Utilities

QNX Utilities

QNX is provided with a variety of utilities which aid in the management of files, the creating and formatting of diskettes, communicating with other computers, etc.

The following pages describe the standard QNX utilities and how to use them. Each utility will be described separately, starting with a brief description of the command syntax, followed by a list of all possible options and a few examples. The command will then be described in more detail. References to other similar or related utilities may be included.

In the following pages, the syntax of a utility may include a few of the following characters:

[...]	- **Square brackets are used to indicate OPTIONAL fields or arguments.**
|	- **OR bar is used between ALTERNATIVES, one of which must be selected.**
*	- **Asterisk following an argument indicates that a REPETITION of that field is also allowed.**

Any arguments which contain **boldfaced** characters may be abbreviated to the boldfaced characters alone.

Italics are used to represent data which the user must supply, such as a *filename* or *number*.

A few examples are included with every utility to help make the syntax of that command more obvious.

Command Summary

QNX is shipped with a rich variety of commands. It is well worth the time to look through all of this section, reading enough about each utility to get a good feel for what it does and when you would use it. This way, you will know what to look for later, when you need to refer to the manual for the precise details on a particular utility.

Learning the commands available on an operating system is one of the first tasks which confronts most new users. Depending upon your level of experience, this can be a slow process. To aid you in this respect we have listed most commands by category. The most frequently used and needed commands are marked with a star (*).

Communication

COMM		- Modem communication handler
QCP		- QNX Communications Protocol
QTALK	*	- Talk over Communications Line
STTY	*	- Set TTY options

Communication

COMM		- Modem communication handler
QCP		- QNX Communications Protocol
QTALK	*	- Talk over Communications Line
STTY	*	- Set TTY options

Configuration Utilities

ACCOUNTANT	- Accounting Task
BOOT	- Select an OS file for disk booting
CLOCK	- Display Time and Date On Console
CLRHOUSE	- A distributed database and name server
CRON	- Schedule commands in background
DATE	- Display or set date and time
DEF_SERVER	- Define id of global name server
DEFPIPE	- Define temp files for pipes
FDISK	- Create QNX disk partition
KBD	- Redefine keyboard layout
LOCKER	- Implement record locking in QNX
MOUNT *	- Mount disk drives, consoles or libraries
ONTTY	- Create task on another tty
OSCONFIG	- Change operating system parameters
PASSON	- Turn on password protection
QUEUE	- Implement queued message passing in QNX
RTC	- Get Date from Real-time Clock
SEARCH	- Define or Query the Disk Search Order
SHARON/SHAROFF	- Do/don't share code segments
SLICE	- Set the timeslice rate
TCAP	- Manage terminal capability database
TIMER	- Implement timing facility in QNX
TSET	- Set terminal type
TZSET	- Display or set timezone offset

Development Utilities

CC	- Compile Command (C, Basic, ...)
DEBUG	- Invoke the low level system debugger
LINK	- Link object files
MAKE	- Maintain and recreate files

Disk Utilities

CHKFSYS		- Check entire file system for consistency
DCHECK		- Check a disk for bad blocks
DCOPY		- Copy entire disk to a second disk
DDUMP		- Dump the contents of a disk block
DINIT		- Disk Initialization
DMARK		- Mark bad tracks on a disk
FDFORMAT	*	- Format floppy diskettes
PARK		- Park the heads of a hard disk
QUERY	*	- Query the utilization of a disk

Editing and Word Processing

ED	*	- Full screen editor
LED		- Line Editor

File and Directory Manipulation

CD	*	- Change current Directory
CHATTR		- Change Attributes of a file
CHGRP		- Change group (group number) of a file
CHMOD		- Change mode of a file
CHOWN		- Change owner (member number) of a file
DREL		- Release Directory
EO		- Execute On a list of files
FREL	*	- Release File
MKDIR	*	- Make directory
PWD		- Print Working Directory
RM	*	- Remove files
RMDIR		- Remove directories
WS	*	- Walk directory Structure executing a command
ZAP		- Zap damaged files out of existence

Graphics Utilities

BAR	- Draw Bar Charts
DUMP_IBM	- Make a hardcopy of Graphics Screen

Information Handling

CRYPT		- Encrypt/Decrypt files
DIFF		- Difference between two files
DUMP		- Dump the contents of a file in hexadecimal format
EXPL		- Explain
GREP		- Search a file for a pattern
LOCATE	*	- Locate patterns of characters in a file
MORE	*	- Display a file or stdin by pages
MSORT		- Merge sort utility
PACK		- Pack a file to reduce its size
PATCH		- Patch files
SIZE		- Display the size of a file
SORT		- Sort files
SPATCH		- Full screen patch utility
UNPACK		- Unpack a packed file
WC		- Word count
XLAT		- Translate Characters

Listing Directories and Files

DIR	*	- Display directory tree
FILES	*	- List Files
FSTAT		- Display file status
LS	*	- List directory

Miscellaneous User Commands

APB		- Send a Public Bulletin
BEEP		- Beep a user on another terminal/machine
BREAK	*	- Break a task
CALC		- A simple calculator
EC		- Execute a shell file
ECHO		- Echo arguments to standard output
KILL	*	- Kill a task
LOGIN	*	- Log-in to QNX
LOGOFF	*	- Terminate a QNX Session
PATH		- Change command search path
PRI		- Set priority
PROMPTT		- Display tty number
PRTSC		- Print Screen
SH	*	- Execute Shell Commands
SLAY	*	- Kill a task by name
SLEEP		- Sleep for a number of seconds
STYPE		- Type arguments on the terminal (no CR/LF)
TYPE		- Type arguments

Moving Files

BACKUP	*	- Backup Files to a backup diskette
CAT		- Concatenate input files into one larger file
COPY	*	- Copy files
CP	*	- Copy files
FBACKUP		- Archive files to floppy disk(s)
MV	*	- Move files
SPLIT		- Split a file into one or more files
TBACKUP		- Archive files to QIC60 tape(s)

