, 109
- Backslash 81
- Backup 17
- Bad blocks 16, 17
- Baud rate 38
- Bios driver 19
- Bitmap 62
- Blocking 93
- Bmcache 21
- Booting
 - Sys.init 20
 - floppy 3
 - hard disk 18, 20
 - network 31
- COM1 37
- Cache 21, 116
- Carrier detect 43
- Cd 67
- Change directory 89
- Characters, foreign 45, 48
- Chattr 73
- Colour 115
- Comm 41, 43
- Command files 86
- Command interpretor 79
- Communication 95
- Compose characters 45, 48
- Concurrent 109
- Conference system 111
- Configuration, os 27
- Consoles
 - colour 115
 - graphics 58
 - memory 55
 - mounting extra ones 55
- Ctrl-z 48
- Current directory 67, 77, 89
- DCE/DTE 39
- Datapac 112
- Date 3
- Dcheck 17
- Death 99, 100, 109
- Detach 101
- Device
 - editing 41
 - escape sequences 50
 - input 45
 - input gate 49
 - line editing 46
 - line recall 49
 - output 50
 - smartcards 44
 - type-ahead 46
- Device admin 94
- Device names 73, 74, 78
- Devices 45
- Dinit 16
- Directories
 - moving up 68
- Directory 62
 - changing 67
 - current 67
 - number files 62
 - printing 68
 - structure 62
- Disks 7
 - at 10
 - bios 11
 - booting 3, 18, 20
 - booting from 18
 - cache 21
 - controller 7
 - drivers 7
 - formatting 15
 - hdisk.cfg 19
 - installation 12
 - mounting 8, 14
 - nighthawk 11
 - partition 12, 14
 - ps/2 11
 - second physical 27
 - xt 9
- Dmark 16
- Dos partition 13, 26

- Download 111
- Drive numbers 66
- EGA
 - smart cards crashing 57
 - switching consoles 57
- Ega 117
- Exceptions 95, 103
 - actions 106
 - catching 104, 105
 - list of 103
 - network 106
- Execute permission 88
- Fdformat 15
- Fdisk 14, 20
- File admin 94
- File system
 - attributes 69
 - bitmap 62
 - commands 65
 - current directory 67, 77
 - device names 73
 - directories 62
 - drive numbers 66
 - file names 61
 - max file size 61
 - network 75
 - network access 78
 - node numbers 75
 - pathnames 65
 - permissions 69, 73
 - ramdisk 117
 - remdisks 77
 - search order 66, 75
 - structure 61
 - text files 41, 61
 - user numbers 73
- Flashing text 51
- Flow control 41
- Foreign characters 45, 48
- Free software 111
- Graphics
 - Libraries 58
 - consoles 58
- Group 73
- Group permissions 69

- Hard disks
 - see Disks* 7
- Hdisk.cfg 19
- Hierarchical 62
- Highlighted text 51
- Input 45
- Installation
 - command 5
 - manual 7
 - network 29
 - summary 22, 32
- Inter-task communication 95
- Interrupt handlers 103
- Inverse text 51
- Keyboard
 - input gate 49
 - line editing 46
 - line recall 49
 - shift keys 45
 - type-ahead 46
- Kill_vcs 33
- Line editing 46
- Line recalling 49
- Locker admin 95
- Login 3, 42
 - modem 42
- Macros 86
- Member 73
- Memory
 - Requirements 115
 - Used by consoles 55
- Message deadlock 98
- Message example 98
- Message queuing 98
- Messages 95, 96, 99, 100
- Mkdir 73
- Modem 42
- Modems 37, 41
- Mount 115
 - consoles 55
- Multi-tasking 93
- Multi-user 93
- Nacc 30, 78
- Names 106
- Netboot 31

- Netboot directory 18
- Network 94
 - access 30, 78
 - booting 31
 - diagnostic tips 35
 - exceptions 106
 - hardware 29
 - installation 29
 - installation summary 32
 - messages 99
 - poller 32, 100
- Network admin 94
- Nighthawk disk 11
- Node numbers 30, 75
- Nodes 85
- Non-blocking 102
- Null modem 39
- Or-bar 84
- Osconfig 27
- Other permissions 69
- PS/2 disk 11
- Parallel 43
- Parity 38
- Partition 5, 12, 14
 - dos 26
 - more than one 26
- Passwords 31, 73
- Pathnames 65
 - absolute 65, 66, 67
 - network 75, 78
 - relative 67
- Patterns 82
- Permissions 69, 73
 - access chart 69
 - default 71, 72
 - examples 72
 - types of 70, 71
- Phone numbers 1, 112
- Pipes 84
- Poller 32, 100
- Port
 - ids 108
- Ports 95, 100
 - identification 101
 - interrupt handlers 101, 103
 - semaphores 102
 - signals 102
- Printer 41
- Printers 37, 43
- Priority 83, 89, 93
- Pwd 68
- QNX loader 18
- QNX partition 12, 14
- QUICS 111
- Qtalk 43, 112
- Qterm 43
- Queue admin 95
- Quics 111
 - phone numbers 112
 - x.25 113
- RS-232C
 - see serial* 37
- Ramdisk 117
- Rebooting 48
- Receive 96
- Record locking 95
- Record separator 61
- Redirection of I/O 80
- Remote Disks 77
- Remote execution 85
- Remote search 75
- Reply 96
- Root 62, 65
- Scheduling 93
- Screen
 - colour 51, 52, 53
 - cursor addressing 51
 - escape sequences 50
- Search 17
 - remote 30
- Search order 65, 66, 76
 - example 76
 - indirect 76
 - remote 75
- Semaphores 102
- Send 96
- Serial 37
 - 4/8 port cards 38
 - baud rate, parity 38
 - cables 39, 40

- com1, com2 38
- flow control 41
- i/o ports 37
- interrupts 37, 38
- login 42
- modem 42
- modems 41
- outgoing calls 43
- printer 41
- problems 43
- ps/2 38
- smartcards 44
- Server 98
- Setvar 87
- Shared libraries 118
- Shell 79
 - Variables 86, 87
 - background tasks 83
 - command files 86, 88
 - commands 89
 - comment lines 85
 - multiple commands 84
 - pipes 84
 - prompt 79
 - quoting 81
 - redirection 80
 - reference chart 90
 - remote execution 85
 - sys.init 80
 - usage message 83
 - variables 86
 - wildcards 82
- Signals 102
- Smartcards 44
- Spooldev admin 94
- Standard I/O 80
- Star 82
- States 106
- Stty 38
- Super user 4, 73, 83
- Sys.cfg 27
- Sys.init 30
 - Standard I/O 80
 - sample contents 20
- System tasks 94
- Sytem initialization file 30
- Task
 - background 83
 - code sharing 89
 - communication 95
 - creation 48, 109
 - death 99, 100, 109
 - exceptions 95, 103, 104
 - hierarchy 108
 - ids 108
 - messages 95, 96, 97, 98, 99
 - names 106
 - ports 95, 100, 102
 - priority 83, 89, 93
 - scheduling 93
 - states 106, 107
 - system 94, 95
 - tty 109
 - virtual circuits 100
- Task admin 94
- Technical support 1
- Terminals 109
- Text files 41, 61
- Time slice 93
- Timer admin 94
- Tips 115
- Tree structure 62
- Underline text 51
- Update system 111
- Usage message 83
- User directory 62, 73
- User numbers 73
- Variables 86, 87
- Vc_create 100
- Virtual circuits 100
- Warm boot 48
- Wildcard 82
- X.25 111, 113
 - address 112
 - phone numbers 111
- XON/XOFF 41
- XT disk 9
- Xcache 21