

QNX[®] 4.2 Operating System

Utilities Reference Supplement

QNX Software Systems Ltd.

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

© QNX Software Systems Ltd. 1993
All rights reserved.

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

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

Printing history:

July 1993	First edition
November 1993	Reprint
April 1994	Reprint
September 1994	Reprint
April 1995	Reprint

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

Cover art by Robert Tinney

Printed in Canada.

About this Supplement

The *Utilities Reference Supplement* accompanies the QNX 4 operating system, release 4.2, and documents the new utilities that come with this release.

In this supplement you'll also find updated documentation for the following existing utilities; these have changed significantly since the latest edition of the *Utilities Reference*:

- **cc/CC**
- **Proc**
- **qtalk**

You'll also find two new utilities that replace existing utilities:

- **Fsys.aha4scsi** replaces **Fsys.aha4**
- **Fsys.ps2scsi** replaces **Fsys.ps2s**

A future edition of the *Utilities Reference* will contain all the utilities described here.

This supplement contains the following utilities:

cc/CC	compile command (UNIX)	1
Csc	client-side cache (QNX)	15
Dev.ansi	console driver (QNX)	17
esh	embedded shell (QNX)	19
Fsys.aha4scsi	driver for Adaptec 4 SCSI host adapter (QNX)	25
Fsys.ide	driver for IDE fixed disk interface (QNX)	33

Fsys.ps2scsi	driver for IBM PS/2 SCSI host adapter (QNX) 35
fsysinfo	display filesystem statistics (QNX) 41
int10	provide some of the BIOS INT 10 functions (QNX) 45
Iso9660fsys	ISO 9660 Filesystem Manager (QNX) 47
Net.arcxir	Xircom Pocket Arcnet Adapter II network driver (QNX) 53
Net.ether503	3com 503 Ethernet network driver (QNX) 57
Net.ether1000	NE1000/2000 Ethernet network driver (QNX) 61
Net.fd	file-descriptor-based network driver (QNX) 65
Net.tr164a	IBM Token Ring 16/4 Adapter network driver (QNX) 69
Proc	Process Manager (QNX) 71
qtalk	talk over communications line (QNX) 77
Slib	system shared library (QNX) 91
tracectrl	display and change trace parameters (QNX) 93
traceinfo	print trace events (QNX) 95
tracelogger	extract trace events and log them to a file (QNX) 113

Syntax: Invoke the C compiler:

```
cc [options] [operands]  
c89 [options] [operands]
```

Invoke the C++ compiler:

```
CC [options] [operands]
```

Options:

- 0 Produce 16-bit object code, using only 8086 instructions. The code will run on the 16- or 32-bit version of QNX.
- 2 Produce 16-bit object code, using 286 instructions. The code will run on the 16- or 32-bit version of QNX if QNX is operating on a processor upwardly compatible with the Intel 286.
- 3 Produce 32-bit object code, using 386 instructions optimized for Intel 386 instruction timings. The code will run only on the 32-bit version of QNX.
- 4 Same as -3, but for 486 instruction timings. This is the default. For maximum performance, you should also use the `x` optimization (see the `-O` option).
- 5 Same as -3, but for Pentium instruction timings. For maximum performance, you should also use the `x` optimization (see the `-O` option).
- A *library*
Combine all object files provided as operands and build this new library. The *library* filename is given the `.lib` suffix.
- a *library*
Combine all object files provided as operands and add them to this existing library. The *library* filename is assumed to have a `.lib` suffix.
- B Search the QNX 2 migration library that corresponds to the specified memory model (see option `-m`). The QNX 2 migration library provides backwards compatibility for many

discontinued functions. The library is searched before the beta and standard libraries.

- b Search the beta library that corresponds to the specified memory model (see option -m).

The beta libraries are searched before the standard libraries. The beta libraries include updates made since the last formal release of the software and can be updated by modem using the QNX update system.

- c Suppress the link phase and retain all created object files.

- D *name=text*

Define *name* as if by a C language *#define* directive. If no *text* is specified, *name* is defined as 1.

- E Copy the language source files to the standard output, expanding all preprocessor directives. No compilation is performed. See also option -P.

- F Force objects into the same directory as source (default is the current working directory).

- f{*option*}

Pass through to the -f option of **wcc**.

- g{1|2}

Produce symbolic information in the object file and in the executable file. This option is necessary to run the symbolic debugger (**wvideo**). If -g1 is specified, the optimizer remains enabled for code generation. If -g2 is specified, the optimizer is disabled, resulting in code that is easier to debug. Specifying -g is the same as specifying -g2.

- I *directory*

Change the default search algorithm for header files whose names don't begin with a slash (/) so that the directory named by the *directory* pathname is searched before the standard path. The standard path is **/usr/include**. Headers are searched in the following order:

- 1) in the directory of the file with the `#include` line
- 2) in the directories named in the `-I` option
- 3) in `/usr/include`

Directories named with the `-I` option are searched in the order specified. If an `#include` is surrounded by angle brackets (`<>`), the first search is *not* done.

`-j` Change `char` default from unsigned to signed.

`-L directory`

Change the default algorithm (that searches for libraries) to include this directory. Directories are searched in the order they appear on the command line and before the default directory, `/usr/lib`. You may change this default by setting the environment variable `LIBQNX` to one or more directories separated by colons.

`-l library`

(the letter “e”) Search this library file for any unresolved references during the link phase. Libraries are searched in the order they appear on the command line, and before the default libraries. If the library name doesn’t start with a slash (/), a search is made in the following order:

- 1) in the current directory
- 2) in the directories named in the `-L` option
- 3) in the default directory

If the library name doesn’t end with `.lib`, the compilation-model suffix is appended.

`-M` Produce a symbolic map file with the same name as the output file, but with a `.map` suffix.

number of operators used in an expression. Functions that require more than the specified *number_of_quads* aren't expanded inline (the default number is 20). This optimization is useful when locally referenced functions are small in size.

Example:

```
cc -Oe dhrystone
cc -Oe=25 whetstone
```

- f Generate traceable stack frames for functions that contain calls or require stack frame setup. For near functions, **cc** generates the following function prologue sequence:

16-bit compiler	32-bit compiler
push BP	push EBP
mov BP, SP	mov EBP, ESP

For far functions, **cc** generates the following function prologue sequence:

16-bit compiler	32-bit compiler
inc BP	inc EBP
push BP	push EBP
mov BP, SP	mov EBP, ESP

The **BP/EBP** value on the stack will be odd or even depending on the code model.

- f+ Generate traceable stack frames for *all* functions, including those that don't contain calls or require stack frame setup. You should find this useful for developing embedded systems (ROM-based applications).

The **f+** flag generates prologue sequences in the same way as the **f** flag.

- i Generate certain library functions in-line. For a function to be generated in-line, you must include the appropriate header file containing the prototype for the function. These are the functions that can be generated in-line:

<i>abs()</i>	<i>_disable()</i>	<i>div()</i>
<i>_enable()</i>	<i>fabs()</i>	<i>fmemchr()</i>
<i>fmemcmp()</i>	<i>fmemcpy()</i>	<i>fmemset()</i>
<i>fstrcat()</i>	<i>fstrcmp()</i>	<i>fstrcpy()</i>
<i>fstrlen()</i>	<i>labs()</i>	<i>ldiv(32)</i>
<i>lrotl(32)</i>	<i>lrotr(32)</i>	<i>inp()</i>
<i>inpw()</i>	<i>memchr()</i>	<i>memcmp()</i>
<i>memcpy()</i>	<i>memset()</i>	<i>movedata()</i>
<i>outp()</i>	<i>outpw()</i>	<i>_rotl()</i>
<i>_rotr()</i>	<i>strcat()</i>	<i>strchr()</i>
<i>strcmp(16)</i>	<i>strcpy()</i>	<i>strlen()</i>

A (16) indicates that the function is inlined by the 16-bit compiler only; a (32) indicates that the function is inlined by the 32-bit compiler only.

If you specify **Oi**, the macro `__INLINE_FUNCTIONS__` is predefined by the compiler.

- l (lowercase letter “el”) Perform loop optimizations. This includes moving loop-invariant expressions outside the loops.
- m Generate inline 80x87 code for math functions such as `sin`, `cos`, `tan`, and so on. If you specify this option, you must ensure that arguments to these functions fall within the range accepted by their corresponding instructions (e.g. **fsin**, **fcos**) since no check is made at runtime. If you also specify **Ot**, the **exp** function is generated as well.
- r Enable reordering of instructions to achieve better performance on pipelined architectures such as the 486 and the Pentium. This option can make debugging a bit more difficult because the assembly language instructions generated for a source statement may be intermixed with instructions generated for surrounding statements.

- s** Favor space over time when generating code. This results in smaller code but possibly slower execution.
- t** Favor time over space when generating code. This results in faster execution but possibly larger code.
- x** Select the **-Oi**, **-Ol**, and **Wc**, **-s** (no stack overflow checking) options. When you combine **-Ox** with the **-Oa** and **-Ot** options, the code generator attempts to give you the fastest executing code possible.

By default, **cc** selects a balance between “space” and “time.”

-o *outfile*

Use the pathname *outfile* instead of the default **a.out** for the executable file. If the **-c** option is also specified, *outfile* is taken to be the name of an object file to create.

-P

Run only the C preprocessor and leave the output in the same name as the source file but with an **.i** suffix. See also option **-E**.

-p *platform*

Invoke the compiler and linker with options and libraries suitable for this operating system platform.

This option lets you compile and link code for operating systems other than QNX. This option changes the search path for headers and libraries to the following:

```
/usr/platforminclude  
/usr/platformlib
```

For example, **-p dos** would search these directories:

```
/usr/dosinclude  
/usr/doslib
```

For a list of supported platforms, see the **-bt** option for **wcc/wpp** in the *Compiler & Tools User's Guide*. Note that any headers and libraries necessary for cross-compiling must be obtained directly from Watcom.

- Q Compile more quietly without printing the details of each compiler phase.
- S Produce assembly-list files ending in .S for all source or object files provided. If an object file is specified, the original source lines won't be included in the generated assembly.
- s Strip symbolic information that may exist in object files compiled with the -g option. The symbolic information isn't removed from these object files, but the resulting executable file is stripped of all symbolic information.

-T *level*

Set the priority level of the load file, where *level* can be a number from 0 to 3 (default is 3). "T 1" is required for programs using *in*, *out*, *cli*, and *sti*. For more information, see the Description.

-U *name*

Remove any initial definition of *name* that the compiler itself might define. This doesn't affect any names defined in included header files.

-Wc,option

Pass the command specified by *option* off to the language processor, which is specified by the *c* argument. The language processor is typically the letter *c* for the C language. You may specify *c* as one of the following:

For this processor:	specify:
assembler	s
C compiler	c
C++ compiler	C
disassembler	d (used in conjunction with -s)
linker	l
preprocessor	p (used in conjunction with -E or -P)
SQL preprocessor	S

If *option* contains a space, you must enclose the entire option in double quotes (e.g. "**-w****c**, **-x** **file1**").

The command specified by *option* is defined by the language processor being used. Please refer to the appropriate documentation.

-w *warning_level*

Use this warning level. If **-w** isn't specified, the built-in default for the language processor will be used.

-x Print, but don't execute, the steps required to compile or link.

-x Use an experimental version of the compiler tools. Each command will be invoked with a leading **x** (e.g. **wcc** would be invoked as **xwcc**).

-z{*option*}

Pass through to the **-z** option of the C or C++ compiler.

Examples:

Compile **test.c** and create a 32-bit executable program in the current directory with the name **a.out**:

```
cc test.c
```

Compile **test.c** and create a 32-bit executable program in the current directory with the name **test**:

```
cc -o test test.c
```

Compile **test.c** and create an executable program in the current directory with the name **test**. Use the default rules to invoke **cc** (see the **make** utility):

```
make test
```

Compile **test.c** and create a 16-bit object file in the current directory with the name **test.o**:

```
cc -2 -c test.c
```

Disable stack overflow checking by passing the `-s` option to the C compiler:

```
cc -Wc, -s test.c
```



For more information on specific compiler options, see the *Compiler & Tools User's Guide*.

Description:

The `cc` utility is the QNX compiler interface. It is based on the `POSIX c89` utility.

The `cc` utility takes a list of source and object modules on the command line and invokes the appropriate parser to compile each file. Object modules are passed straight through to the linker. The file suffix determines which parser is used, as follows:

<code>.S, .asm</code>	Assembler
<code>.c</code>	C compiler
<code>.C, .cc, .cpp</code>	C++ compiler
<code>.sqc</code>	Embedded SQL preprocessor

The parser must exist for you to use it. The `cc` utility was designed to work with potential future parsers and may contain entries for parsers not yet available.

The `cc` utility doesn't allow multiple options to be specified after a dash character (`-`). For example, `-bc` isn't valid; you must specify `-b -c` instead. Operands (source and object files) and options may be mixed and specified in any order. Some options like `-I` and `-L` are order-dependent—the order in which they appear in the command line determines the order of the searches that are made. All command-line arguments are processed before any compilation or linking begins.

When `cc` encounters a compilation error that stops an object file from being created, it writes a diagnostic to the standard error and continues to compile other source files, but it bypasses the link phase and returns a nonzero exit status. If any file generates warnings or errors, and standard error is a tty (i.e. not a pipe), a new file will be created with the same name as the input file but with a `.err` suffix.

If the link phase is entered and is unsuccessful, a diagnostic is written and **cc** exits, once again, with a nonzero status.

The **-c** option suppresses the link phase. If you have many separate source files that you must update and modify individually, you'll probably use the **-c** option frequently.

In QNX 4.1, a program can run at one of the privity levels in the following table. The **-T** option lets you set the privity.

Privity	Description
0	Privity is determined at load time.
1	Required for programs using cli , sti , in , out opcodes. Most programs with interrupt handlers fall into this category.
2	Reserved.
3	Used by the vast majority of programs.

With level 0, the privity of the program will automatically be set when the program is loaded. If the program is invoked by the superuser, its privity will be set to 1; if the program is invoked by a normal user, its privity will be set to 3. You should find this feature useful if your program needs to do in/out opcodes only when it is run by a superuser.



All programs compiled for version 4.0 of QNX have been set to the special privity value of 0.

If a program is set to privity level 1, any attempt to execute that program by a non-superuser will result in denied access.

Although you must be a superuser to execute privity level 1 programs, privity and superuser are two different concepts. The concept of superuser comes from POSIX, while privity relates only to Intel 286, 386, 486, and Pentium processors and allows or disallows a small set of opcodes.

If you link a program under QNX 4.1 (or later), its privity will default to 3. If you find this program terminates with a SIGSEGV, check to see if it faults on one of the special opcodes that require privity level 1. If so, you should relink using the `-T 1` option.

You may occasionally wish to examine the assembly code produced by the code generator. The `-S` option produces an assembly file ending in `.s`. You can also take an object file and convert it back to assembly with this option.

If you need to specify a parameter to any of the language processors, you may use the `-wc,option`. Check the documentation on each processor to determine its options.

Exit status: 0 Successful completion.
 >0 An error occurred.

Environment variables:	LIBQNX	The list of library directories to search—directories are separated by colons. If LIBQNX isn't defined, cc uses: /usr/lib
-------------------------------	---------------	---

If **LIBQNX** is defined, it is used as is.

Files: The default output file is named **a.out**. This default may be changed with the **-o** option.

Note that the **make** utility, when used with the default settings, produces an output file with the same name as the input file. For example:

```
make file1
```

results in the executable output file: `file1`

See also: `lex`, `make`, `yacc`

Compiler & Tools User's Guide

cc/CC

compile command (UNIX)

Syntax: `Csc -c cache_size [-D] [-e pattern] [-l pattern] [prefix...] &`

Options: `-c cache_size` Set the size of the cache, specified in one of the following formats:

`n` (n kilobytes)
`nM or nm` (n megabytes)
`nK or nk` (n kilobytes)

`-D` Print debugging information, giving details of cache usage.