Network QNX Utilities

ALIVE	- Set up/down status of a node
KILL_VCS	- Kill all virtual circuits to a node
NACC	- Set network access to disks, devices and CPU
NET	- Query machines on the network
NETBOOT	- Service boot requests from the network
NETSTATS	- Display network statistics
NETTEST	- Check data transmission between two nodes
POLL	- Poll nodes (network version only)

Printing Utilities

LIST	*	**- List files on the line printer**
LPS		**- Postscript Laser Printer Filter**
P	*	**- Print file contents on terminal**
SPOOL	*	**- Spool files to a printer**
SPOOLDEV		**- Create Pseudo-Printers**

System Information

ACCSTATS		- Print Accounting Statistics
DOPEN		- Display open devices
FOPEN		- Display open files
SAC		- Display system activity at each priority level
TSK	*	- Display task information
WHO	*	- Who is logged-in to the system

ACCOUNTANT - Accounting Task

Syntax:

accountant [**b**=*buf_limit*] [**f**=*file_name1*] [**f**=*file_name2*] [**t**=*ttys*] [+verbose] **&**

Options:

b=*buf_limit* - Number of accounting entries to buffer before switching to secondary file (if it was specified).
f=*file_name1* - Primary file to save accounting entries in.
f=*file_name2* - Secondary file to save accounting entries in.
t=*ttys* - List of tty's to gather accounting information on.
+verbose - Print each login/logoff on the screen.

where: *ttys* = *tty_range*[,*tty_range*]
tty_range = *tty_num*[:*tty_num*]

Examples:

```
accountant &
accountant t=0,2,3:6 &
accountant f=3:/accounting/log &
accountant f=3:/accounting/logX &
accountant f=3:/accounting/logX f=[4]3:/accounting/logX b=50 &
```

Description:

ACCOUNTANT is a task which runs in background and collects accounting information from the QNX TASK administrator. In a local area network you must run ACCOUNTANT on each node for which you wish to collect accounting information. If you do not specify which tty's to gather information on, then information will be gathered on all tty's.

Accounting information is generated on each login and logoff of a user. Each accounting entry is 48 bytes long and contains the following information.

```
struct accstat_entry {
    char accs_type;     /* 0-Not Used 1-Login 2-Logoff >3-User defined */
    char accs_tty;
```

```
unsigned accs_nid;
char accs_group_num;
char accs_user_num;
unsigned accs_time[2];
char accs_name[16];
char accs_spare[48 - 16-4-6];
} ;
```

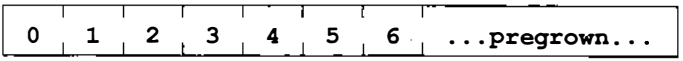
This program may be run in one of two modes, **file mode** or **buffered mode** (the default).

FILE MODE

When you specify the name of a file using the **f=***filename* option, accounting information is immediately written to the file. If the *filename* ends in a capital X, the X will be replaced by the year and month (eg: 8701 for January 1987). For each record written the file is opened and closed. If the file is busy, ACCOUNTANT will retry the open 10 times. If that fails it will buffer the information in memory. This procedure is followed for each accounting entry received. When the file is successfully opened, all buffered accounting entries will be written. There is room for about 1000 buffered entries, after which any new ones will be lost.

If you have specified a secondary filename, ACCOUNTANT will switch over and attempt to use it, after buffering a certain number of accounting entries. This limit defaults to 100 but may be changed using the **b=***buffer_limit* option. If you specify two files, any program which prints reports will have to read from both files removing accounting information in time order. ACCOUNTANT's ability to buffer entries in memory allows a reporting program to open the file for read (which locks out ACCOUNTANT), while it generates it's report.

Adding a single record to the end of a file would result in a very fragmented file over a long period of time. To reduce this fragmentation, ACCOUNTANT will pregrow the file by 200 accounting entries each time the file becomes full.



The first 4 bytes of record 0 contain the number of valid data records (as a long). Use fget(&lval, 4, fp) in C.
The first accounting entry starts at record 1.
The end of the file is pregrown with zero's.

BUFFERED MODE

If you do not specify any filenames on the command line, ACCOUNTANT will buffer all accounting entries. In this mode the **b=** option (if present) is ignored. To collect the accounting information you must write a program which sends messages to ACCOUNTANT, requesting the buffered accounting entries. The message consists of a single byte containing a message code of 3. The reply will consist of a msg_type followed by a 48 byte accounting entry.

```
struct user_accs_msg {
    char msg_type;
    struct accstat_entry accs_data;
    } ;
```

The the msg_type will be one of the following:

Accs_type	Reason
0	Data returned ok.
-1	No more buffered accounting entries.
-2	File mode, not buffered mode is running.
-3	Unable to save request.
-4	Bad message code in msg.type field.

ACCOUNTANT registers the local name "**account**". This allows a task in a local area network to find and poll each ACCOUNTANT in the system.

In both file and buffered mode, a task may send ACCOUNTANT 48 byte accounting entries. The msg_type code should be a 2 and the accs_type code must be greater than 2. These are written to the file or buffered depending on the mode. This allows special login/logoff programs to add extra accounting information. The accs_date field is always filled in by ACCOUNTANT. All other fields are left untouched.

See Also:

ACCSTATS

ACCSTATS

ACCSTATS - Print Accounting Statistics

Syntax:

accstats *file* ... [**t**=*ttys*] [**u**=*userid*] [**-**stats] [**-**verbose]

Options:

t=*ttys*	- Only print accounting information on these ttys.
u=*userid*	- Only print accounting information on this user. The userid may be a pattern.
-status	- Suppress the status information.
-verbose	- Suppress the printing of each login/logoff.

where: *ttys = tty_range*[,*tty_range*]
tty_range = tty_num[:*tty_num*]

Examples:

```
cd /acclogs
accstats *
accstats 8701 t=0,2,3:6
accstats 870[1234] u=dtdodge
```

Description:

ACCSTATS is a utility which will print a summary of the accounting information saved by the ACCOUNTANT program. The above examples assume that file names are in the form *yearmonth*. This is a standard option available with ACCOUNTANT.