`-e pattern` Exclude paths that match this pattern.

`-l pattern` If a file matches this pattern, lock the file in the cache (the file will be refreshed if its content changes).

`prefix` A prefix to take over for caching. The default is / (cache the entire pathname space).

Examples: Run with 3 megabytes of cache:

```
Csc -c 3M &
```

Run with 1 megabyte of cache, include only the files under `/usr` and `/local`, and exclude a set of database files:

```
Csc -c 1024 -e "/local/database/*" /usr /local &
```

Run with 1.5 megabytes of cache and lock the system headers:

```
Csc -c 1536 -l "/usr/include/*.h" &
```

The pattern-matching mechanism used by `Csc` differs from the one used by the Shell. Thus the `*` in the second example will match all files in the `/usr` and `/local` directories as well as all files in their subdirectories (e.g. `/usr/bin`). Likewise, the `*` in the third example will match all the `.h` files in `/usr/include` as well as all the `.h` files in its subdirectories.

Description: The client-side cache (`Csc`) implements local caching on a network workstation (i.e. on the client side of a network connection). `Csc` quickly learns your working set of utilities, source files, data files,

and so on, and keeps them up-to-date in local cache. Whenever you use a file, **Csc** verifies it has the most current version and then updates the cache, if necessary. Thus you gain the performance advantage of a ramdisk without incurring any of the file version problems possible with a ramdisk. For example, if you're programming on a workstation, **Csc** will speed up your edit/compile/debug cycle but won't interfere with updates to your source files, headers, and development tools.

All file reading goes through **Csc**, which caches the data read. If a file is reread, the data will be returned from cache, thus saving the time that would otherwise be spent going across the network. **Csc** implements a sparse cache, so only the accessed portions of a file are actually placed in the cache (this is especially useful when working with link library files).

Csc only supports files opened for read-only; if a file is opened for read/write or write-only, the I/O requests go directly to the filesystem.

Csc verifies a cached file only when the file is opened or when a stat function is called (e.g. when **ls** is run). This may result in cache inconsistencies for files that are frequently updated (e.g. database files). To avoid this potential problem, either use **-e** to exclude frequently updated files or have all programs open such files for read/write. Either way, **Csc** will be bypassed and the data won't be cached.

See also: **Fsys, Pipe**

Caveat: You need to start the Pipe Manager (**Pipe**) before you run **Csc**.

Syntax: **Dev.ansi** [-C *number*] [-H] [-I *number*]
 [-k *rate* [, *delay*]] [-N *name*] [-n *number*]
 [-O *number*] [-Q] &

Options:

- C *number* Set the size of the canonical input buffer (default is 256).
- H Disable hardware scrolling.
- I *number* Set the size of the raw input buffer (default is 256).
- k *rate* [, *delay*]
 Set the keyboard rate in Hz (default is 30) and the delay in milliseconds (default is 500).
 To prevent **Dev.ansi** from setting the keyboard rate to the default value of 30 Hz, you should specify -k 0. This may be necessary with some keyboards.
- N *name* Use this prefix for the device name (default is **ansi**).
- n *number* Allow no more than this number of virtual consoles (default is 1).
- O *number* Set the size of the output buffer (default is 2048).
- Q Start in QNX terminal-emulation mode (default is ANSI).

Examples:

- Dev.ansi** & Start **Dev.ansi**; use defaults.
- Dev.ansi** -n 4 & Allow 4 virtual consoles.

Description: The **Dev.ansi** driver manages console devices in a QNX system. You can start **Dev.ansi** only after the Device Manager (**Dev**) has been started.

Hardware scrolling If **Dev.ansi** detects that your video hardware can support hardware scrolling, it programs the hardware to do so, making scrolling a lot faster. However, if you run a program that's sensitive to the state of the video hardware, this use of hardware scrolling might interfere with the program's video output. If it does interfere, specify the -H option.

Dev.ansi

console driver (QNX)

Terminal-emulation modes

When **Dev.ansi** is started, it makes the terminal type the same for all its consoles. By default, the consoles will all emulate the ANSI terminal. If you specify **-Q**, they will all emulate the QNX terminal.

If you choose to start **Dev.ansi** without **-Q**, you'll need to change the entry in the **/etc/default/login** file that sets the **TERM** environment variable.

Each console keeps track of its terminal emulation mode. To change the emulation mode of a particular console, run the following commands on that console:

To change to:	run these commands:
qnx emulation	TERM=qnx stty protocol=0 term=\$TERM
ansi emulation	TERM=ansi stty protocol=1 term=\$TERM



Unlike the **TERM** environment variable, **stty term** settings aren't restored when a shell script terminates. Therefore, if you run the above pairs of commands in a shell script, you must either reload the **stty** settings before exiting from the shell script or reload the **stty** settings from the command line after the script terminates.

See also: **Dev**, **Dev.con**, **Dev.par**, **Dev.ptty**, **Dev.ser**, **stty**, *User's Guide*

Syntax: `esh [-c command] [-irv] [script_file]`

Options: `-c command`

Run this command.

`-i` Enter interactive mode after running a script file. If you don't specify this option, **esh** will terminate after running a script.

`-r` Run in restricted mode. In this mode, you can't perform certain operations. If you try to, you'll see the error message, "Operation not permitted."

Restricted operations include creating remote processes and specifying a path that starts with a double slash for **cd** and other command names. For more information, see "Command-line format" and "Built-in commands."

`-v` Be verbose: echo each command before executing it.

script_file

A file containing shell commands to execute.

Examples: Invoke a subshell:

```
esh
```

Invoke a subshell with a script file:

```
esh /etc/backup
```

Run the following command and exit:

```
esh -c "ls /bin"
```

Description: The **esh** utility provides a subset of the functionality found in the standard QNX Shell, **/bin/sh**. You should find **esh** useful for situations where memory requirements are limited. For example, you could use it to run a simple system initialization file for an embedded system.

Command-line format:

In **esh**, command lines take this form:

```
[//node] command arg1 arg2 ... [redir-op file] [&]
```

where:

//node

The node the command is to be executed on. Remote execution isn't allowed in restricted mode.

command

The command to be executed. If it doesn't start with a slash, the command will follow the path set by the **PATH** environment variable. In restricted mode, a command can't start with a slash.

redir-op file

A redirection operator. When a command is invoked, three standard files are set up in its environment. These files, standard input, standard output, and standard error output (**stdin**, **stdout**, **stderr**), are usually attached to the active terminal. You can redirect a command's standard input, standard output, and standard error as follows:

Specifying:	will:
<i><file</i>	redirect standard input from this file.
<i>>file</i>	redirect standard output to this file. If the file exists, it's overwritten; if the file doesn't exist, it's created.
<i>>>file</i>	redirect standard output to this file. If the file exists, the information is appended to the end of the file; if the file doesn't exist, it's created.
<i>2>file</i>	do the same as <i>>file</i> , but for standard error.
<i>2>>file</i>	do the same as <i>>>file</i> , but for standard error.

- & If a command contains an unquoted **&**, then **esh** doesn't wait for the command to complete execution but immediately moves on to process the next command. The standard input of the command is redirected from **/dev/null**, and SIGINT and SIGQUIT are ignored.

Filename expansion:

You use most QNX commands for manipulating files in one way or another. As such, **esh** has a filename "shorthand" (consisting of *****, **?**, **[**, and **]**) you can use to specify the files that a command is to operate on. This shorthand is the same used by the standard QNX Shell. For more information, see "Filename expansion" in **sh**.

Quoting:

The following characters have a special meaning in **esh**:

& \ " * ? [<space>

To suppress the special meaning of these characters and keep their literal meaning, you use quoting.

To quote a sequence of characters or sequence of words, enclose the sequence in double quotes. To quote a single character, use double quotes or precede it with the escape character (****).

Escape character (backslash)

The escape character (****) preserves the literal meaning of the next character. You can't obtain a single backslash by quoting **** with double quotes. To obtain a backslash, enter **** instead.

Double quotes

Enclosing characters and words in double quotes (**" "**) preserves the literal meaning of all characters within double quotes, with the exception of the **** character. For example:

"ab cd"

represents a single, five-character argument.

You can keep the literal meaning of a double quote with the **** character. For example:

ab\"cd

represents the single, five-character argument **ab"cd**.

Built-in commands: The following commands are “built in” to **esh**. That is, **esh** interprets and executes them internally.

alias list aliases

alias [*name=value...*]

Without arguments, **alias** lists all aliases and their values. If only the name is specified, its value is listed. Any name specified with a value defines an alias.

Alias expansion occurs when the first word of a statement is a defined alias, except when that alias is already being expanded.

List all aliases:

```
alias
```

Remove an alias:

```
alias name=
```

Some common aliases:

```
alias cp="cp -i"  
alias files="/bin/ls -alFR"  
alias l="/bin/ls -lFR"  
alias lf="/bin/ls -CF"  
alias m=less  
alias md=mkdir  
alias rd=rmdir  
alias from="fgrep 'From ' $MAIL || echo No mail."
```

cd change working directory

cd [*directory*]

The **cd** utility changes the working directory of the current execution environment. If *directory* isn't specified, the value of the **HOME** environment variable becomes the new working directory. If

restricted mode (**-r**) is set, you can't start the pathname with a double slash.

exec execute a command and/or manipulate file descriptors

exec [*command* [*argument...*]]

The **exec** command opens, closes, or copies file descriptors as specified by any I/O redirections given as part of *argument*. If a command is specified, that command is spawned as a replacement for **esh**. Any specified arguments are passed to the spawned process.

exit cause **esh** to exit

exit [*n*]

The **exit** command causes **esh** to exit with the exit status specified by *n*. If *n* isn't specified, **esh** exits with the status of the last command executed.

export mark environment variables for export

export *name* [=word] ...

export -p

The **export** command marks the specified names for **export**, which will cause them to be in the environment of subsequently executed commands. If you specify the **-p** option, the names and values of all exported variables are written to the standard output. If restricted mode (**-r**) is set, you can't use this command.

reopen attach input and output to a device

reopen [*device*]

The **reopen** command closes standard input, standard output, and standard error and attaches them to the specified device. This

command is often used in **sysinit** files. If you don't specify a device, **//0/dev/con1** is used.

If restricted mode (**-r**) is set, you can't use this command.

set -v Enable verbose mode.

This command causes all commands to be echoed to the standard output before they're executed.

Environment variables:

You can use the following variables to modify **esh**'s environment:

- HOME** The pathname of the user's home directory
- LOGNAME** The user's **login** name.
- PATH** The directory search path used by **esh** for locating executable programs and **esh** shell scripts. To change **PATH**, you must use the **export** command. If **PATH** isn't in the existing environment when **esh** is invoked, it is set to **/bin:/usr/bin**.
- SHELL** The pathname of the user's preferred shell.
- TERM** The terminal type.
- TMPDIR** The pathname of a directory that can be used by utilities that need a place to create temporary files.
- TZ** The timezone setting.

See also: **sh**

Syntax: `Fsys.aha4scsi` [*general_options*] [*fsys fsys_options*]
 [*aha4scsi aha4scsi_options*]...

General options: If specified, these options must precede any **fsys** or **aha4scsi** options:

-a *unit=type*

Add this unit to the driver's device list. You should find this option handy in situations where a SCSI device might not be powered up when the driver performs its scan. Without **-a**, the only devices the driver will support are those that respond during the scan. You can specify multiple **-a** options.

The *unit* can be one of the following combinations (the number of commas affects how the unit information is interpreted; see also the Examples section):

<i>controller, target, lun</i>	fully specified controller, target, and LUN
<i>controller, target</i>	equivalent to <i>controller, target, 0</i>
<i>target</i>	equivalent to <i>0, target, 0</i>

In the *type* argument, you can specify one of the following SCSI device types:

<i>type</i>	string equivalent
0	direct-access (e.g. hard disk)
1	sequential-access (e.g. tape)
2	printer
3	processor
4	WORM
5	CD-ROM
6	scanner
7	optical
8	medium_changer
9	communications

To specify the *type*, you can use either the number or its string equivalent. When specifying a string, you need include only as many characters as are necessary to uniquely identify the type. For example to indicate type 1, you could specify **se**. Note that strings are case insensitive.

Fsys.aha4scsi

driver for Adaptec 4 SCSI host adapter (QNX)

Not all of the above types are currently supported by the Filesystem Manager. See the **-n** option.

- q** Be quiet: don't display any information on startup.
- v** Be verbose: display full information about SCSI units (devices) on startup.

fsys options: These options control the driver's interface to **Fsys**. If specified, they must follow the **fsys** keyword:

-B num

Use this many 512-byte buffers for blocking/deblocking (default is 8). You need to specify this option if you have one or more units that don't use 512-byte sectors. CD-ROM units, for example, may use 2048-byte sectors.

The value of *num* is the sum of all the buffers required for SCSI units that use sectors larger than 512 bytes. To obtain this value, you must repeat the following formula for each unit:

$$num = (2 \times (sector_size \div 512)) + \dots$$

-d Allow concurrent I/O with other DMA-based drivers.

Some ISA motherboards have problems when on-board DMA and bus-mastering DMA occur concurrently. To avoid those problems, **Fsys.aha4scsi** and **Fsys.floppy** perform I/O concurrently only if the **-d** option is specified to either or both drivers. In general, you shouldn't use **-d** on an ISA machine if the floppy is going to be used at the same time as the SCSI hard disk. This option may be used safely on MCA and EISA machines.

-f name

If a floptical disk is detected, use this name instead of the direct-access name. For more info, see the **-n** option.

-h num

Report this many heads to **Fsys** for hard disks (default is 64). The QNX filesystem doesn't need this information for normal operation. The information is needed only to let **fdisk** write the correct boot cylinder for booting.

-M *num*

Allow no more than this many units to register with **Fsys** (default is the number of SCSI units available).

-m *num*

Transfer no more than this many 512-byte sectors from the SCSI controller to memory in one operation (default is 127).

-N *name*

Assign this unique version name to this invocation of the driver. The name can range from 1 to 8 characters in length.

-n *type=name*

For the given SCSI unit type, use the specified name. For example, to have hard drives use `/dev/scsin`, you would specify **-n 0=scsi**. You can specify multiple **-n** options.

By default, the driver assigns names as follows:

<i>type</i>	string equivalent	default <i>name</i>
0	direct-access (e.g. hard disk)	hd
1	sequential-access (e.g. tape)	tp
4	WORM	wo
5	CD-ROM	cd
7	optical	mo
—	—	fl*

* Although floptical disks are direct-access devices, the driver assigns them the name **fl** instead of the default direct-access name, **hd**. To specify a different name string for floptical disks, use the **-f name** option.

As in the **-a** general option, you can specify the *type* using either the number or its string equivalent.

To have the driver ignore a type of device, specify an empty setting. For example, to ignore hard disks, specify:

-n 0=

Fsys.aha4scsi

driver for Adaptec 4 SCSI host adapter (QNX)

- R Disable reserving the SCSI unit when the unit is open.
- r Reassign bad blocks automatically. If a bad block is found on a drive, a “REASSIGN BLOCKS” command will be given to the controller, causing the drive to map the block through a defect-mapping table.
Use this option with care since bad blocks could be erroneously reported by the device if the SCSI bus isn’t properly terminated. If this occurs and you’ve specified -r, the defect-mapping table will be filled unnecessarily.

aha4scsi options:

These options control the driver’s interface to the AHA 4 series controllers. If you’ve installed multiple controllers, you can repeat these options for each controller. Remember, however, to specify the **aha4scsi** keyword before each controller’s set of options.

-a *controller*

Use this controller address. By default, the driver scans for all AHA 4 controllers. With this option, the driver doesn’t perform a scan and uses only the controllers you specify.