See Also:

ACCOUNTANT

ALIVE

ALIVE - Set up/down status of a node

Syntax:

alive [[+]*nid_list*]* [**-***nid_list*]* [**b=***nid_list*]*
[**a=***nid_list*]* [**e=***nid_list*]* [**n=***target_node*]

OR

alive [+**b**usy] [+**u**p] [+**d**own] [+**n**ewboot] [**n=***target_node*]

WHERE
nid_list is *nid_range*[,*nid_range*[,...]]*
nid_range is *nid*[:*nid*]

Options:

+*nid_list*	- These nodes are UP.
-*nid_list*	- These nodes are DOWN.
b=*nid_list*	- These nodes are BUSY.
a=*nid_list*	- Allow these nodes network access to this node.
e=*nid_list*	- Don't allow these nodes network access to this node (exclude).
+**b**usy	- Inform POLLER that this node is BUSY.
+**u**p	- Inform POLLER that this node is UP.
+**d**own	- Inform POLLER that this node is DOWN.
+**n**ewboot	- Inform POLLER that this node has booted.
n=*target_node*	- Node on which to change up/down status (default: this node).

Examples:

alive	- Display current up/down status of all nodes.
alive n=3	- Display current alive status of node 3.
alive +2	- Node 2 is up.
alive -5,6,7:12	- Nodes 5, 6 and 7 through 12 are down.
alive +2,3 b=6	- Nodes 2 and 3 are up and 6 is busy.
alive -2 n=10	- Inform node 10 that node 2 is down.
alive +n	- Inform the POLLER that this node has booted.
alive +b n=7	- Inform node 7 that this node is busy.

alive e=1:10 a=7 - Exclude nodes 1 through 10 except 7.

Description:

The ALIVE command may be used to manually indicate to your local machine which nodes are up, down or busy. This command should not be needed if the POLLER is running.

The first form of the alive command will set the local up/down status, unless the **n=** option is used. The second form will inform the POLLER by default unless the **n=** option is used.

The operating system will not initiate any communications with another node unless it thinks that node is UP.

When a node changes from UP to DOWN, the operating system automatically releases any resources which were being used by tasks on that node (usually files), and unblocks any local tasks which were waiting for messages from tasks on that node. This cleanup does not take place on the transition from UP to BUSY, or BUSY to DOWN.

BUSY status is useful to temporarily remove a node from the network without releasing any resources on other nodes. This might be done on a node before using the debugger to prevent the POLLER from flagging the node as down.

alive +busy

When the debugging session has ended, the alive command can again be issued to tell the poller that the node is now UP and ready to receive communications from other nodes.

alive +up

When a node is first booted, it is usually a good idea to inform the poller that the node is now up. The poller will then transmit this information to all other nodes in the network. To achieve this, the following command can be included in the file "/config/sys.init" for your node:

alive +n

The POLLER is responsible for scanning all nodes which are alive and ensuring that they are capable of receiving communications. If a node fails to respond to the poller in a reasonable amount of time, it is marked as DOWN and all other nodes are informed. This has the effect of releasing any resources which this node may have been using anywhere on the network.

The **a=** and **e=** options inform the operating system on this node to prevent or allow network accesses from the nodes specified in the list. This permits "clusters" of nodes to co-exist on the same physical network without interfering with each other. This facility also provides a measure of security on larger networks.

The ALIVE command only applies to the network version of QNX.

See Also:

KILL_VCS
NACC
POLL

APB

APB - Send a Public Bulletin

Syntax:

apb *message*

Examples:

apb The pizza's here!!!
apb Cartridge disk two is now mounted!

Description:

The APB command will try to display a message on every terminal of every node in a QNX network. Each user will receive the message at the next convenient time. The SHELL will print the message, for example, just before it prompts for a new command.

If another APB is sent before the first one is printed on a terminal, the second message will replace the first message.

See Also:

BEEP

BACKUP - Backup Files to a backup diskette

Syntax:

backup *src_directory dest_directory* [*options*]*

Options:

c=0 - Copy only those files which have not changed since the last backup.

c=1 - Copy only those files which have changed since last backup (default).

c=x - Ignore change status.

d=*date* - Copy only files which have changed since this *date*. Date is of the form dd-mm-yy (default date is 1-1-80).

g=*group* - Copy only files which have the indicated *group number* (decimal number from 0 to 255).

m=*member* - Copy only files which have the indicated *member number* (decimal number from 0 to 255).

l=*levels* - Specifies how many *levels* of the specified source directory will be copied. Typically "l=1" is used to cause only the files at the current level to be copied. This option is particularly necessary when copying a directory to some sub-directory of itself (default is all levels).

p=[^]*pattern* - Indicates that only files with names that match this *pattern* are to be copied. Multiple "p=" options are allowed, in which case files which match ANY of these patterns will be copied. Also, if the pattern is preceded by an up-arrow (^), then files which match this pattern will NOT be copied.

pd=[^]*pattern* - Indicates that only directories with names that match (or don't match) this *pattern* are to be copied.

s=[c][d][p] - Suppress the copying of the
date
permissions (includes group/user numbers)
and/or
clearing of the changed bit on source file.

t=*time* - Copy only files which have changed since this time on the given date. Time is in the 24 hour format: hh:mm:ss (default time is 0:0:0).

+all - Equivalent to **c=x**

+before - Change the meaning of the date and time options to mean before. The default is now or after. **+n**ewest - If

the destination file is newer than the source, then do not copy; keep the newest of the two.

-cdirectory - If the directory does not already exist on the backup disk, prompt the user if he wishes to create it, skip it or retry after changing diskettes. The default is to always create directories as required.

-cfile - If the file does not already exist on the backup disk, prompt the user if he wishes to create it, skip it or retry after changing diskettes. The default is to always create files as required.

-error - Suppress message and prompt upon error condition.

-pause - Suppress prompt (don't pause to load disks).

-verbose - Turn off verbose option. This stops the BACKUP command from displaying the names of the files it is copying.

+write - Overwrite READ-ONLY files as well.

+quit - Abort on error.

Examples:

backup 3:/ 1:/ - Backup all files from drive 3 to drive 1 which have been modified since last BACKUP.

backup 3:/ 1:/ -cf -cd - Backup from drive 3 to drive 1. If files or directories don't already exist on drive 1, then ask user if they should be created.

backup /user/joe /user/bill +a s=c
- Copy all of joe's files to bill's directory without changing the modified state of joe's files.

backup 2:/dir1 1:/dir1 p=*.c p=^test.c l=1
- Copy files under /dir1 on drive 2 to the same directory on drive 1. Only copy modified files which end in ".c" except "test.c". Don't copy sub-directories.

backup 3:/user [2]1:/user d=14-8-84 t=9:00
- Backup files from 3:/user to 1:/user on node 2. Copy only modified files which have changed since 9:00 am on 14-Aug-84.

Description:

BACKUP provides a mechanism for backing up files. The decision to back up a file can be based on a number parameters. The default is to backup only those files which have changed since the last time backup was used. The **d=, g=, m=, p=, pd=** and **t=** options are by default don't care conditions.

Typically, a user will use the BACKUP command to copy all of the files which he has changed onto a backup diskette. This eliminates the need to copy every file onto the backup diskette at the end of every session. Only the changed files are copied, which can be far less time consuming than an entire disk copy. This also allows selective files or directories to be restored from a backup diskette. The date, permissions and owner are be default copied to the backup files. The LS command can be used to highlight those files which have been modified since the last backup.

ls +modified

As an example of using the BACKUP utility, consider a user who is developing a rather large application consisting of several C source programs. These programs may all reside on one disk under a directory called "/work". During the course of his day he will change many files, and create several object files as a result of compiling these files. The object files need not be backed up since they can always be created again from the source files. If he uses the naming convention that all C source programs end with ".c", and all object files end with ".o", the user could backup only his C programs by typing:

backup /work /backup p=*.c

The backup command will by default only transfer the files which were changed since the last backup. After backing up each file, BACKUP will clear the modified bit on the file. If you wish to save every file you may specify either **m=x** or **+all.**

BACKUP is also handy for copying entire directories (with the +a option) on the same disk. In this case it is desirable to suppress the clearing of the modified bit since you are using backup more as a move command than a backup command.

backup /dir1 /dir2 +all s=clear

Filename patterns can be any valid filename, with the following "wildcard" characters:

*	**will match any character, or run of characters**
?	**will match any one character**
[ccc]	**will match any of the characters in the brackets**

Some examples of filename patterns are:

p=*.*	**any file with a dot (.) in it.**
p=*.[ch]	**any file ending in 'c' or 'h'**
p=?a	**any two character filename ending in 'a'**
p=[abc]*	**any file starting with 'a','b' or 'c'**
p=^*.o	**any file which doesn't end in '.o'**

In cases where a directory is larger than the diskette, it is sometimes necessary to backup the files to different physical diskettes. The **p=** option may be used to select files.

```
backup /util 1:/util p=[abcdefghijkl]*
backup /util 1:/util p=[mnopqrstuvwxyz]*
```

Once you have created a backup disk the **-cf** and **-cd** options will then allow all files to be backed up, prompting the user to insert the proper backup diskette as required.

See Also:

COPY
FBACKUP
TBACKUP

BAR - Draw Bar Charts

Syntax:

bar *filename*[*,field*]* *options**

Options:

field	- Which field to plot.
-border	- No border around the graph.
+grid	- Draw horizontal grids.
+high_res	- Use high resolution graphics mode.
+hres	- Force to CGA 640x200, 2 colour resolution.
+Hres	- Force to EGA 640x350, 16 colour resolution.
-legends	- Don't include legends.
+line	- Plot line graphs.
+mres	- Force to CGA 320x200, 4 colour resolution.
+outline	- Draw outlines around each bar.
-relative	- Don't draw relative to baseline if b= option is specified.
+shade	- Shade bars instead of fill in colour.
+vstack	- Stack bars vertically.
-ylabels	- Don't draw labels on the y axis.
a=*char_angle*	- Angle of label characters (degrees)
b=*baseline*	- Draw a baseline at this value. Bars will be drawn relative to this line unless -r option is used.
c=*colours*	- Only use this many colours.
f=*field*	- Which *field* of the current file to plot.
g=*gridscale*	- Only use this many colours.
l=*labels_file*	- Label x axis with labels in this file.
m=*graphmode*	- Graphics mode to use.
p=*print_cmd*	- Create hard copy printout via this command.
t=*text*	- Label the Plot with this title.
x=*xsize*	- Limit x axis to this size.
y=*ysize*	- Limit y axis to this size.
n*=legend*n	- Replace legend *n* with this name. (eg: "1=sales" "2=1985 YTD")

Examples:

bar graph1	- Plot data in graph1
bar apples peaches pears	- Plot side-by-side
bar pigs cow sheep +v +H +s	- High resolution shaded plot stacked vertically
bar sales l=months	- Plot with labels on the horizontal axis.
bar plot1 -y -l	- Plot with no labels or legends.
bar 1985 1986 b=37.6	- Plot 2 files with a baseline
bar t1 t2 t3 b=10.0 +r	- Plot 3 files showing deviation from a baseline.
bar data,1 data,2 data,3 +l	- Line graph of 3 fields.
bar test "p=dump_ibm [2]$lpt"	- Plot graph and create hard copy on node 2's printer.