-b *timeout*

Set the ISA/EISA Bus busy timeout. By default, the driver queries the card for this value. For possible timeout values, see the *AHA-1540A/1542A Technical Reference* (available from Adaptec).

-d *dma_channel*

Assume that the controller uses this DMA channel for bus mastering. By default, the driver queries the controller for this value.

-f Do a full scan for the installed SCSI units. By default, the driver uses the controller’s built-in scan (see the -u option).

You should specify this option for any controller whose built-in scan may ignore removable media.

If you have removable hard disks and don’t want to use the -f option with the 1540C controller, you must enable the controller’s advanced configuration option “Support Removable Disks under BIOS as Fixed Disks.”

If you specify **-f** and you use the **mount** command in your boot image or **sysinit** file, you may need to specify the **-w** (wait) option to **mount**. The full scan may take longer than **mount**'s default wait time.

-i *irq*

Assume that the controller is using this interrupt. By default, the driver queries the controller for this value.

-m *mboxes*

Support this many mailboxes (default is 16). Mailboxes are used for simultaneous access to multiple SCSI units.

-r Reset the controller and SCSI bus at initialization time.

-t *speed*

Set the ISA/EISA Bus transfer speed. For the possible values of *speed*, which is specified in hex, see the *AHA-1540A/1542A Technical Reference*.

-u *num*

Allow no more than this many SCSI units per controller.

You may need to specify this option if you've also specified **-f**. By default, the driver sets the maximum number of units (devices) to the number of units found during the built-in scan. But with **-f**, the driver sets the maximum number of units to 4.

-x *on,off*

Set the on-and-off duration of the ISA/EISA bus. For possible values, see the *AHA-1540A/1542A Technical Reference*.

Fsys.aha4scsi

driver for Adaptec 4 SCSI host adapter (QNX)

Examples: Support all AHA 4 series controllers:

```
Fsys.aha4scsi
```

Register the direct-access unit as `/dev/hardn` and the sequential-access unit as `/dev/tapen`:

```
Fsys.aha4scsi fsys -n direct=hard -n seq=tape
```

Support a single AHA controller at address 0x334 and run quietly. Register direct-access units (e.g. hard disks) as `/dev/scsin`:

```
Fsys.aha4scsi -q fsys -n 0=scsi aha4scsi -a 334
```

Use sixteen 512-byte deblocking buffers for two CD-ROMs that each use 2048-byte sectors:

```
Fsys.aha4scsi fsys -B 16
```

In the above example, the argument for `-B` was calculated as follows:

$$16 = (2 \times (2048 \div 512)) + (2 \times (2048) \div 512))$$

Support all AHA 4 series controllers and a CD-ROM that has a SCSI target ID of 4 (in case the CD-ROM isn't powered up when the driver starts, we've used the `-a` option):

```
Fsys.aha4scsi -a 4=CD
```

If you had two SCSI controllers and the CD-ROM was connected to the second one (determined by the driver's scan), you would modify the above command as follows:

```
Fsys.aha4scsi -a 1,4=CD
```

Description: **Fsys.aha4scsi** is the driver for the Adaptec 4 series of SCSI disk controllers, which include these models:

AHA-1540
AHA-1540A/1542A
AHA-1540B/1542B
AHA-1540C/1542C
AHA-1640
AHA-1740 (in 1540 emulation mode)

Fsys.aha4scsi scans for controllers in the command-line sequence given by the **-a** option. If no **-a** option is given, the driver scans for controllers at the following hex addresses, in order:

330
334
130
134
230
234

Controllers are numbered from 0 to *n*, in the order they're found.

For each controller, the driver performs a scan, looking for installed units. All targets are scanned (0 to 7) and for each target, each LUN (Logical Unit Number) is scanned (0 to 7). Devices are numbered starting from 0, and each type of device is numbered separately. For example, if you have a hard disk and a CD-ROM device, they'll be numbered **hd0** and **cd0**, respectively.

The Filesystem Manager supports only the following types of SCSI units:

- 0 direct-access (e.g. hard disk)
- 1 sequential-access (e.g. tape)
- 4 WORM (write once/read multiple)
- 5 CD-ROM
- 7 optical

Tape units If a tape unit is found during the driver's scan or if you specify a sequential-access unit with the **-a** general option, two block special files are created under **/dev**. By default, they're named **/dev/tp0**

Fsys.aha4scsi

driver for Adaptec 4 SCSI host adapter (QNX)

and `/dev/tpr0`. Both these files reference the same physical drive, but `/dev/tpr0` will perform a rewind on a close.



To boot from a unit connected to the controller, make sure the controller is configured for a 64-head BIOS or specify the driver's **`fsys -h`** option.

See also: **`Fsys`**, **`Fsys.aha`**, *AHA-1540A/1542A Technical Reference* (available from Adaptec)

Syntax: **Fsys.ide** [-a *controller*] [-c *drivespec*] [-i *interrupt*]
 [-m *maxsize*] [-N *version_name*] [-n *device_name*]
 [-P]

Options:

-a <i>controller</i>	Assume the controller is at this address (default is 1f0 hex).
-c <i>drivespec</i>	Assume these parameters for the drive. You specify these in the following format: <i>heads, cylinders, sectors [, disknum]</i> You must include <i>heads</i>, <i>cylinders</i>, and <i>sectors</i>. If you omit <i>disknum</i>, unit 0 is assumed.
-i <i>irq</i>	Assume the controller is using this interrupt (default is 14).
-m <i>maxsize</i>	Allow no more than this many blocks to be transferred from the disk to memory in one operation (default is 127).
-N <i>version_name</i>	Assign this unique version name to this invocation of the driver. The name can be from 1 to 11 characters in length.
-n <i>device_name</i>	Assign this name to the block special device associated with the hard disk. The default is hd , which yields a device name of /dev/hd<i>n</i> , where <i>n</i> is a unit number assigned by Fsys .
-P	Run the driver as a separate process rather than as an extension of Fsys . This option improves interrupt latency, but it may also reduce throughput when used with non-caching controllers and slow machines.

Description: **Fsys.ide** is the driver for the Integrated Drive Electronics (IDE) disk drive interface. The IDE technology integrates the traditionally host-based controller electronics within the drive itself.

Fsys.ide

driver for IDE fixed disk interface (QNX)

Manufacturers of IDE drives have collectively endorsed a standards process that defines a common command and cabling specification. The standard they've developed is now a "working draft" of the American National Standard of Accredited Standards Committee X3. Most IDE drives also conform to the defacto-standard IBM PC/AT controller. Such drives will work with the **Fsys.wd** driver. For drives that vary from our interpretation of the IBM PC/AT specification and won't work with **Fsys.wd**, use **Fsys.ide**.

Fsys.ide and **Fsys.wd** share several options that are functionally identical. However, note that the default I/O size in **Fsys.ide** is 127 blocks (compared to 34 in **Fsys.wd**).

In order to obtain drive parameters, **Fsys.ide** examines values established by the BIOS of your computer. If necessary, you can override these default parameters with the **-c** option.

To define the largest single read or write the driver will support, use the **-m maxsize** option; the default value for this option is 127.

To help prevent you from unintentionally starting the same driver twice, **Fsys** rejects the invocation of any driver that has the same version name as a previously mounted driver. So if you wish to run multiple copies of **Fsys.ide**, you must use the **-N version_name** option to assign a unique version name to the driver. (You can display the version name of a driver with the **sin ver** command.) You should also use the **-n device_name** option to specify a unique device name, since another driver may have already created the default device name.



Avoid starting **Fsys.ide** and **Fsys.wd** on the same machine. Note that **Fsys** won't detect that both drivers are running.

See also: **Fsys**, **sin**, other **Fsys** drivers

Caveats: Some IDE drives don't match the draft standard that **Fsys.ide** assumes (*AT Attachment Interface for Disk Drives*, Rev 4, 17-Mar-93). If a drive won't function with either **Fsys.wd** or **Fsys.ide**, please inform our technical support group. We'll need the make, model, and year of purchase of your drive.

Syntax: **Fsys.ps2scsi** [*general_options*] [**fsys** *fsys_options*]
[**ps2scsi** *ps2scsi_options*] ...

General options: If specified, these options must precede any **fsys** or **ps2scsi** options:

-a *unit=type*

Add this unit to the driver's device list. You should find this option handy in situations where a SCSI device might not be powered up when the driver performs its scan. Without **-a**, the only devices the driver will support are those that respond during the scan. You can specify multiple **-a** options.

The *unit* can be one of the following combinations (the number of commas affects how the unit information is interpreted; see also the Examples section):

controller, target, lun fully specified controller, target, and LUN
controller, target equivalent to *controller, target, 0*
target equivalent to *0, target, 0*

In the *type* argument, you can specify one of the following SCSI device types:

<i>type</i>	string equivalent
0	direct-access (e.g. hard disk)
1	sequential-access (e.g. tape)
2	printer
3	processor
4	WORM
5	CD-ROM
6	scanner
7	optical
8	medium_changer
9	communications

To specify the *type*, you can use either the number or its string equivalent. When specifying a string, you need include only as many characters as are necessary to uniquely identify the type. For example to indicate type 1, you could specify **se**. Note that strings are case insensitive.

Fsys.ps2scsi

driver for IBM PS/2 SCSI host adapter (QNX)

Not all of the above types are currently supported by the Filesystem Manager. See the **-n** option.

- q** Be quiet: don't display any information on startup.
- v** Be verbose: display full information about SCSI units (devices) on startup.

fsys options: These options control the driver's interface to **Fsys**. If specified, they must follow the **fsys** keyword:

-B num

Use this many 512-byte buffers for blocking/deblocking (default is 8). You need to specify this option if you have one or more units that don't use 512-byte sectors. CD-ROM units, for example, may use 2048-byte sectors.

The value of *num* is the sum of all the buffers required for SCSI units that use sectors larger than 512 bytes. To obtain this value, you must repeat the following formula for each unit:

$$num = (2 \times (sector_size \div 512)) + \dots$$

-d Allow concurrent I/O with other DMA-based drivers.

Some ISA motherboards have problems when on-board DMA and bus-mastering DMA occur concurrently. To avoid those problems, **Fsys.ps2scsi** and **Fsys.floppy** perform I/O concurrently only if the **-d** option is specified to either or both drivers. In general, you shouldn't use **-d** on an ISA machine if the floppy is going to be used at the same time as the SCSI hard disk. This option may be used safely on MCA and EISA machines.

-f name

If a floptical disk is detected, use this name instead of the direct-access name. For more info, see the **-n** option.

-h num

Report this many heads to **Fsys** for hard disks (default is 64). The QNX filesystem doesn't need this information for normal operation. The information is needed only to let **fdisk** write the correct boot cylinder for booting.

-M num

Allow no more than this many units to register with **Fsys** (default is the number of PS/2 SCSI units available).

-m num

Transfer no more than this many 512-byte sectors from the SCSI controller to memory in one operation (default is 127).

-N name

Assign this unique version name to this invocation of the driver. The name can range from 1 to 8 characters in length.

-n type=name

For the given SCSI unit type, use the specified name. For example, to have hard drives use **/dev/scsi**, you would specify **-n 0=scsi**. You can specify multiple **-n** options.

By default, the driver assigns names as follows:

<i>type</i>	<i>string equivalent</i>	<i>default name</i>
0	direct-access (e.g. hard disk)	hd
1	sequential-access (e.g. tape)	tp
4	WORM	wo
5	CD-ROM	cd
7	optical	mo
—	—	f1*

* Although floptical disks are direct-access devices, the driver assigns them the name **f1** instead of the default direct-access name, **hd**. To specify a different name string for floptical disks, use the **-f name** option.

As in the **-a** general option, you can specify the *type* using either the number or its string equivalent.

To have the driver ignore a type of device, specify an empty setting. For example, to ignore hard disks, specify:

-n 0=

Fsys.ps2scsi

driver for IBM PS/2 SCSI host adapter (QNX)

- R Disable reserving the SCSI unit when the unit is open.
- r Reassign bad blocks automatically. If a bad block is found on a drive, a "REASSIGN BLOCKS" command will be given to the controller, causing the drive to map the block through a defect-mapping table.
Use this option with care since bad blocks could be erroneously reported by the device if the SCSI bus isn't properly terminated. If this occurs and you've specified -r, the defect-mapping table will be filled unnecessarily.

ps2scsi options:

These options control the driver's interface to the PS/2 SCSI controller. Remember to specify the **ps2scsi** keyword before these options.

-a *controller*

Use this controller address.

-i *irq*

Assume the controller is using this interrupt. By default, the driver queries the controller for this value.

-u *num*

Allow no more than this many SCSI units per controller (default is 4).

Examples:

Support a PS/2 SCSI controller:

```
Fsys.ps2scsi &
```

Register the direct access unit as **/dev/hardn** and the sequential access unit as **/dev/tapen**:

```
Fsys.ps2scsi fsys -n direct=hard -n seq=tape &
```

Support a controller at address 0x3540 and run quietly. Register direct access units (e.g. hard disks) as **/dev/scsin**:

```
Fsys.ps2scsi -q fsys -n 0=scsi ps2scsi -a 3540 &
```

Support a PS/2 SCSI controller with 6 SCSI units and don't display the list of connected devices. Register direct access units (e.g. hard disks) as `/dev/scsin`:

```
Fsys.ps2scsi -q fsys -n 0=scsi ps2scsi -u 6 &
```

Use sixteen 512-byte deblocking buffers for two CD-ROMs that each use 2048-byte sectors:

```
Fsys.ps2scsi fsys -B 16
```

In the above example, the argument for `-B` was calculated as follows:

$$16 = (2 \times (2048 \div 512)) + (2 \times (2048) \div 512))$$

Description:

Fsys.ps2scsi is the driver for the IBM PS/2 SCSI controller.

Fsys.ps2scsi scans for a controller in the command-line sequence given by the `-a` option. If no `-a` option is given, the driver scans for a controller at the following hex addresses, in order:

3540
3548
3550
3558
3560
3568
3570
3578

The driver then scans for installed units. All targets are scanned (7 to 0). The driver numbers the units (starting at 0) and registers them with **Fsys** in the order it finds them.

The Filesystem Manager supports only the following types of SCSI units:

- 0 direct-access (e.g. hard disk)
- 1 sequential-access (e.g. tape)
- 4 WORM (write once/read multiple)
- 5 CD-ROM
- 7 optical

Fsys.ps2scsi

driver for IBM PS/2 SCSI host adapter (QNX)

Tape units If a tape unit is found during the driver's scan or if you specify a sequential-access unit with the **-a** general option, two block special files are created under **/dev**. By default, they're named **/dev/tp0** and **/dev/tpr0**. Both these files reference the same physical drive, but **/dev/tpr0** will perform a rewind on a close.



To boot from a unit connected to the controller, make sure the controller is configured for a 64-head BIOS or specify the driver's **fsys -h** option.

See also: **Fsys, Fsys.aha, Fsys.aha4scsi**

Syntax: `fsysinfo [-b] [-d secs] [-r] [file]`

Options:

- `-b` Base the counts from the time **fsysinfo** starts (useful only with `-d`)
- `-d secs` Update the full-screen display every *secs* seconds. To exit the display, you press <Ctrl>-<C>.
- `-r` Reset **Fsys**'s internal counters. To use this option, you must be logged in as **root**.
- file* Display statistics for this file (default is ".").

Examples: Display statistics for the current directory:

```
fsysinfo
```

Reset the stat counters in the **Fsys** that's responsible for `/tmp`:

```
fsysinfo -r /tmp
```

Update the statistical display every three seconds, showing **Fsys** statistics for `/`:

```
fsysinfo -d3 /
```

Description: The **fsysinfo** utility displays statistical information obtained from the Filesystem Manager (**Fsys**). The utility displays this information in three sections:

- configurable limits
- cache-and-disk statistics
- critical-function statistics

The **fsysinfo** utility displays this information only once unless you specify `-d`.

Because **Fsys** doesn't proportion its cache according to partitions, disks, or drivers, the statistics you see represent the total usage of the **Fsys** responsible for the named path (i.e. the statistics don't reflect usage for just that path).

If you specify the `-b` option, the statistics shown in the cache-and-disk and critical-function sections show activity from the

fsysinfo

display filesystem statistics (QNX)

time **fsysinfo** started. Otherwise, they show activity from when **Fsys** started or from when you reset **Fsys**'s counters with **fsysinfo -r**.

Configurable limits

The configurable limits section shows information for the following configurable **Fsys** limits:

Limit	Fsys option for configuring limit
inodes	-i
file descriptions (open control blocks)	-f
locks	-l
threads	-t

For each of these configurable limits, the display shows the:

- current value
- greatest value that has been used at one time
- limit set by **Fsys**

If your machine is tight on memory you can use this information to tune your **Fsys** limits. You should find the default values generous for your system.

Cache-and-disk statistics

The cache-and-disk statistics can help you tune the cache size to obtain the best performance. Generally, the higher you raise the cache hit ratio (displayed as a percentage on the first line of these statistics), the better the performance.

This statistic:	represents the number of:
cache hits	times Fsys had to read a block and found that block already in cache. A cache hit may occur when a program does a read, or when Fsys performs an internal function (e.g. directory scanning).

This statistic:	represents the number of:
cache delays	times a driver started reading a requested block that wasn't yet available (Fsys performs a pre-read when it recognizes that the file usage pattern is sequential). If Fsys pre-reads a single block, any subsequent blocks for the read request that reside in cache are also considered delayed and are added to this number.
cache misses	times the requested block wasn't in cache, and Fsys had to start a disk read.
cache lookups	reads, writes, and other internal functions that have been performed by Fsys (does <i>not</i> represent the sum of hits, delays, and misses).
cache writes	cache blocks that have been queued for writing to disk. Because of write buffering, a block may be queued many times before hitting the disk.
disk reads	block read requests Fsys has made to its drivers.
disk writes	block write requests Fsys has made to its drivers.

Critical-function statistics

These statistics represent how often certain functions in **Fsys** have been called. You can't use this information for tuning purposes, but it does give an idea of how busy the system is.

With the exception of *namei()*, which is the function used to map a pathname to an in-memory inode, all the critical function names have a one-to-one correspondence with library functions of the same name.

fsysinfo

display filesystem statistics (QNX)

When you run **fsysinfo** in full-screen display mode (**-d**), the critical-functions section displays:

- the number of times each function was called
- a per-second count of the number of calls since the last display
- the average number of calls per second since the **Fsys** counters were initialized (or since **fsysinfo** was started if started with **-b**)

When you don't run **fsysinfo** in full-screen mode, it displays only the number of times each function was called and the average number of calls per second.

See also: **Fsys**

Syntax: `int10 [-a] [-b type] [-c console|none] [-R]
 [-s script_file] [-t trans_file] [-v] [command|&]`

Options: **-a** Auto-detect BIOS types; this is the default. The options **-b**, **-t**, and **-s** override auto-detection; use **-a** to add auto-detected types.

-b type
 Set BIOS type, overriding auto-detection.

-c console | none
 Attach to this console (default is standard error) or don't assume any console.

-R Display CPU registers before and after INT 10 calls.

-s script_file
 Load this SVPMI script file, overriding auto-detection.

-t trans_file
 Use this translation file, overriding auto-detection. For a description of translation files see
 `/etc/readme/technotes/int10`.

-v Be verbose.

command|&

If `int10` isn't run in the background (with `&`), you can specify one of the following optional display commands:

This <i>command</i> :	displays:
id	current BIOS type
modes	list of supported video modes
query	list of supported BIOS types

Once the information is displayed, `int10` will terminate.

int10

provide some of the BIOS INT 10 functions (QNX)

Examples: Auto-detect the video BIOS and display the list of modes:

```
int10 modes
```

Auto-detect the video BIOS and install the INT 10 service :

```
int10 &
```

Load an SVPMI script and install the INT 10 service:

```
int10 -s vga &
```

Description: The `int10` utility provides support for programs using the INT 10 software interrupt. It provides only those INT 10 BIOS functions (as well as the VESA Super VGA BIOS Extension) required by programs that use the Watcom Graphics Library. Once the emulator is running, programs can use the INT 10 software interrupt to change video modes and to set color-palette entries.

If you use `-s`, requests will be satisfied by executing code generated from the SVPMI script. Otherwise, `int10` will execute the BIOS.

When programs using INT 10 calls terminate, the `int10` utility attempts to restore the previous video mode.



To use Watcom Graphics Library functions, you must start `int10` in the background.

See also: `/etc/readme/technotes/int10`,
`/etc/readme/technotes/svpmi`,
Watcom C Graphics Library Reference

Caveat: Since `int10` interprets the BIOS code, this code runs slower than it would if it were running natively. Because of this speed delay, some BIOSs with tight timing restrictions may fail to work with `int10`. If this happens, you'll need to use the `-s` option.

Syntax: `Iso9660fsys [-w secs] [directory=file] ... &`

`Iso9660fsys [-s] [-w secs] &`

Options:

- `-s` Suppress scanning of the `/dev` directory for CD-ROM block special files.
- `-w secs` When mounting the filesystem, wait no more than this many seconds. This option is useful if you want to make **Iso9660fsys** wait for the **Fsys** driver to set up its devices.
If you don't specify this option, **Iso9660fsys** will wait no more than 60 seconds for the devices to be set up.
- `directory=file` The directory where the ISO 9660 filesystem is to be mounted and a file containing the filesystem. If you use this syntax, no scan of the `/dev` directory is performed.
The file you specify doesn't have to be a CD-ROM block special file. For example, it could be a regular file that's an image of a CD-ROM or it could be a WORM drive containing an ISO 9660 filesystem.

Examples: Scan the `/dev` directory and automatically mount any CD-ROM block special files. If the scan were to find `/dev/cd0`, the name of the newly mounted ISO filesystem would be `/cd0`:

`Iso9660fsys &`

Mount the block special file `/dev/cd0` as `/cdrom`; don't scan for other CD-ROM devices:

`Iso9660fsys /cdrom=/dev/cd0 &`

Description: The **Iso9660fsys** filesystem manager provides transparent access to ISO 9660 filesystems. It supports Rock Ridge Interchange Protocol (RRIP) extensions (for a POSIX filesystem) and also supports High Sierra filesystems.



If you use Rundos to access the mounted filesystem, Rundos will ignore the RRIP filename extensions in favor of the ISO filenames. It will also ignore filenames that don't comply with ISO 9660 level 1.

By default, **Iso9660fsys** does a scan until at least one CD-ROM block special file is found or the timeout occurs. All CD-ROM devices found during the scan are mounted. If you specify **-s** or the *directory=file* syntax, the scan isn't performed.

For *directory=file*, you should also note that the specified file needn't be an actual CD-ROM as long as it contains an ISO 9660 filesystem.

**Information
directory**

A directory called **.cd.information.** is always created beneath the directory of the mounted ISO filesystem (e.g. **/cd0/.cd.information.**). Another directory may also be created if a valid volume ID is present on that filesystem. The name of this other directory is *=volume_id* and may contain up to 11 characters.

Both of these directories contain the same files. These files contain information about the mounted filesystem. If any of the information fields described in the following table are empty, the corresponding file of the same name won't be present.

Field/File	Contents
creation	zero-length file with a time stamp of the creation date
modification	same as above, but for modification date
expiration	same as above, but for expiration date
effective	same as above, but for effective date
volumesetid	Volume set ID
publisher	CD-ROM publisher
preparer	CD-ROM preparer
application	CD-ROM application
copyright	CD-ROM copyright information
abstract	CD-ROM abstract
bibliographic	CD-ROM bibliographic (not on High Sierra CDs)
system	CD-ROM system area description
volumeid	CD-ROM volume ID (used to make the “=” directory)
rrip_version	version of Rock Ridge Interchange Protocol
control	QNX control file (described below)

Control file

The **control** file is always present. By writing commands to this special file, you can control the CD-ROM drive. For example, the following command would eject the disk from the drive:

```
echo eject >/cd0/.cd.information./control
```

Note that commands must be written atomically to this file. The following table lists the commands you can use.

Iso9660fsys

ISO 9660 Filesystem Manager (QNX)

Command	Description
raw=on off	display raw ISO 9660 filenames (default is off); if raw is turned on, ISO 9660 filenames won't be translated by Iso9660fsys
rockridge=on off	enable Rock Ridge extensions (default is on)
uid=user_id	use this user ID for ISO 9660 filenames (default is 0)
gid=group_id	use this group ID for ISO 9660 filenames (default is 0)
play [=first [-last]]	play audio CD from <i>first</i> to <i>last</i> track
volume=level	adjust the volume (0-255)
stop	stop audio CD from playing
pause	pause audio CD
resume	resume audio CD
eject	eject CD from drive
load	reload CD to drive

The play, volume, stop, pause, resume, eject, and load commands are ignored if they're not supported by the drive. If you're using a SCSI drive and it doesn't always respond, you might wish to verify that your drive supports the following SCSI commands, which are the actual directives used by the SCSI driver.

Function	SCSI type	Command name	Command code
play	SCSI2	PLAY AUDIO TRACK/INDEX	0x48
volume	SCSI2	MODE SELECT (6) page 0x0E	0x15
stop	SCSI	START STOP UNIT	0x1B
eject	SCSI2	START STOP UNIT	0x1B
load	SCSI2	START STOP UNIT	0x1B
pause	SCSI2	PAUSE/RESUME	0x4B
resume	SCSI2	PAUSE/RESUME	0x4B

See also: `mount`, `umount`

Caveats: Some CD-ROM vendors may recommend the following alternative method of mounting an ISO 9660 filesystem:

```
mount -t iso9660 [block_special_file directory] ...
```

For this command to succeed, **Iso9660fsys** must be running in the background. For example, the following commands would mount the ISO filesystem as `/cd0`:

```
Isofsys9660 -s &  
mount -t iso9660 /dev/cd0 /cd0
```

Iso9660fsys

ISO 9660 Filesystem Manager (QNX)

Syntax: `Net.arcxir -n phys_nid [-i irq] [-l log_net_id]`
`[-p io_port] [-r media_rate] [-s] [-v] &`

Options: `-i irq`

Use this interrupt line with the parallel port. By default, interrupts aren't used. The first and second parallel ports are typically associated with interrupts 7 and 5.

-1 log_net_id

Connect to this logical network (default is 1).

-n phys_nid

Program the Xircom adapter to be this physical node ID. You must specify this node ID for the driver to function, and the ID must be unique for the Arcnet network to function.

-p *io port*

Use this I/O port base (default is 0x378, the address of the first parallel port on most machines). If the default value fails, you can try 0x278, usually the address of the second parallel port.

If your parallel port is on a **TTL** monochrome video card, the address may be 0x3BC.

-r *media rate*

Advertise this bit-transmission rate (default is 25000000). The Network Manager uses this information when multiple network drivers are running.

- s Terminate the driver if SIGPWR is set as a result of a **shutdown** command. By default, the driver doesn't terminate after a **shutdown** command; this behavior helps ensure an orderly termination of the virtual circuits.

- v Be verbose: print out hardware info on the console when starting up.

Net.arcxir

Xircom Pocket Arcnet Adapter II network driver (QNX)

Examples: Use the first (and probably the only) parallel port; don't use interrupts:

```
Net.arcxir &
```

Use the first parallel port and IRQ 7:

```
Net.arcxir -i 7 &
```

Use the second parallel port; don't use interrupts:

```
Net.arcxir -p 278 &
```

Use the second parallel port and IRQ 5:

```
Net.arcxir -p 278 -i 5 &
```

Description: The **Net.arcxir** network driver communicates directly with a Xircom Pocket Arcnet Adapter II, which plugs into a parallel port of your computer. The driver provides the Network Manager with reliable data transfer over an Arcnet network.

By default, **Net.arcxir** assumes that the Xircom adapter is connected to the first parallel port and that interrupts aren't used. The driver polls the parallel port.

If your parallel port hardware supports interrupts (and most do), you can increase performance by specifying the **-i** option.



The Rundos DOS emulation package reduces the polling frequency of **Net.arcxir** and thus reduces the driver's network transfer rate. If you're running Rundos, you should specify the **-i** option.

Usually the first parallel port drives IRQ 7 and the second parallel port drives IRQ 5, but check your computer's documentation and configuration to be certain.

You must specify a physical node ID with the **-n** option since **Net.arcxir** doesn't use the parameters programmed by Xircom's DOS software.

To install:	you should:
a Xircom card	read the technote contained in the file <code>/etc/readme/technotes/Net.arcxir</code>
multiple network cards in one machine	specify a unique logical network ID with the <code>-l</code> option to every network driver
multiple Xircom cards in one machine	specify a unique I/O address with the <code>-p</code> option to every invocation of <code>Net.arcxir</code>

Exit status:

- 0 Successful.
- >0 An error occurred.

See also: `Net`, `Net.arcnet`, `netinfo`, `shutdown`, *User's Guide*

Net.arcxir

Xircom Pocket Arcnet Adapter II network driver (QNX)

Syntax: `Net.ether503 [-P] [-a] [-i irq] [-l log_net_id]
 [-m mem] [-n tx_num_retries] [-p io_port]
 [-r media_rate] [-s ram_size]
 [-t tx_retry_ticks] [-v] &`

- Options:**
- P** Run promiscuously: buffer every packet, whether or not it's destined for this node. (A node usually discards any packet not addressed to it.)
 You may use this option in conjunction with special software to "sniff" other traffic on the network.
 - a** Use the card's AUI (15-pin) port instead of its onboard transceiver (BNC, TP).
 - i irq** Use this interrupt line (IRQ). You can specify a value from 2 to 5; the default is 5.
 - l log_net_id** Connect to this logical network (default is 1).
 - m mem** Use this memory segment. This option overrides the driver's automatic detection of where the shared RAM is located.
 - n tx_num_retries**
 After failing to transmit to a remote node, retry transmission no more than this number of times (default is 7).
 - p io_port** Use this I/O port base. This option overrides the driver's automatic I/O port detection.
 - r media_rate** Advertise this bit-transmission rate (default is 10000000). The Network Manager uses this information when multiple network drivers are running.
 - s ram_size** Set the shared RAM segment on the card to this length (default is 8K). The *ram_size* argument is given in kilobytes.

-t *tx_retry_ticks*

Wait this many 50-millisecond intervals between tx retries (default is 6).

-v Be verbose: print out hardware info on the console when starting up.

Examples: Use the default IRQ as well as automatic detection of I/O port and shared RAM:

```
Net.ether503 &
```

Do the same as above, but use IRQ 2:

```
Net.ether503 -i 2 &
```

Use the default IRQ, an I/O port base of 0x250, and automatic detection of shared RAM (normally, you need to specify an I/O port only if you've installed multiple 3com 503 cards—with multiple cards present, the driver doesn't automatically detect the I/O port):

```
Net.ether503 -p 250 &
```

Use the default IRQ, a RAM segment of 0xCC00, and automatic detection of the I/O port (you don't normally need to specify the RAM segment location since the driver can automatically determine this once it has determined the I/O port base):

```
Net.ether503 -m CC00 &
```

Use IRQ 2, an I/O port base of 0x310, and a RAM segment of 0xD800:

```
Net.ether503 -i 2 -p 310 -m D800 &
```

Support two cards—one using IRQ 2 and an I/O port base of 0x2A0 on logical network 1, the other using IRQ 5 and an I/O port base of 0x330 on logical network 2. Use automatic RAM segment detection for both:

```
Net.ether503 -i 2 -p 2A0 &
```

```
Net.ether503 -p 330 -i 5 &
```

For this setup, you could configure the RAM segment at 0xC800 for the first card and at 0xCC00 for the second card. To configure the RAM segment, see the card manufacturer's documentation.