Description:

BAR is a business graphics utility which draws bar charts based on information in one or more files.

BAR requires that a GRAPHICS LIBRARY and a FLOATING POINT LIBRARY be mounted before using. The utility will determine the best dimensions and choice of colours depending on the installed graphics library. Vertical and horizontal dimensions can be reduced using the **x=** and **y=** options. BAR will attempt to use the highest resolution mode which is supported by the currently mounted graphics library. The **+h**, **+m**, and **+H** options can be used to force a particular resolution. The **m=** option can also be used to force a resolution.

The information in each file is processed, using the FIRST field on each line of the file (unless another field is requested). The remainder of the line can be used for comments. The number of bars to draw is determined from the file which contains the maximum number of lines. The width of each bar is adjusted automatically to provide the best plot. The plot is automatically scaled in the vertical dimension to accommodate the largest entry.

A border is drawn around the plot unless **-b** is specified, with appropriate grid marks drawn on both left and right borders. The vertical axis is labeled automatically with the minimum and maximum values unless **-y** is specified. If a grid is requested (**+g**), then only the grid lines are labeled. The grid scale is selected to best fit the data. Different grid scales may be chosen using the **g=** option.

The orientation of the label characters can be changed with the **a=** option, although this is seldom useful.

Each graph is assigned its own colour where possible unless shading is requested (**+s**), in which case each graph will be assigned a unique style of shading. The graphs are plotted horizontally by default (normal bar chart) unless the vertical stacking option is specified (**+v**) in which case the graphs are stacked on top of each other. The scale will be adjusted automatically for vertical stacking. Negative values are forbidden in vertical stacking mode.

Legends will be drawn on the bottom of the graph, unless the **-l** option is used. The legends can be customized using the **1=**, **2=** and similar options. If no legend is specified, the name of the file will be used.

Relative plots with respect to a baseline are drawn if a baseline is given (**b=**) and the **-relative** option is NOT specified. Relative plots show the bars as either positive or negative changes from the baseline. Relative plots are not allowed with vertical stacking (**+v**).

The horizontal axis can be labeled by using the **l=** option to specify a file containing horizontal labels. This could be, for instance, a file consisting of the months of the year. Extra elements in the label file will be ignored.

Line graphs are plotted if the **+line** option is specified. Dashed lines will be used if **+shade** is also specified.

Hard copy can be obtained by specifying the name of a command which will make a copy of the graphics device, once the plot is completed. BAR pauses when the plot is finished, and if a command has been specified with **p=**, will call that command if the user types the character **p** (for print). Any other character will cause BAR to terminate without creating hard copy (see the last example).

Most of the command line options can also be embedded **within** the file being plotted. Any line which starts with an opening square bracket ([) will cause BAR to parse the string within the brackets as if it were typed on the command line *after* all the other arguments.

Any line starting with a double-quote (") is ignored, so can be used as a comment line.

Any line which starts with characters enclosed in round brackets () will use those characters as a *label* for that entry. The data within round brackets will otherwise be ignored.

All remaining tokens on a line are treated as *files* to be plotted.

The simplest form of a data file is simply a list of numbers, in one or more columns, such as shown below:

```
56.1 26.1 9.1
43.1 22.2 8.9
45.7 18.2 8.8
39.4 16.7 9.0
35.5 16.3 8.7
30.9 14.1 8.5
28.6 10.0 8.3
29.2 11.0 8.8
34.3 10.1 9.1
37.3 15.5 9.0
45.7 20.3 9.3
50.0 24.4 9.0
```

The 3 fields of this file could be plotted as a STACKED BAR graph using the command:

bar file,1 file,2 file,3 +v +g "t=1987 Production" 1=H/W 2=S/W 3=Other

If this file is **always** plotted with the same arguments, you could choose to embed the parameters *within* the file, by putting the following lines at the *beginning* of the file:

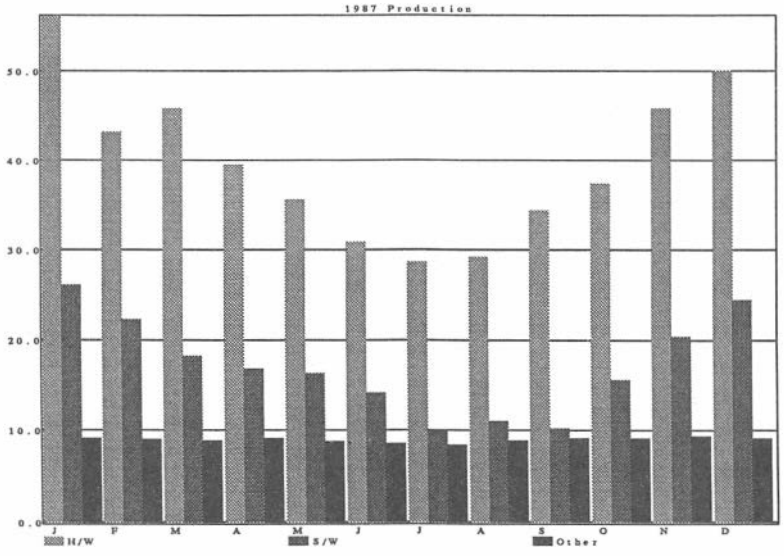
```
" field 1 is plotted first
[f=2]
[f=3]
[+v]
[+g]
[t=1987 Production]
[1=H/W]
[2=S/W]
[3=Other]
```

With these extra lines, the same plot will result from the command:

bar file

In order to demonstrate the capabilities of BAR, the above data was used to generate the following actual **untouched** graphs:

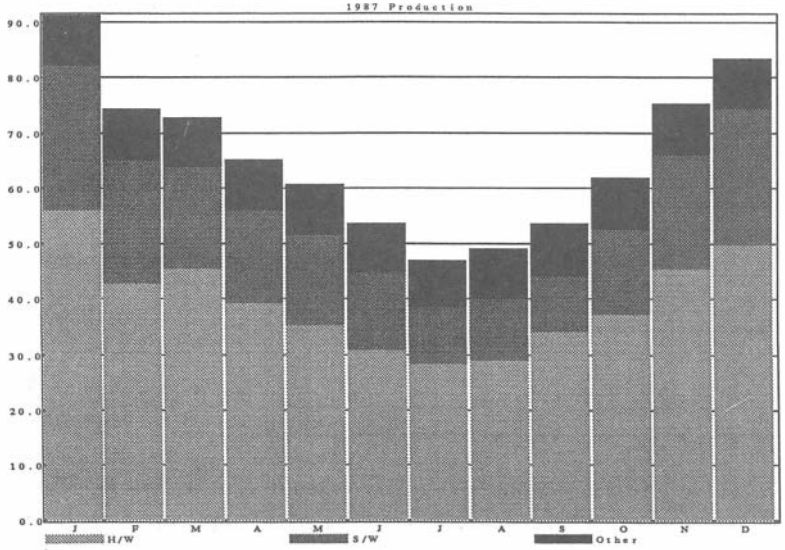
BAR GRAPH



bar file,1 file,2 file,3 +g "t=1987 Production" 1=H/W 2=S/W 3=Other

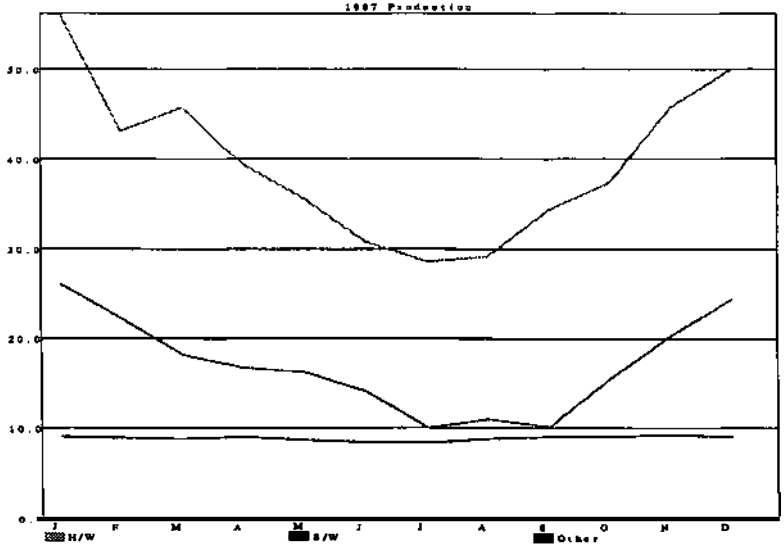
BAR

STACKED BAR GRAPH



bar file,1 file,2 file,3 +g +v "t=1987 Production" 1=H/W 2=S/W 3=Other

LINE GRAPH



bar file,1 file,2 file,3 +g +l "t=1987 Production" 1=H/W 2=S/W 3=Other

See Also:

DUMP_IBM
MOUNT

BEEP

BEEP - Beep a user on another terminal/machine

Syntax:

beep *userid* [*message*]
beep [*node*] *tty* [*message*]

Options:

userid - A user signed on in the system.
node - A node within a QNX network.
tty - The tty number to beep.
message - Message to send.

Examples:

beep gord
beep gord Its time for lunch
beep 3 Please get off the modem port
beep 1 3 Please get off the modem port on node 1

Description:

BEEP will send a message to a user. The user may be logged in on the console or any terminal in the network. A stand-alone system is considered as a single node network. If the message is omitted, the user will only receive a series of beeps to attract attention. The user might then drop into CHAT or MAIL. If a message is specified, it will be printed at the current cursor position on the user's screen. This may cause minor confusion if the user was running a full screen application like ED. The beeps will be sent right *after* the message is displayed.

Note that BEEP opens the user's screen for write. If the user has partially typed in a line, then BEEP will wait for a carriage return. If a user is signed on more than once, the message will be delivered to each screen the user is logged onto.

See Also:

APB
WHO

BOOT - Select an OS file for disk booting

Syntax:

boot *os_file_name* [options]*

Options:

-pause - Don't pause.
+qnxloader - Write the special QNX loader.
-qnxloader - Remove special QNX loader.
c=*config_file* - A configuration file to bind into the OS.
d=*disk_driver* - A hard disk driver to bind into the OS.

ROM Boot options:

+H*ard_disk* - The disk is going to be a hard disk device.
+F*loppy_disk* - The disk is going to be a floppy disk device.
+P*artition* - Is a partitioned disk and the boot loader will be made aware of this.
-P*artition* - Is not a partitioned disk and the boot loader will be made aware of this.

Examples:

```
boot 1:/netboot/os.2.10atp
boot 3:/netboot/os.2.10atp d=/drivers/disk.at +q
boot 3:/netboot/os.2.10atp
```

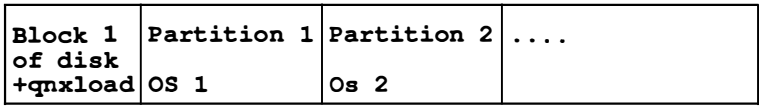
Description:

BOOT selects the name of a file you wish to boot from disk. The disk may be a floppy or a hard disk.

To understand the purpose of the +qnxloader, some background information on hard disk booting is necessary.

Booting from hard disk is a two step procedure. The BIOS first reads in block one of the hard disk and transfers to a bootstrap loader which decides which of the 4 partitions is active. This loader reads in the first block of the active partition and transfers to another bootstrap loader. This loader is operating system specific, and loads in the operating system for that partition.