Description: The **Net.ether503** network driver communicates directly with a 3com 503 Ethernet network card operating in shared memory mode. The driver provides the Network Manager with reliable data transfer over an Ethernet network.

By default, **Net.ether503** automatically determines the I/O port base and the segment of the shared RAM. However, it doesn't automatically detect the hardware interrupt (IRQ) since the 3com 503 card cannot store this parameter. **Net.ether503** will use IRQ 5 unless you specify a different IRQ with the **-i** option.

The 3com 503 card supports only IRQs 2, 3, 4, and 5. Make sure no other devices are using the same interrupt as the 3com 503 card.

Before installing your card, note the following:

To install:	you should:
a 3com 503 card	read the technote contained in the file <code>/etc/readme/technotes/Net.ether503</code>
multiple network cards in one machine	specify a unique logical network ID (-1 option) to every network driver
multiple 3com 503 network cards in one machine	specify the -p option to every invocation of Net.ether503

Exit status: 0 Successful.
>0 An error occurred.

See also: **Net**, **Net.ether8003**, **netinfo**, *User's Guide*

Net.ether503

3com 503 Ethernet network driver (QNX)

Syntax: **Net.ether1000** [-6] [-P] [-i *irq*] [-l *log_net_id*]
 [-n *tx_num_entries*] [-p *io_port*]
 [-r *media_rate*] [-s *ram_size*]
 [-t *tx_retry_ticks*] [-v] &

- Options:**
- 6** Assume that a 16-bit NE2000 card with 16K RAM is installed. This option overrides the driver's automatic detection of the NE2000 card.
 - P** Run promiscuously: buffer every packet, whether or not it's destined for this node. (A node usually discards any packet not addressed to it.)
 You may use this option in conjunction with special software to "sniff" other traffic on the network.
 - i *irq*** Use this interrupt line (IRQ). You can specify a value from 2 to 5; the default is 3. This option overrides the automatic IRQ detection the driver performs if a boot ROM is installed.
 - l *log_net_id*** Connect to this logical network (default is 1).
 - n *tx_num_retries*** After failing to transmit to a remote node, retry transmission no more than this number of times (default is 7).
 - p *io_port*** Use this base I/O port address (default is 0x300). This option overrides the automatic port detection the driver performs if a boot ROM is installed.
 - r *media_rate*** Advertise this bit-transmission rate (default is 10000000). The Network Manager uses this information when multiple drivers are running.
 - s *ram_size*** Set the shared RAM segment on the card to this length (given in kilobytes). This option overrides the default of 8K for the NE1000 and 16K for the NE2000.

-t *tx_retry_ticks*

Wait this many 50-millisecond intervals between tx retries (default is 6).

-v

Be verbose: print out hardware info on the console when starting up.

Examples:

For descriptions of the Novell configuration options in the following examples, see your Novell documentation.

With a boot ROM
installed...

Use Novell option 9 (IRQ 5, I/O port 0x320, ROM at 0xCC00):

```
Net.ether1000 &
```

Support two cards—one using Novell option 4 (IRQ 2, I/O port 0x300) on logical network 1, and the other using Novell option 9 (IRQ 5, I/O port 0x320, ROM at 0xCC00) on logical network 2:

```
Net.ether1000 -i 2 -p 300 &
```

```
Net.ether1000 -i 5 -p 320 -l 2 &
```

With *no* boot ROM
installed...

Use the factory defaults of IRQ 3 and I/O port 0x300:

```
Net.ether1000 &
```

Use IRQ 5 and the default I/O port 0x300:

```
Net.ether1000 -i 5 &
```

Use IRQ 2 and I/O port 0x340 (Novell option 10):

```
Net.ether1000 -p 340 -i 2 &
```

Description:

The **Net.ether1000** network driver communicates directly with a Novell NE1000 (8-bit) or NE2000 (16-bit) Ethernet network card; it also works with compatible cards. The driver provides the Network Manager with reliable data transfer over an Ethernet network.

If you have installed a boot ROM using one of the following Novell configurations, **Net.ether1000** will automatically detect the IRQ and I/O port parameters (ROM address and I/O port in hex).

Novell option	ROM address	I/O port	IRQ	Default boot action
0	C800	300	3	from network
2	D000	340	4	from disk
9	CC00	320	5	from network
X	D400	360	2	from network



If you haven't installed a boot ROM or aren't using the default I/O port and the default IRQ, you must specify the **-p** and **-i** options.

Before installing a boot ROM or network card, note the following:

To install:	you should:
a boot ROM	read the documentation that came with the boot ROM
an NE1000 or NE2000 card	read the technote contained in the file <code>/etc/readme/technotes/Net.ether1000</code>
multiple network cards in one machine	specify a unique logical network ID (-1 option) to every network driver
multiple NE1000 or NE2000 or compatible cards in one machine	specify -p and -i to every invocation of Net.ether1000 even if a boot ROM is installed in one or more of the cards

Exit status: 0 Successful.
>0 An error occurred.

See also: **Net**, **Net.ether503**, **Net.ether8003**, **netinfo**, *User's Guide*

Net.ether1000

NE1000/2000 Ethernet network driver (QNX)

Syntax: **Net.f^d** [-a *phys_addr*] [-c] [-d *dev_rate*] [-f *tty*] [-h]
 [-i *timeout*] [-k] [-l *log_net_id*]
 [-m *min_com_length*] [-n *tx_num_retries*]
 [-r *media_rate*] [-t *tx_retry_ticks*] [-v] *tty* &

Options:

-a <i>phys_addr</i>	Use this 48-bit hex physical address, as Ethernet does.
-c	Don't append and check for CRC.
-d <i>dev_rate</i>	Use this device media rate for timeout calculations. By default the driver uses the baud rate of the device.
-f <i>tty</i>	Use this device for network input and output. Specify "-" for standard input.
-h	Ignore SIGHUP. By default the driver terminates on SIGHUP. This signal is sent when, for example, the carrier is lost from a modem.
-i <i>timeout</i>	Use this interbyte timeout (in tenths of seconds). The default is 10, i.e. 1 second.
-k	Don't compress.
-l <i>log_net_id</i>	Connect to this logical network (default is 1).
-m <i>min_com_length</i>	Compress packets that are this length or longer (default is 12).
-n <i>tx_num_retries</i>	After failing to transmit to a remote node, retry transmission no more than this number of times (default is 3).
-r <i>media_rate</i>	Advertise this bit-transmission rate. By default, the driver automatically calculates this value. The Network Manager uses this information when multiple drivers are running.
-t <i>tx_retry_ticks</i>	Wait this many 50-millisecond intervals between tx retries. By default, the driver automatically

Net.fd

file-descriptor-based network driver (QNX)

calculates this value, based on the length of the transmission and on the media rate.

-v

Be verbose: print out configuration info on the console when starting up.

tty

Use this device for network input and output (default is **/dev/ser1**).

Examples:

Use the first serial port (**/dev/ser1**) as logical network 1:

```
Net . fd &
```

Use the second serial port as logical network 1:

```
Net . fd /dev/ser2 &
```

Use the first serial port as logical network 2:

```
Net . fd -1 2 &
```

Support two logical networks: use the first serial port as logical network 1 and use the second serial port as logical network 2:

```
Net . fd &
```

```
Net . fd -1 2 /dev/ser2 &
```

Description:

The **Net . fd** network driver uses file-descriptor-based I/O (i.e. *open()*, *read()*, *write()*, and so on) to receive and transmit packets. It provides the Network Manager with reliable data transfer over any media supported by a file-descriptor-based server process.

For example, you could use **Net . fd** to connect two machines with a null-modem RS-232 serial cable. By using file descriptor I/O to the serial devices, **Net . fd** would implicitly use **Dev**'s serial driver (**Dev . ser**) and set up a logical network link.



You should read the technote contained in the **/etc/readme/technotes/Net . fd** file.

If you start more than one network driver on the same machine, you must specify a unique network ID (**-1** option) to every driver.

Exit status: 0 Successful.
 >0 An error occurred.

See also: **Net**, **netinfo**, *User's Guide*

Net.fd

file-descriptor-based network driver (QNX)

Syntax: **Net.tr164a** [-a] [-l *log_net_id*] [-m] [-p *io_port*]
 [-r *media_rate*] [-t *table_size*] [-v] &

Options:

- a** Assume an AT bus adapter (default is auto-detect).
- l *log_net_id*** Connect to this logical network (default is 1).
- m** Assume a microchannel adapter (default is auto-detect).
- p *io_port*** Use this I/O port base address. This option overrides the driver's auto-detect of 0xA20 to 0xA23 (for primary LAN cards) or 0xA24 to 0xA27 (for alternate LAN cards).
- r *media_rate*** Advertise this bit-transmission rate (default is 16000000). The Network Manager uses this information when multiple network drivers are running.
- t *table_size*** Set the source routing table to this size (default is 64).
- v** Be verbose: print out hardware configuration info on the console when starting up.

Examples: Use automatic detection of all hardware parameters:

```
Net.tr164a &
```

Do the same as above, but for a card on logical network 2:

```
Net.tr164a -l 2 &
```

Support two cards—one configured as the “primary LAN adapter” and connected to logical network 1, the other configured as the “alternate LAN adapter” and connected to logical network 2:

```
Net.tr164a -p A20 &
```

```
Net.tr164a -p A24 -l 2 &
```

Net.tr164a

IBM Token Ring 16/4 Adapter network driver (QNX)

Description: The **Net.tr164a** network driver communicates directly with an IBM Token Ring Network 16/4 card. It works with both ISA and MCA cards that operate in shared-memory mode (“busmaster” mode isn’t supported). The driver provides the Network Manager with reliable data transfer over a Token Ring network.

By default, **Net.tr164a** automatically locates the network adapter at either the primary (0xA20) or the alternate (0xA24) I/O port base. Once it determines the I/O port base, the driver can automatically determine all other hardware parameters.

Note that an ISA bus Token Ring card also uses I/O port 2Fn, where *n* is the interrupt (e.g. 2, 3, 7). MCA bus cards don’t use this port.

Before installing your card, note the following:

To install:	you should:
a Token Ring 16/4 card	read the technote contained in the file <code>/etc/readme/technotes/Net.tr164a</code>
multiple network cards in one machine	specify a unique logical network ID (<code>-l</code> option) to every network driver
multiple Token Ring 16/4 cards in one machine	specify the <code>-p</code> option to every invocation of Net.tr164a

Exit Status: 0 Successful.
>0 An error occurred.

See also: **Net**, **netinfo**, *User’s Guide*



You can't start the Process Manager at the command line.

In our manuals, you'll see the Process Manager referred to as **Proc**. But in your build files you must specify which version of the Process Manager you want to use:

Proc16 allows 16-bit programs to run

Proc32 allows both 16- and 32-bit programs to run

You can run only one Process Manager per computer.

Buildfile syntax:

Proc16 [*options*]

Proc32 [*options*]

Options:

- a** Disable time accounting. This option saves $16 \times \text{num_procs}$ bytes of memory and slightly reduces the work done in the timer interrupt handler.
- b** If a warm boot is requested via the **shutdown** utility or **<Ctrl>-<Alt>-<Shift>-**, tell the BIOS to do a full restart rather than a fast warm restart. This option is often useful with machines that fail to reboot when a warm boot is requested.
- D** If the debugger is bound into the operating system, suppress its automatic invocation upon boot.
- d num_major**
The number of major device numbers to be supported (default is 32).
- f min max total**
This option controls the availability of file descriptors (FDs):
 - min* The minimum number of FDs that must be available before a process will start. This argument defaults to 16 and should be left at that value if POSIX-compliance is a requirement (POSIX requires that a process be capable of opening at least 16 files).

You can save some memory by reducing this number. The vast majority of processes never use more than 5 FDs. Each FD requires 16 bytes of memory.

max The maximum number of FDs that can be available for a process (default is 512).

total The total number of FDs available in the system (default is *num_procs* × 16).

Each process starts off by taking *min* FDs, which it may grow to *max* FDs.

- i** *irq* Make *irq* the highest-priority hardware interrupt in the system. You can specify a number from 0 through 7 (default is 3).
- L** Local. Don't permit outbound network access from this computer.
- l** *node* (the letter "e") Assign the specified logical node ID to this machine (default is 1). Each connected machine must have a unique logical node ID.
- M** *machine_name*
The symbolic name for this computer (default is AT or PS/2). Some programs check for this name and alter their behavior accordingly. For example, the floppy disk driver and some PS/2 disk drivers look for the name PS/2.
- m** *num_shared_segs*
The number of shared segments (managed by the *qnx_segment()*... family of functions) that will be available in this operating system image (default is 300).
- n** *num_names*
The maximum number of names that can be registered locally by the *qnx_name_attach()* function (default is 30).

-P *letter*

Patch the kernel code according to the given *letter*, which may be **one** of the following:

- n** Disallow nested interrupts. This patch won't allow an interrupt to be interrupted by another interrupt.
- r** Disallow restarts of kernel operations if an interrupt schedules a process when a process with a lower priority is in the Kernel. While this option might improve system throughput, it will increase process scheduling latency.

-P *num_procs*

The total number of real processes, virtual circuits, and proxies that can exist at any time (default is 150).

-R *num_64k*

Reserve all of base memory (0 .. 640K) and the indicated number of 64K pieces starting above 1M. For example, a value of **-R 0** would reserve all of base memory and start QNX at 1M. You may need this option only if you're using Rundos.

-r *prefix_len*

The size of the system prefix space (default is 400 characters; maximum allowed is 2000).

-S *gdt_s ldt_s*

Set the maximum number of global segments and local segments in the system (default is 100 global segments and $6 \times \text{num_procs}$ local segments). Most programs need two local segments, while a few may need 30 or more. Global segments may be used by some programs for permanent shared memory.

-s *num_sessions*

The maximum number of sessions that will be supported (default is 32).

-T *trace_size*

The size of the trace buffer, in kilobytes (default is 2K). See also the **tracectrl**, **traceinfo**, and **tracelogger** utilities.

-t *num_timers*

The number of timers that will be supported (default is 40). These timers are allocated to processes with the *timer_create()* function and returned with the *timer_delete()* function. See also the *sleep()* and *alarm()* functions.

-v *num_meg*

(**Proc32** only) The number of megabytes of virtual memory space to allocate. The default is six times the amount of physical memory; the maximum is 250 megabytes.

-v *level*

Set the verbosity *level* (0 to 3; default is 1). Messages are always displayed on the console in the form:

QNX (*level*) : *message*

The messages should be interpreted as follows:

Level 0 : An internal inconsistency was detected. Contact QNX Software Systems.

Level 1 : The Kernel was forced to take a planned but uncommon recovery action.

Level 2 : Expected actions have occurred as a result of a misbehaving process or remote computer.

Level 3 : Status messages during normal operation.

Description: The Process Manager (**Proc**) works closely with the Kernel to provide essential operating system services. In fact, the Kernel is bundled with the code for **Proc** and shares the same address space. For these reasons, **Proc** is a required system process in all operating images that are built. You cannot start **Proc** after booting.

Most of the services **Proc** provides require the use of memory within **Proc**. The amount of memory used is controlled by tables whose sizes are set by options to **Proc**. Some tables are fixed in size, while others dynamically request more memory as needed. Unlike other system processes, **Proc** cannot grow its data segment at runtime. This means you must provide sufficient internal memory (referred to as heap space) when you build **Proc** into an OS image. This heap space is provided as a number in a *build_file* used by the **buildqnx** utility. To estimate **Proc**'s minimum heap space requirements, you may use this formula:

$$\begin{aligned}
 \text{heap} = & \text{num_major} \times 2 \\
 & + \text{num_shared_segs} \times 1 \\
 & + (\text{num_names} \times 8) + (\text{total length of registered names}) \\
 & + (\text{num_procs} \times \text{process_entry_size}^\dagger) + (\text{total length of unique} \\
 & \hspace{15em} \text{command names}) \\
 & + \text{each_code or data_segment} \times 4 \\
 & + \text{prefix_len} \\
 & + \text{num_sessions} \times 20 \\
 & + \text{num_timers} \times 24 \\
 & + 4000
 \end{aligned}$$

[†] where *process_entry_size* is 192 for **Proc32** and 148 for **Proc16**.

The number of code and data segments are determined by the programs you run. A small- or flat-model program requires one code and one data segment. While the system is running, you may use the **sin info** command to display the amount of free heap space available in **Proc**. The command displays three heaps:

This heap:	displays the heap space for:
Heapp	Proc (as described by the formula above)
Heapf	FDs (-f option)
Heapl	LDTs (-s option)

Proc

Process Manager (QNX)

See also: `buildqnx`, `sinit`, `tracectrl`, `traceinfo`, `tracelogger`

Watcom C Library Reference

Syntax: `qtalk [options] [system]`

Options: `-b baud`

Set the baud rate of the current device.

`-c hh`

Set this as the character that invokes `qtalk` commands (default is `^A`).

`-D delay`

Wait this many 1/20th second periods before running the command specified by the `-x` option.

The `qtalk` utility reads any data emitted by the modem during this delay period and displays the data on your screen before the command is started.

If you use `-x` without this option, `qtalk` won't wait to start the command. As a result, if the modem emits any information once `qtalk` has emitted the dialing string, you have no way of predicting whether `qtalk` or the command will see that information.

`-d hh`

Replace ASCII rubout with this character.

`-e` Enable local echo.

`-h` Hang up the current modem line if someone dials a new system from within `qtalk` via the `<Ctrl>-<A>` command. By default, the current modem device is closed but the line isn't dropped.

`-l logfile`

Log a recording of the `qtalk` session in *logfile*.

`-m modem[, init_string]`

The name of the device to use, or the name of a modem pool to choose a device from. If you specify multiple `-m` options, `qtalk` will try them, in order, until it finds a device that's not in use.

If you specify an *init_string*, it will be emitted to the modem before anything else.

If the modem is defined in the modem pool file `/etc/config/qtalk.modems`, the `init_string` may be overridden by an `init_string` defined in the entry for the selected modem. The `init_string` defined closest to the actual naming of the selected device always supersedes other `init_strings` (for more information, see the discussion on modem pools).

`-o protocol=command`

Redefine transfer-protocol options, where *protocol* is one of the following:

This protocol:	will do a:
<code>qcp_se</code>	qcp send
<code>qcp_re</code>	qcp receive
<code>zmodem_se</code>	zmodem send
<code>zmodem_re</code>	zmodem receive
<code>other_se</code>	<code>other</code> send
<code>other_re</code>	<code>other</code> receive

and where *command* is the command that will perform the file transfer. This command will be run by the Shell. The macro `$MODEM` will be set to the pathname of the modem device and, in the case of a send, the macro `$FILENAME` will be set to the filename to be transmitted.

You can disable `qtalk`'s automatic invocation of a protocol by setting *command* to a null string (`" "`). For example, the following would disable automatic zmodem receive:

```
qtalk -o zmodem_re=""
```

For information on default protocol commands or for more information on automatic invocation of protocols, see the section on “Invoking `qcp` and `zmodem` automatically.”

The `other` protocol lets you configure `qtalk` to use your own commands for sending and receiving files.

-P Ignore the 8th bit of received characters.

-s *system_directory*

Instead of **HOME/.qtalk**, use this file to look up systems to dial.

When diagnosing problems another user is having, you can specify **-s /home/user_id/.qtalk** to better emulate that user's environment.

-T *terminal_type*

Use this terminal emulation for screen output. The terminal type must be one of the following:

ansi
pcansi
qnx
qnx8

If you don't specify the **-T** option, **qtalk** will leave your console protocol as is.

See also the description of the **T** command in the Commands section.

-t *xfer_protocol*

Set the current transfer protocol to **qcp**, **zmodem**, or **other**. (To change the command string that will be run to perform the file transfer, use **-o**.)

-x *"command"*

Run this command after emitting the dialing string for the named system (see also **-D**). When the command is run, the **MODEM** environment variable is set to the pathname of the selected modem device.

system

The name of the system you want **qtalk** to call. The system must be defined either in **HOME/.qtalk** (or the file named by **-s**) or in the system-wide dialing directory, **/etc/config/qtalk**. For more information, see the description of the **d** (dial system) command.

talk over communications line (QNX)

talk over communications line (QNX)

Deprecated options:

-E Suppress FF codes (emulates **QNXS** terminals used in QNX 2).
Equivalent to **-T qnxs**.

Examples:

qtalk home Phone home.

qtalk quics Call the QNX update system.

qtalk -e -d 08 Communicate with a machine that doesn't echo (half duplex) and expects an ASCII **<backspace>** (08 hex) to delete characters.

qtalk -l /dev/par	Communicate with another system and print a hardcopy record.
--------------------------	--

qtalk -m //2/dev/ser2 Use the **/dev/ser2** serial port on node 2.

```
qtalk -o qcp_re="qcp $MODEM re -n"
```

Use the **qcp -n** option for **qcp** receives (i.e. receive only files that are newer than existing files).

Description:

The **qtalk** utility lets QNX users communicate with other computers via a serial line that's usually connected to a modem. The destination may be another host computer, in which case **qtalk** lets you use your computer as a terminal. The **qtalk** utility also lets two QNX users communicate and transfer files.

The **gtalk** utility sends any characters you type on the keyboard to the other system via the modem. Any characters received by the modem are displayed. In local echo mode, typed characters are echoed on the display as well as being sent over the modem.

Defaults

The **qtalk** utility lets you to define new defaults for anything that can be specified by command-line options. You define these defaults as a system named **defaults**. The **qtalk** utility will look for **defaults** first in **\$HOME/.qtalk** and, if not found there, then in **/etc/config/qtalk**.

Command-line options defined in the **defaults** system are applied *before* the options specified in the actual command line. Thus you can override the settings in your **defaults** system with command-line options.



The **-s** command-line option doesn't change how **qtalk** searches for the **defaults** system.

Logging a session

You can log a recording of a **qtalk** session with the **-l** option:

```
qtalk -l /dev/par
```

OR

```
qtalk -l /tmp/logfile
```

Using the command character

You can use a special command character to set special modes and options while you're within the **qtalk** environment. This special character defaults to Control A (^A) unless you change it with the **-c** option when you invoke **qtalk**. When you enter the command character, you will be shown some current settings and will be prompted to enter a command. If you enter the command character twice, a single command character is echoed to the modem. This allows you to send the command character to the remote system if you need to.

Replacing the rubout/delete character

You'll find the **-d hh** option useful when communicating with computers that have a rubout character different from the one you're used to. Many systems use the <Backspace> key (08 hex) to erase a character. QNX systems default to the ASCII rubout character (7F hex). If you type:

```
qtalk -d 08
```

qtalk translates your <Rubout> key into backspace automatically.

Enabling flow control

Using very high-speed modems, or producing a log on slow printers or floppy disks, may cause some characters to be lost. To prevent characters from being lost in these cases, you can enable input flow control prior to invoking **qtalk** (see **stty**). This works only if the machine sending the data supports flow control of its output.

Transferring files

The **qtalk** utility lets you transfer files through either of two methods. You invoke the simpler method with the **write** and **log** commands to transfer text files to or from another system. No error checking is performed, however, so you should use this method only when communication lines are good (i.e. direct connect lines or reliable modem connections).

The second method, involving **send** and **receive** commands, uses a more sophisticated protocol that includes error checking and retransmission when transferring files to other systems.

To send a QNX file to a host using **write**, you should first make the host capable of receiving a stream of text. You could do this with an editor or the **cat** utility. You can then use the **write** command to send the file.

Invoking **qcp** and **zmodem** automatically

The **qtalk** utility attempts to detect **qcp** and **zmodem** file transfers when **qtalk** is on the receiving end. If it detects one of these, **qtalk** automatically invokes the appropriate receive command for the protocols being used. You can disable this behavior; see the **-o** option.

Here are the default protocol commands:

```
qcp_re="qcp $MODEM re"
qcp_se="qcp $MODEM se $FILENAME"
zmodem_re="rz <$MODEM >$MODEM"
zmodem_se="sz $FILENAME <$MODEM >$MODEM"
other_re=""
other_se=""
```

You can't automatically enable the **other_re** protocol. To invoke **other_re** or **other_se**, choose **send** and **receive** from the command menu when the current file transfer protocol has been set to **other**.

Modem pools:

The **qtalk** utility provides a modem pool facility that serves as a shorthand for specifying multiple modems (otherwise done by specifying multiple **-m** options). For example, instead having to type:

```
qtalk -m //1/dev/ser1 -m //2/dev/ser1
      -m //2/dev/ser2 -m //30/dev/ser2
```

you could simply type:

```
qtalk -m hst
```

if the modem pool **hst** has been defined to include those four modems.

The **qtalk** utility tries each device in each pool until it finds a modem that's unused (i.e. no other process has the modem open). When it finds an unused modem, **qtalk** will do the following, in order:

- 1) emit an *init_string* (if there is one) to the modem
- 2) emit a dial string (if there is one)
- 3) run the command specified by **-x** (if **-x** was specified)
- 4) put you in an interactive session with the modem

Defining a modem pool

You define modem pools in the **/etc/config/qtalk.modems** file. Here's the format of this file:

```
poolname modem[,init_string]...
[<whitespace>modem[,init_string]]...
.
.
```

where:

- poolname* An arbitrary name that identifies a group of modem definitions.
- modem* Either the pathname of a modem device (e.g. **//3/dev/ser2**) or the name of another modem pool (e.g. **hst**). A *modem* definition can also include an optional initialization string (*init_string*).

If *modem* is a modem device (e.g. `//3/dev/ser2`) and the entry for that device contains an *init_string*, **qtalk** will, upon selecting that device, emit the string to the device and then emit any other communication to the device.

If *modem* is a modem pool (e.g. **hst**), **qtalk** will use the first available modem device in that pool. If the selected modem entry contains an *init_string*, that string will be emitted. If the selected entry doesn't contain an *init_string*, but the line that references the modem pool does contain an *init_string*, that string will be emitted.

You can specify multiple modem definitions, separated by whitespace. You may continue onto the next line by starting it with one or more `<space>` or `<tab>` characters. If the next line doesn't begin with whitespace, it will be regarded as a new pool definition, not a continuation of the previous one.



If you specify an *init_string*, you can't insert whitespace between the modem, the comma, and the *init_string*. Also, if the *init_string* contains any `<space>`s or `<tab>`s, you must enclose it in quotes.

Example entry

Here's an example entry in the `/etc/config/qtalk.modems` file:

```
hst //1/dev/ser1 //2/dev/ser1 //2/dev/ser2
    //30/dev/ser2, ""

resethst hst, "^~~~ATZ"
```

This example defines two pools: one (**hst**) a simple pool of modems; the other (**resethst**) a pool that has the same modems but with an *init_string* of `"^~~~ATZ"`. Note that one modem in the **hst** pool, `//30/dev/ser2`, has an empty *init_string* (i.e. `""`).

Any modems accessed through the **hst** pool won't have an *init_string* emitted to the device. But if those same modems are accessed through a reference to the **resethst** pool, all but `//30/dev/ser2` will have the `"^~~~ATZ"` *init_string* emitted to the device.

The presence of the empty *init_string* (`""`) ensures that no *init_string* will be sent to `//30/dev/ser2` regardless of whether the modem

was accessed via the **hst** pool or the **resethst** pool. This happens because the *init_string* closest in the pool hierarchy to the actual naming of the device always supersedes any other *init_strings*.

As an alternative to defining the **resethst** pool, you could either name the "**^~~~ATZ**" *init_string* for each modem in the **hst** pool, or you could specify the *init_string* with the **-m** option (**-m hst, "^^^^ATZ"** does the same as **-m resethst**).

In *init_strings* (as well as dialing strings, described below), you can specify any of the following characters; **qtalk** will act on these characters instead of sending them straight to the modem:

- | send a carriage return
- ~ delay for 1 second
- ' delay for 100 milliseconds
- ^ drop DTR for 1 second (forces modem to hang up and reset)
- ! send a 500-millisecond break
- \o emit a single character represented by *o*, where *o* is an octal number of up to three digits
- \xhh same as above, but specified in hex

Commands: To specify any of the following commands, you first press the command character (usually <Ctrl>-<A>).

- b** (break) Send a break over the modem. You may also send out breaks by pressing the break key (<Ctrl>-<Break> on the console keyboard).
- C** (command character) Change the command character. You will be prompted to enter the new command character in hex (e.g. 0x02) or as *^char* (e.g. **^b**).
- c** (change directory) If you specify the **c** command, **qtalk** prompts you for a new directory name, and will attempt to move to that directory. If **qtalk** moves successfully to the directory you specify, the directory becomes the current working directory (see the **pwd** utility) for the duration of **qtalk**, or until you change directory again. When **qtalk** terminates,

your current working directory reverts to the directory you were in when **qtalk** was invoked.

- d** (dial system) If you specify the **d** command, **qtalk** prompts you to enter a system name. It will first look for that name in your own dialing directory (**\$HOME/.qtalk**) and if it doesn't find it there, it will look in the system-wide dialing directory, **/etc/config/qtalk**.

If you enter a question mark (?) for the system name, you'll see the contents of both dialing directories (if they exist) and you'll be prompted again for a system name. To abort the dialing command and return to normal communications mode, press <Enter> without entering a name.

Dialing is implemented by looking up the system name—which can also be specified on the command line when **qtalk** is invoked—first in your **\$HOME/.qtalk** file, then in the **/etc/config/qtalk** file. These files share the same format:

```
system_name [dialing_string]
[<whitespace> command-line_options]
```

In these files, you must specify the system name at the very left of the line with no whitespace before it. The dialing string is optional (this string contains commands to be sent to the modem before you're given interactive control). If you specify a dialing string, you must separate it from the system name by <space>s or <tab>s, or both.

In dialing strings, you can specify any of the following characters; **qtalk** will act on these characters instead of sending them straight to the modem:

	send a carriage return
~	delay for 1 second
'	delay for 100 milliseconds
^	drop DTR for 1 second (forces modem to hang up and reset)
!	end a 500-millisecond break
\o	emit a single character represented by o, where o is an octal number of up to three digits
\xhh	same as above, but specified in hex

Additional lines beneath the line that defines the system name and dialing string may contain additional **qtalk** command-line options to apply when talking to this system.

If you specify these additional lines, begin each one with at least one **<tab>** or **<space>** character. Note that a single command-line option can't span multiple lines. You can, however, place just one option per line.

Here's an example of a dialing directory entry that instructs a Hayes-compatible modem to call the QNX Update System (QUICS):

```
quics ATDT16135910934
```

The following is a more elaborate example that enforces a QNX terminal type and enables the QNXS terminal-type emulation. It will use either modem **/dev/ser1** or modem **//2/dev/ser1**, whichever is currently not in use:

```
quics ATDT16135910934  
    -T qnxs -m /dev/ser1  
    -m //2/dev/ser1
```

- | | |
|-----------------------------|--|
| D (delete character) | Change the delete character. You will be prompted to enter the new delete character in hex (e.g. 0x08) or as <i>^char</i> (e.g. <i>^h</i>). |
| e (echo) | If you specify the e command, the local echo feature is toggled. Some systems expect the "terminal" to perform local echoing (half duplex). |
| h (hang up) | If you specify the h command, the CTS/RTS lines are lowered for approximately 1/2 a second. This permits modems that support hardware hangup to do so. |
| l (log) | Begin or end logging of this session. If no log file is open, qtalk asks for the name of a file to log into. If logging is already in progress, it is terminated and the log file is closed. The l command records every character that is sent or received in the log file. |

You can use **1** to take “snapshots” of data from a host computer at slow speed.