BOOT



The first level bootstrap loader as provided by DOS will always boot the active partition as set by the QNX or DOS FDISK program. If you specify the **+q**nxloader option, we will replace this bootstrap with one that will still default to the active partition, but will pause for several seconds and allow you to override from the keyboard by typing a single digit from 1 to 4. This gives more flexibility. If you have non-standard hardware, or are **already** using a special loader, then this loader may cause you problems. For 99.9% of all installations we recommend you use the QNX loader. You need only specify the **+q**nxloader option once.

When you specify **+q** option, **BOOT** looks at the partition loader already on the disk (probably DOS's) and if it is different than the QNX loader, saves it away to a file called ***/config/hdisk.ldr***. Then, at any time in the future, if you use the new **-q** option, you can restore the original loader back to the partition sector.

When QNX boots, it needs a hard disk driver in order to access the hard disk. Unlike DOS it cannot use the BIOS driver which does not work in protected mode. You must specify which hard disk driver to bind into a hard disk boot. This is done using the **d=***disk_driver* option. This option takes the name of a mountable disk driver, and creates a file called **/config/hdisk.cfg** which will be loaded in from hard disk along with the operating system file. You need only specify the **d=*driver*** option the first time you configure for hard disk booting.

You may have several OS image files under the directory **/netboot**, and you may use the BOOT command to select which one you wish to boot from. This is very handy if you download a beta version of an OS from the QNX Update System and wish to try it out.

boot /netboot/testos - Select a test version to boot.

The BOOT command also allows you to specify a configuration file which lets you change some of the operating system defaults; such as the number of open files supported. This is specified as a **c=*filename*** option. You may select **any** file name, however we recommend that you choose names like those shown below. If you boot across a QNX network, then you **must** use a name as shown below.

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

The **sys.cfg** file is created and maintained by the OSCONFIG command. It takes the file name as an argument. For example

osconfig 3:/config/sys.cfg

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

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

NOTE: Once booting from hard disk, you should **not** remove the O/S file, the driver file or the config file.

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

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

ROM BOOT options :

+Hard_disk

The boot disk is going to be a hard disk device. This option conditions the boot loader so that it uses the hard disk BIOS read call instead of the floppy disk BIOS read call (ie: register dl has 0x80 'or'ed in for the read).

+Floppy

The boot disk is going to be a floppy device. This option conditions the boot loader so that it uses the floppy BIOS read call instead of the hard disk read call. (ie: register dl does not have 0x80 'or'ed in for the read).

+Partition

The boot disk being prepared is a partitioned disk and the boot loader placed on the disk will be made aware of this. Normally, floppy and ROM disks would not use this option.

-Partition

The boot disk being prepared is not a partitioned disk and the boot loader placed on the disk will be made aware of this. Normally, floppy and ROM disks would use this option.

These options should be used to descibe what the ultimate ROM disk will boot as, and should not necessarily describe the media the boot command is executed on. For example, if you had a ROM disk driver that acted like a hard disk, you would use the +H option, even though you were preparing the ROM image on a floppy disk.

See Also:

NETBOOT
MOUNT
OSCONFIG

BREAK

BREAK - Break a task

Syntax:

break *task-id*

• *This is a local shell command* •

Description:

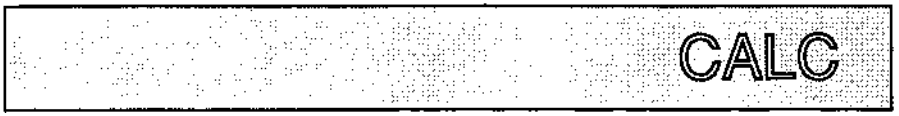
BREAK allows a task whose taskid is known to be killed gracefully. The taskid can be found by using the TSK command. Only tasks which are created by yourself can be broken. The super-user can break any task in the system with the exception of the operating system and user *administrator* tasks.

BREAK is typically used for aborting background programs since they are no longer associated with a terminal, and therefore cannot be terminated with the BREAK key. A background LIST program is an example of a task which a user may wish to *break.*

BREAK is the preferred method of killing tasks, since KILL will not allow a task to terminate normally, so may result in devices or files being left in an indeterminate state. BREAK will usually only kill a task which manages critical resources at a time where it is safe to do so.

See Also:

KILL
SLAY
TSK



CALC - A simple calculator

Syntax:

calc [+full_screen [*rows columns*]]

Options:

+full_screen	- Full-screen/interactive mode.
rows	- Number rows for calculator "window".
columns	- Number columns for calculator "window".

Examples:

calc	- Drop into calculator.
calc +f	- Invoke full screen mode.
calc +f 10 35	- Invoke CALC in 10x35 window.

Description:

CALC performs a simple desk-top calculator function.

CAUTION: before CALC can be used, a floating-point shared library must have been installed:

mount lib /config/float.slib
or
mount lib /config/float8087.slib

CALC evaluates expressions with internal **double precision** accuracy. This results in 15 significant digits of accuracy, and values between 4.19E-307 to 1.67E308 being supported.

Numerical constants can be entered in standard arithmetic form, or as Octal, Hexadecimal, or ASCII character constants:

10	**- the number 10**
1.1e-37	**- the number 1.1 x 10**37**
$123abc	**- a Hexadecimal number**
@17777	**- an Octal number**
'A	**- the ASCII character 'A' (0x41)**

Within the calculator, expressions are entered in algebraic form (NOT reverse Polish). Brackets may be used to group expressions. The following arithmetic operators are supported:

- \+ **Add**
- \- **Subtract**
- * **Multiply**
- / **Divide**

CALC supports 26 *registers* identified by the lower-case letters 'a' through 'z'. Values are stores into *registers* with the '=' character. For example:

p = 3.1415926 / 2

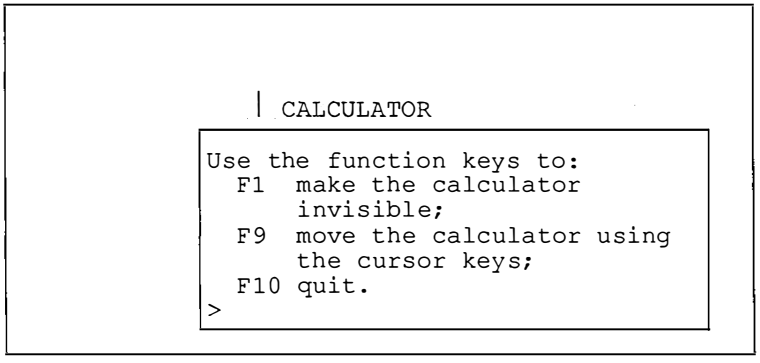
Registers can be used anywhere a *number* can be used:

360 * (.707 + p)

While entering data, you are free to *edit* the line using the left arrow, right arrow, Ins, Del, and Rubout Keys. In addition, the UP arrow key can be used to recall the previous line, which can then be edited.