- o (modify protocol options) Prompts you for the protocol whose command string you wish to change (**qcp**, **zmodem**, **other**). The current send and receive command strings are displayed and you’re then prompted to enter a new command string. To set a string to null, use **""**. If you don’t want to make any changes, press **<Enter>**.
- p** (parity) Ignore parity (top bit) of received characters. If this option is already set, turn it off.
- q** (quit with hangup) This executes a hangup command (see **h** above) then causes **qtalk** to exit.
- r** (invoke a receive) Invoke the currently selected file transfer protocol to receive a file. Note that **qtalk** will automatically invoke **qcp** and **zmodem** if it recognizes a received startup sequence of one of these protocols.
- s** (send a file) Send a file, using the currently selected file transfer protocol. This sends a file to another system running the same protocol, which is more secure than simply writing the file to the modem (see the **write** command).

When the currently selected protocol is **qcp**, you can send more than one file by specifying the **x=index_file** option when **qtalk** asks for the file to send. This file will contain a list of files to send, one per line. You can also specify more than one filename, separated by spaces.

The **qcp** utility lets you follow the name of the file to send with the name of the destination file. You separate the two filenames with a comma. This is also true for filenames within an index file (option **x=**). For example:

```
Send file(s)?    file1 main.c,new_main.c
```

sends the file **file1** as **file1** and the file **main.c** with the name **new_main.c**. If you don't specify a new name, **qtalk** creates a file with the same name as the file that is sent.

Files received by **qtalk** using the **qcp** protocol have the same attributes and date as the file on the sending machine.

Most transfer protocols require that the modem port be configured for 8-bit data (see the **stty** command).

T (switch terminal emulation)



The **T** command and the **-T** option require that the device **qtalk** is running on supports the specified protocol.

The **T** command lets you switch the terminal emulation used by the console that **qtalk** is running on. You can specify one of the following terminal types:

This type:	is useful for:
ansi	communicating with systems that support ANSI or vt100 type terminals. It uses ISO Latin-1 supplemental characters (default mode of Dev.ansi console driver).
pcansi	communicating with a DOS BBS. This type uses the IBM PC supplemental character set.
qnx	communicating between QNX systems.
qnxs	communicating with QNX 2 systems that have been set up to run with a qnxs terminal.

When you use the **T** command, you'll be prompted for a terminal type. If you don't wish to set or change the terminal type, simply press <Enter>.

t (select transfer protocol)

Shows you which protocol is being used (e.g. **qcp**, **zmodem**) and asks you to select a new one. If you press <Enter>, no changes are

qtalk

talk over communications line (QNX)

made. This command controls the protocol used when you use the **send** and **receive** commands from the command menu.

w (write) Write a file to the modem. The file is transmitted over the modem, and also echoed on the display if local echo is enabled. The **write** command doesn't perform any error checking.

You may abort writing to the modem at any time by pressing any key.

x (exit) Exit from **qtalk** without performing a hangup. It's good practice to use the **q** (quit with hangup) command for leaving **qtalk**, unless you really don't want to perform a hangup.

! (execute a shell command) This command lets you execute any command from within **qtalk**. You'll probably find that you'll use the **!** command often to:

- execute the **ls** command in order to see what files are in your current directory prior to sending some of them
- execute non-native file transfer protocols (such as Xmodem or Kermit) from within **qtalk** if you don't want to bother setting the **other** protocol to the desired commands



The **zmodem** utilities **rz** and **sz** (which also implement XMODEM and YMODEM) can be downloaded from the QNX update system.

See also: **Dev.ansi, qcp, stty**

Syntax: **Slib16**
 Slib32

Example: The following build file would be for a node using an Ethernet 1000 card, where only 16-bit programs could run:

```
sys/Proc16
$ 42000 Proc16 -1 $(lnode)

sys/Slib16
$ 1 Slib16

/bin/Net
$ 16000 Net -m $(netmap)

/bin/Net.ether1000
$ 20000 Net.ether1000

/bin/sinit
$ 1000 sinit -r //$(bnode)  TERM=qnx  TZ=$(TZ)
```

Same as above but for a node where both 16- and 32-bit programs could run:

```
sys/Proc32
$ 42000 Proc32 -1 $(lnode)

sys/Slib16
$ 1 Slib16

sys/Slib32
$ 1 Slib32

/bin/Net
$ 16000 Net -m $(netmap)

/bin/Net.ether1000
$ 20000 Net.ether1000

/bin/sinit
$ 1000 sinit -r //$(bnode)  TERM=qnx  TZ=$(TZ)
```

Description: For QNX to operate, a system shared library must be present in the operating system image that is booted. There are two of these libraries:

- **Slib16**—allows 16-bit programs to run. This library is mandatory. It must be specified in the build file for all operating system images.
- **Slib32**—allows 32-bit programs to run. This library is optional. It can be started from the build file, the system initialization file,

or the command line. You need it only if you want to run 32-bit programs.

If you want to run both 16- and 32-bit programs on your system, you must run both shared libraries.



If 32-bit programs are run without the 32-bit shared library (**slib32**), they will fail.

See also:

Proc

“Building a custom operating system image” in *User’s Guide*

Syntax: `tracectrl [-i] [-n node] [-p percent_full] [-s severity]`

Options:

- `-i` Display current trace parameters.
- `-n node` Query the specified node (default is this node).
- `-p percent_full`
Set the high water mark which triggers the logging process (e.g. `tracelogger`). If you specify a *percent_full* of 0, a trigger will occur on every trace event. For more information, see `tracelogger`.
- `-s severity` Tell `Proc` to save only those trace events that meet or exceed this severity level. You can specify a *severity* argument from 0 (most critical) to 7 (least critical).

Examples: Save events of severity 0 to 3 and trigger the logging process (e.g. `tracelogger`) when the trace buffer is 60% full:

```
tracectrl -s 3 -p 60
```

Description: The `tracectrl` utility lets you display or change trace parameters.

Trace events are stored in chronological order in the Process Manager's in-memory trace buffer. The following parameters, which `tracectrl` displays when invoked with the `-i` option, reflect the Process Manager's current trace capabilities:

Overruns

The number of overruns that have occurred. An overrun occurs if the trace buffer has become completely full and another trace event is added to the buffer. Adding a trace event to the full buffer causes the oldest event to be discarded.

If overruns are occurring, you should do either or both of the following:

- increase the size of the trace buffer by specifying a larger `-T` option to `Proc` (see the documentation for `Proc`)
- consider using `tracelogger` to automatically log the trace buffer as trace events occur

tracectrl

display and change trace parameters (QNX)

Severity

A severity level. The buffer stores any events that meet or exceed this level (0 is most critical, 7 is the least).

Reader

The process ID of the logging process currently extracting events from the trace buffer. For example, if **tracelogger** is running, this value will be **tracelogger**'s *pid*.

BufferSize

The size, in bytes, of the trace buffer. To set this size, use **Proc**'s **-T** option.

DataSize

A running total of the number of bytes of data in the trace buffer.

Trace Selector

The memory selector of the trace buffer; **Proc** owns this selector.

High Water

The number of bytes the trace buffer must contain before the logging process is triggered. This number reflects the value entered with the **-p** option on **tracelogger** or **tracectrl**.

Proxy

The proxy to trigger when the trace buffer contains the number of bytes indicated by **High Water**. This proxy is set up between **Proc** and the logging process when the logging process is first run. The value has meaning only if there is a reader.

Exit Status: 0 Successful completion.
 >0 An error occurred.

See also: **Proc**, **traceinfo**, **tracelogger**, *Trace...()* and *qnx_trace...()*
 functions in the *Watcom C Library Reference*

Syntax: `traceinfo [options] [logfile...]`

- Options:**
- c** Clear `Proc`'s trace buffer after a successful read. Trace events are cleared only if this option is specified. To use this option, you must have superuser privileges. If you specify a *logfile*, this option is ignored.
 - d** *start_date* [: *end_date*]
Print trace events that occurred between *start_date* and *end_date*. If *end_date* isn't given, `traceinfo` uses the current date and time as the *end_date*.
The date format is[[[yy]mm]dd]HH/MM. The utility assumes the current date and time for any omitted fields.
 - e** *format_file*
Read additional format information from this file. The utility always reads the `/etc/config/traceinfo` file, which contains the trace format strings for the system processes (`Fsys`, `Proc`, `Net`, ...). For more info, see the section on "Trace format files."
 - F** *date_format_string*
Use this format string for printing the date (default is "%b %d %T"). This option overrides the (`date`) print variable in the trace format file. For more info, see the section on "Trace print variables."
 - H** *print_format_string*
Use this format string when printing trace events. This option overrides the (`header`) print variable in the trace format file. For more info, see the section on "Trace print variables."
 - M** [!] *major* [: *minor*]
Print events whose major- and minor-code numbers match these values. If you don't specify a minor code, all trace events matching the major code are printed.
If you precede the major code with a ! symbol, events with that code aren't printed.

traceinfo

print trace events (QNX)

Values may be decimal or hexadecimal. Using multiple **-M** options, you can specify up to 10 matches and 10 exclusions.

-m Print the millisecond timestamp after the date, according to the (**milli**) format. For more info, see the section on “Trace print variables.”

-n *node*

Get the trace events from this node (default is node on which **traceinfo** is run). This option applies only if you don't specify a logfile (i.e. trace events are being read from **Proc**'s event buffer or from standard input).

-r Print raw event data in hex.

-s [*severity1* :] *severity2*

Print trace events whose severity level falls in the range of *severity1* to *severity2* (default is 0 : 7).

logfile

A file containing trace events. See the **tracelogger** utility. If *logfile* is **-**, **traceinfo** reads events from standard input.

If *logfile* isn't specified, **traceinfo** reads directly from **Proc**'s trace buffer.

Examples:

Print trace events that have occurred since 1:00 pm today:

```
traceinfo -d 1300
```

Print trace events that have occurred between August 12 00:00 and August 13 00:00 of the current year:

```
traceinfo -d 08120000:08130000
```

Print trace events, using the given date format string:

```
traceinfo -F "%Y,%m,%d,%T"
```

Print trace events, using the given header format string:

```
traceinfo -H "(severity) (date) (tracestr)"
```

Print all events except for those with major code 4:

```
traceinfo -M !4
```

Print events with major code 2 and minor code 3, as well as those with major code 6:

```
traceinfo -M 2:0x03 -M 6
```

Similar to above, except that events with major code 6 and minor code 2 aren't printed:

```
traceinfo -M 2:0x03 -M 6 -M !6:2
```

Print events with severity 0 to 4:

```
traceinfo -s 4
```

Print events with severity 3:

```
traceinfo -s 3:3
```

Print events with severity 2 to 4:

```
traceinfo -s 2:4
```

Description:

The **traceinfo** utility prints trace events in human-readable format. If you don't specify a logfile, **traceinfo** reads events directly from the trace buffer maintained by the Process Manager (**Proc**). If you specify one or more logfiles, **traceinfo** reads them instead of **Proc**'s trace buffer.

Events are printed one-per-line according to the format specified in the trace format file. The default format file is `/etc/config/traceinfo`. You can have **traceinfo** read additional format files by specifying the `-e` option. For more information, see the section on "Trace format files."

Trace events are binary data consisting of a *trace code* and its associated *event data*. Every trace code is made up of a 20-bit major code and a 12-bit minor code. The content of the data portion of a trace event is determined by the process that generated the event. The **traceinfo** utility simply formats and prints the event data

traceinfo

print trace events (QNX)

according to print format descriptions. For example, **Proc** uses the trace code 0x00001001 for a “Bad Status” event:

Trace code	Event data
0x00001001	03 2E 00 01 00 00 20 20 34 FF
Major code	
Minor code	

Event data is a stream of bytes; in most cases the data is a series of integers. When a trace event is generated, **Proc** records whether the event was created by a 16- or 32-bit program. **Proc** does this because the number of bytes that constitute an integer affects how **traceinfo** processes the event data, and consequently, may affect your format specifications (integers are 2-byte items for 16-bit programs and 4-byte items for 32-bit programs).

When **traceinfo** formats the event data, it maintains a running pointer that always points to the next integer to be processed. This pointer is incremented by either 2 or 4, depending on whether the trace event was made by a 16- or 32-bit program. For example:

16-bit program	0	1	2	3	4	5	
	032E	0001	0000	2020	34FF	22ae	Event data
32-bit program	0	1	2				

If the above event data were made by a 16-bit program, you could print the data with the following format specification:

%d %d %d %d %d %d Print the decimal equivalent of all 6 integers. Each integer is 2 bytes. Data pointer increments by 2 each time.

If the above event data were made by a 32-bit program, you could print the data with the following format specification:

%d %d %d

Print the decimal equivalent of all 3 integers. Each integer is 4 bytes. Data pointer increments by 4 each time.

In a format string you must sometimes explicitly refer to a specific integer without using `traceinfo`'s running data pointer. To do this, you specify an *index* in the format string. An index is an integer-based offset from the beginning of the event data. Because this offset is integer-based, it will increment in the same manner as `traceinfo`'s pointer index (by either 2 or 4 bytes, depending on the program that created the event). The index starts at 0, so an index of 0 refers to the first item. For more information, see the section on "Format Specifications."

Trace format files:

Trace format files describe how to print trace events by means of a *printf()*-style format specification. The structure of these files is based on special characters and command keywords.

Lines that begin with:	indicate:
#	a <i>command keyword</i> entry
<tab>	a <i>minor-code format string</i>
any other characters	the line is to be ignored

Command keywords

You can specify the following command keywords:

#trace

Define a trace *print variable*. For example:

#trace date "%b %d %y" /* use Month Day Year */

For more info, see the section on "Trace print variables."

traceinfo

print trace events (QNX)

#major

Define a new major-code header. For example:

```
#major 0x00001 /* Process Manager */
```

All subsequent minor trace format strings refer to this major code. For more info, see “Major-code header” and “Minor-code format strings.”

#table

Define a new major table header. For example:

```
#table 0x00045 /* Strings for major event 0x00045 */
```

For more information, see “Tables.”

#include

Load another trace format file. For example:

```
#include /etc/config/screentrace
```

The specified file is opened and read as though its contents were part of the original trace format file. You can use up to five levels of nesting. For more info, see “Include files.”

Trace print variables

The **traceinfo** utility has *print variables* that represent information **Proc** records internally with each event. (In the case of the event date, both a full date and a *millisecond time stamp* are recorded.) By modifying the format specification, you can control the format **traceinfo** uses when printing this information.

Note that two of the print variables aren’t based on **Proc**’s internal event information:

- **(header)**---describes which of the other variables are printed and in what order.
- **(tracestr)**—the content of this variable varies depending on the event being processed. It always contains the minor-code format string for the current event being processed by **traceinfo**.

To specify print variables, use this syntax:

```
#trace print_variable format_spec
```

You can specify the following print variables:

(date)

Use this format when printing the trace event date. This variable is overridden by the **-F** option.

String	Description	Example
%a	abbreviated weekday name	Wed
%A	full weekday name	Wednesday
%b	abbreviated month name	Dec
%B	full month name	December
%d	day of month (01-31)	02
%D	format <i>mm/dd/yy</i>	12/31/92
%H	hour as 24-hour clock	19
%j	day of year (001-366)	234
%m	month (01-12)	04
%T	24-hour clock <i>HH:MM:SS</i>	23:45:12
%r	12-hour <i>HH:MM:SS</i> (am/pm)	09:45:12 PM
%y	year (00-99)	92
%Y	year with century	1992

See also the *strftime()* function in the *Watcom C Library Reference*.

(header)

Use this string to determine how other print variables are printed and in what order. This variable is overridden by the **-H** option.

Each print variable is referenced in parentheses. For example:

- Print the date of the event and the formatted event data:
(date) (tracestr)
- Print major code and severity:
The major code=(major), the severity=(severity)

This would produce output similar to:

The major code=00003, the severity=2

(major)

Use this format when printing the major code of the trace event.

traceinfo

print trace events (QNX)

(**milli**)

Use this format when printing the millisecond time stamp of the event.

(**minor**)

Use this format when printing the minor code of the trace event.

(**severity**)

Use this format when printing the severity of the trace event.

(**tracestr**)

Use the minor-code format string for the current trace event. For more info, see “Minor-code format strings.”

Major-code header

The command keyword **#major** defines a major-code specification that all subsequent minor-code format strings are related to.

The **#major** keyword has this syntax:

```
#major major_code
```

The *major_code* may be decimal or hexadecimal. If you specify it in hexadecimal, precede the number with a 0x (e.g. **0x00001**). Anything else on the line is discarded as a comment.

The following example would start a new major-code specification for major-code 16:

```
#major 16 /* start of my new major code section */
```

Minor-code format strings

These format strings follow a major-code header. They describe how to format the data for each of the minor-code events that have the preceding major code.

To specify minor-code format strings, use this syntax:

```
<tab>minor_code format_spec...
```

Each line must begin with a **<tab>**, followed by a minor code and then a format specification. Numbers may be decimal or hexadecimal. The *format_spec* argument is a *printf()*-style string

(with some extensions). For more information, see “Format specifications.”

In the following example, a major-code header indicates that the subsequent minor-code format strings are for the Process Manager. The example contains two minor-code format strings (for the minor codes 10 and 12):

```
#major 0x00001      /* Process Manager */
<tab>10 Session Leader died (%s)
<tab>12 Dead loader 1 (%s)
```

Tables

Tables let you use the currently pointed-to event data as an index into a table of fixed constant strings. You can use this mechanism as an alternative to using minor trace codes to differentiate the various events that have the same major code. Also, if you use tables, programs don’t need to put strings in their trace events, thus reducing the amount of data that must be stored in the trace buffer.

Note that table strings aren’t interpreted as format strings; they’re printed literally.

Each table begins with the command keyword **#table**, followed by whitespace and then a major code (which may be decimal or hexadecimal):

```
#table major_code
```

Here’s an example:

```
#table 0x00023      /* table header for major code 0x23 */
```

Following the above keyword line, one or more table entries specify literal strings for events that can occur for the specified *major_code*. To specify these entries, use this syntax:

```
<tab>index literal_string
```

Each table entry uses this syntax, which consists of a **<tab>** character followed by an index number, whitespace, and then a constant string.

traceinfo

print trace events (QNX)

Here's an example table entry:

```
#table 0x00023 /* table entries for major code 35 (0x23) */
    0   Data buffer OK.
    1   Data input buffer has overflowed!
    2   Data output buffer has overflowed!
    3   Error! Loss of carrier!
   100  Unable to continue!

#major 0x00023
    1   Error message: %(table)
```

As this example shows, you need only one minor-code format string to use the table facility.

Include files

You can include another trace format file to be read. The specified file is opened and read as if its contents were part of the original trace format file. You can have up to 5 levels of nesting.

To specify an include file, use this syntax:

```
#include filename
```

For example:

```
#include /etc/config/screentrace
```

Format specifications:

Each format specification begins with the % character; the basic layout is as follows:

```
%[@index, ][-][0|<space>][min][.max][S|L][format_character|format_keyword]
```

where:

@index,

A decimal number indicating the integer offset from the start of the event data. The index starts at 0; thus an index of 0 refers to the first integer. The index is interpreted as either a 2-byte or 4-byte integer, depending on whether the process that created the event is 16-bit or 32-bit.

- Left justify the output.

0 or <space>

A padding character. Use a 0 to pad with zeros; use a space to pad with spaces.

[min]

A decimal integer indicating the minimum field width of the printed data.

[.max]

A decimal integer indicating the precision of the printed data. This must be preceded by a period.

[S|L]

A short or long indicator that will override the default integer size used when the event data is interpreted.

The default size is determined by whether the trace event was created by a 16- or 32-bit program:

S = Short (2-byte integers)

L = Long (4-byte integers)

While it affects the interpretation of formats such as %d and %x, this option does *not* affect the running pointer.

traceinfo

print trace events (QNX)

printf()-style format
characters

The following format characters are supported:

Character	Description
d	decimal
u	unsigned
x	hexadecimal
b	binary
o	octal
c	char (1 byte)
s	string
	If used, this character must be the last field specifier in the format string. It causes any unprocessed event data to be output in ASCII without recognizing a NUL character as a string terminator.
e	Skip (eat) the data being pointed to. Don't print. You can use this option to move the running data pointer without printing anything.

Format keywords

Format keywords specify common ways to interpret the event data currently being pointed to. Each format keyword must be enclosed in parentheses.

(pidname) Assume the data pointer is pointing to a process ID (PID). If possible, print the process name associated with the PID.

If the PID no longer exists, or the start time of the process is later than the trace-event time, then **(pidname)** is expanded to **pid=number** or **vid=number**.

If the process name is too large to fit in the specified length, the process name starts with a * and is right justified so that the characters furthest to the right in the process name are printed.

(strerr) Assume the data pointer is pointing to an *errno* value and print that *errno*'s associated error string (see */usr/include/errno.h*).

- (errno)** Assume the data pointer is pointing to an *errno* value and print that *errno*'s manifest as a string—ENOEXEC, for example (see `/usr/include/errno.h`).
- (table)** Print a table string.
With this keyword, **traceinfo** looks for a *table* entry that matches the major code of the current event. If a valid table entry exists, **traceinfo** uses the value pointed to by the current data pointer as an index into that table and prints the associated literal string.
For more information, see the section on “Tables.”
- (date)** Assume the data pointer is pointing to a date and print the date as the output of the *ctime()* function.
- (minor)** Print the minor trace code associated with this trace event. The code is printed according to the format of the **(minor)** print variable.
- (major)** Print the major trace code associated with this trace event. The code is printed according to the format of the **(major)** print variable.
- (pnet)** Assume the data pointer is pointing to a physical network ID that's six bytes in length.
If the second two words of the network ID are zeros, **traceinfo** assumes the ID is an Arcnet address and prints only the first word. Otherwise the network ID is printed as a six-byte hex number.

traceinfo

print trace events (QNX)

Formatting examples:

The manifests used in the following examples are defined in `/usr/include/sys/trace.h` and `/usr/include/sys/tracecod.h`.

A simple example

For the trace C code:

```
Trace2(_TRACE_TEMPORARY | 1, _TRACE_EXPECTED, 0x123, 7523);
```

the corresponding format line could be:

```
#major 0xFFFFF    /* temporary */
1  Trace type 1 numbers are 0x%x, %d
```

and would produce:

```
Jan 19 11:34:02 Trace type 1 numbers are 0x123, 7523
```

A more advanced example

Let's now look at several format lines for the following trace C code:

```
#define _TRACE_DEMO    _TRACE_TEMPORARY
/*
 * This is _TRACE_TEMPORARY until QNX Software Systems assigns
 * a trace code to your company
 */

func(pid_t sender, msg_t type) {
    msg_t      status;

    if(Send(sender, &type, sizeof(type), &status, sizeof(status)) == -1) {
        Trace(_TRACE_DEMO | 2, _TRACE_EXPECTED, type, sender, errno);
    } else if(status != EOK) {
        Trace(_TRACE_DEMO | 3, _TRACE_EXPECTED, type, sender, status);
    }
}
```

Simple trace-format event line

Here's a simple trace-format event line for this example:

```
#major 0xFFFFF    /* temporary */
2  Send of message %d to pid %d failed with errno %d
3  Reply of message %d from pid %d failed with a status %d
```

Example output:

```
Jan 19 11:34:02 Reply of message 291 from pid 5678 failed
with a status 22
```

A more advanced
example (cont'd)

Using printf()-like formats

You can also use *printf()*-like formats:

```
2 Send of message %04.4x to pid %d failed with errno %d
3 Reply of message %04.4x from pid %d failed with a status %d
```

Example output:

```
Jan 19 11:34:02 Reply of message 0123 from pid 5678 failed
with a status 22
```

*Expanding numbers to
common formats*

You can use format keywords to expand numbers to common
formats:

```
2 Send of message %04.4x to %(pidname) failed with %(errno)
3 Reply of message %04.4x from %(pidname) failed with a
status %(errno)
```

Example output:

```
Jan 19 11:34:02 Reply of message 0123 from /home/peterv/test
failed with a status EINVAL
```

*Printing data that's not
referenced by the
current data pointer*

Using an index override, you can print data that the current data
pointer no longer references:

```
2 MSG=%04.4x Send failed with %@2, (errno) (%@2,d) from
%@1, (pidname)
3 MSG=%04.4x replied with %@2, (errno) (%@2,d) from
%@1, (pidname)
```

Example output:

```
Jan 19 11:34:02 MSG=0123 replied with EINVAL(22) from /bin/test
```

Printing errno

If you also want the *errno* string printed, you could do the following:

```
2 MSG=%04.4x to %-15.15(pidname) Send %-20.20(strerror)
(%@2, (errno))
3 MSG=%04.4x to %-15.15(pidname) Reply %-20.20(strerror)
(%@2, (errno))
```

Example output:

```
Jan 19 11:34:02 MSG=0123 to /bin/test Reply Invalid argument
(EINVAL)
```

traceinfo

print trace events (QNX)

Passing a “long”
integer from a
16-bit program

For the trace C code:

```
long a = 0x12345678;
Trace2(_TRACE_TEMPORARY | 4, _TRACE_EXPECTED, a >> 16,
      a & 0xFFFF);
```

the corresponding line could be:

```
4 Trace type 4 long number is %Ld
```

Example output:

```
Jan 19 11:34:02 Trace type 4 long number is 305419896
```

Advanced examples

The *Trace...()* functions always pass a native “int.” If you wish to create trace events with the same information for both 16- and 32-bit programs, you may have to use overrides when printing the data. Note that you can avoid this by using different minor codes for the 16-bit and 32-bit data.

Passing a “long”

Let’s now look at passing a long integer from a 16- or 32-bit program and interpreting this event data as a long integer.

For the trace C code:

```
long a = 0x12345678;
#ifdef __386__
Trace2(_TRACE_TEMPORARY | 5, _TRACE_EXPECTED, a);
#else
Trace2(_TRACE_TEMPORARY | 5, _TRACE_EXPECTED, a >> 16, a &
      0xFFFF);
#endif
```

the corresponding line could be:

```
5 Trace type 5 long number is %Ld
```

Example output:

```
Jan 19 11:34:02 Trace type 5 long number is 305419896
```

Passing a “short”

Let’s now look at passing a *short* integer from a 16- or 32-bit program and interpreting this event data as a short integer.

For the trace C code:

```
short a = 1234, b = 6789;
#ifdef __386__
Trace2(_TRACE_TEMPORARY | 6, _TRACE_EXPECTED, a | b << 16);
#else
Trace2(_TRACE_TEMPORARY | 6, _TRACE_EXPECTED, a, b);
#endif
```

the corresponding line could be:

```
6 Trace type 6 short numbers are %0,Sd and %1,Sd
```

Example output:

```
Jan 19 11:34:02 Trace type 6 short numbers are 1234 and 6789
```

A complex example

The following 16-bit example generates a trace event that describes when the current machine was booted:

```
#include <stdio.h>
#include <sys/osinfo.h>
#include <sys/psinfo.h>
#include <sys/kernel.h>
#include <sys/trace.h>

char buffer[7];
struct _osinfo osdata;

#define MY_TRACE_MAJOR 0x00044 /* major trace code assigned
    by QNX Software Systems Ltd. */

main()
{
    pid_t id;
    struct _psinfo data;
    short int tmp;

    if(qnx_osinfo(getnid(), &osdata) == -1) {
        exit(-1);
    }
    id = qnx_psinfo( 0, 1, &data, 0, 0 ); /* get psinfo for Proc */
    sprintf(buffer, "%s",
        osdata.bootsrc == 'H' ? " Hard" :
        osdata.bootsrc == 'h' ? "*Hard" :
        osdata.bootsrc == 'F' ? " Flop" :
        osdata.bootsrc == 'f' ? "*Flop" :
        osdata.bootsrc == 'N' ? " Net" : " ???? " );
    tmp = (short int)osdata.nodename;

    Trace3b( MY_TRACE_MAJOR << 12 | 1, TRACE_CRITICAL,
        (data.un.proc.start_time & 0x0000FFFF), /* store
            low word first */
        (data.un.proc.start_time >> 16),
        tmp,
        strlen(buffer),
        buffer);
}
```

traceinfo

print trace events (QNX)

This example produces a trace event similar to the following (use **traceinfo -r** to see the raw, unformatted trace output):

```
2 Aug 19 22:49:38 00043001 F792 2A92 0001 4820 7261 64
```

If you used a format string such as the following to interpret the data:

```
#major 0x00043      /* my trace code events */  
1 (%(minor)) Node %@2,d Booted on %@0, (date) : %@3,4.4s Boot
```

then the output from **traceinfo** would look something like this:

```
Aug 19 22:49:38 (1) Node 1 Booted on Wed Aug 19 21:38:26 1992 : Hard Boot
```

Files: /etc/config/traceinfo
 /usr/include/errno.h
 /usr/include/sys/trace.h
 /usr/include/sys/tracecod.h

See also: **Proc**, **tracectrl**, **tracelogger**, **ctime()**, **qnx_trace...()**,
 strftime(), and **Trace...()** functions in the *Watcom C Library Reference*

Syntax: `tracelogger [-f logfile] [-p percent_full] [-t minutes]`

- Options:**
- `-f logfile` Append trace information to this logfile. The filename can include `%m` for a month expansion (01-12), `%d` for a day expansion (01-31), or both.
If you don't specify `-f`, trace events are logged to standard output, allowing you to pipe them directly into the `traceinfo` utility.
 - `-p percent_full` Extract events from the trace buffer when it becomes this full. To have events extracted as soon as they occur, set `percent_full` to 0.
If you don't specify this option, `tracelogger` will set this high water mark to 50%, regardless of the current setting for it.
 - `-t minutes` Extract events from the trace buffer if none have been extracted for this many minutes (default is 5). To disable this feature, set `minutes` to 0.

Examples: Start in the background and create logfiles under `/tracedata` based on month and day (e.g. trace events on March 12 will be appended to the file `/tracedata/data.0312`). Also, extract events if the trace buffer becomes 20% full or if the 20% mark isn't reached after 3 minutes:

```
tracelogger -f /tracedata/data.%m%d -p 20 -t 3 &
```

Pipe trace events into the `traceinfo` program:

```
tracelogger -p 20 | traceinfo -
```

Description: The `tracelogger` utility automatically extracts trace events and logs them to a file. The logged event data is stored in a raw binary format that you can display in human-readable form with the `traceinfo` utility. Together, these utilities provide a means of capturing runtime events that you can review when diagnosing problems.

tracelogger

extract trace events and log them to a file (QNX)

Trace events are maintained by the Process Manager (**Proc**) in an in-memory buffer. When programs call one of the *Trace* library functions, a trace event is generated and quickly added to this in-memory buffer. For information on this family of functions, see the *Watcom C Library Reference*.

Most of the system managers (**Dev**, **Fsys**, **Proc**, **Net**, ...) generate trace events while they operate. To adjust the severity of the events that are logged as well as other trace parameters, use the **tracectrl** utility.

To have **tracelogger** extract data from the trace buffer immediately, you can use the SIGALRM signal. For example:

```
slay -s SIGALRM tracelogger
```



The **tracelogger** utility must be started as **root** and is typically run in the background. To stop **tracelogger**, make sure you're logged in as **root**, then type:

```
slay tracelogger
```

See also:

Proc, **traceinfo**, **tracelogger**, *Trace...()* and *qnx_trace...()* functions in the *Watcom C Library Reference*