To terminate CALC in interactive mode, type Ctrl-D.

In full-screen mode, the following screen will be displayed:



As shown, the F1, F9, and F10 keys have special meaning.

When F9 is typed, CALC will indicate that it is in *move* mode with an asterisk in the upper left corner of its "window". The keypad arrow keys can then be used to move the window around the screen. When the "window" is where you want it, type F9 again.

If the CALC window is *hiding* some part of the screen you wish to see, you can type F1 to make CALC temporarily invisible. Typing F1 again will re-display the CALC "window".

See Also:

PRTSC

CAT

CAT - Concatenate input files into one larger file

Syntax:

cat [*input_file*]* [>*output_file*]

Examples:

cat small1 small2 >big
cat file1 file2 file3 file4 >massive
cat file
cat file1 - file2 >new
cat >easy

Description:

CAT will concatenate a list of files creating one larger file. The default output is the standard output (usually the terminal), which the user will typically redirect to a file or line-printer.

The standard input (keyboard) can be specified with a dash (-) which will cause CAT to insert data taken from the standard input until an end-of-file (Ctrl-d on keyboard) is encountered (as shown in the example). Also, you can quickly create a file without an editor by using CAT, as in the last example, which will read data from standard input until end-of-file.

See Also:

SPLIT

CC - Compile Command (C, Basic, ...)

Syntax:

cc *file* [*options*]*

Options:

+Asm	- Do not run the asm command, and retain the .a file (asm input).
-core	- Do not run the link command even if only one source file is entered, and retain the .o file (link input).
-Core	- Same as -core.
-Exec	- Don't execute any commands (debug).
+L*c*	- **Include any libraries needed for source modules ending in the suffix *c*. (see +W*c* option)**
-Nested	- Nested comments will all terminate on first "*/". Usually every nested level requires its own end sequence.
+Optimize	- Run the "C" optimizer.
+optimize	- Run the "C" optimizer.
-preproc	- Do not run the C preprocessor (CPP) (default).
+preproc	- Run the C preprocessor (CPP).
+Preproc	- Only run the C preprocessor leaving the output in the same name with the suffix capitalized. For example test.c -> test.C test.b -> test.B
+Qbug	- Create symbol tables for a symbolic debugger.
-Remove	- Don't remove the temporary files.
-Stackchk	- Suppress the stack checking code produced by the code generator. (same as **+Wi,s=0**)
-Tmp	- Do not use /tmp as the work directory. Temporary files will be created in your current directory.
-Verbose	- Suppress the printing of each phase of compilation.
+Verbose	- Very verbose output.

+8087 - Force the code generator to produce native 8087 code for floating point operations. During link the native 8087 system libraries will be searched. The default is to generate software interrupts into a floating point shared library. Native code requires an 8087/80287 but can be as much as 500 times faster.

+W*c,arg* - This option is used to pass the argument (*arg*) to a specific processor (*c*) as follows:

?	cpp	C pre-processor
a	asm	assembler
b	cb1	basic parser
c	cc1	C parser
i	cc2	code generator
y	yacc	yacc

c=*core* - Specify name of compiled loadfile.

E=*obj.o* - Use the indicated object module in place of the system one in **/lib/entry.o**.

O=*objdir* - Place all object modules in the indicated directory rather than the current directory.

T=*treesiz* - The parsers and code generator build parse trees in internal tables. Very complex expression may require you to increase this space. The default is 2500.

+386[/16] - Internal use only.

The following three arguments cause the optional C preprocessor to be invoked (forcing +**p**), and are used as follows:

de=*name*=*value* - Works as if "#define *name value*" was in the source.

un=*name* - Removes any "#define *name*" implemented by the C preprocessor.

in=*dir* - Adds *dir* to the automatic library search for **#include <name.h>** files.

Examples:

cc hanoi.c

Compile **hanoi.c** and create an executable program in the current directory with the name **hanoi**.

cc hanoi
One by one append the following suffixes to the file **hanoi** , {**.c, .b, .y, .i, .a**} and on a match invoke the appropriate parser. If the file **hanoi.c** exists, then this will have the same effect as the previous example.

cc clock.c -S
Compile **clock.c** and create an executable program in the current directory with the name **clock**. The program will not contain stack checking code. This will make it slightly smaller and faster.

cc clock.c +Wc,e=errors
Compile **clock.c** and create an executable program in the current directory with the name **clock**. Any errors detected during compilation will be placed in the file **errors**.

cc prtsc.c c=/cmds/prtsc +V
Compile **prtsc.c** and create an executable program in the **/cmds** directory with the name **prtsc**.

cc banner.c -c de=function=1
Compile **banner.c** using the optional preprocessor to add "#define function 1" and retain the output **banner.o** in the current directory. (link was suppressed)

cc fastprog.a -c +Wa,+286
Assemble fastprog.a program and pass the argument +286 to the assembly. The link was suppressed and an object module fastprog.o will be produced.

cc basic.b csub.c
Compile **basic.b** and C function **csub.c** producing two object files (**.o**). The two object files will be linked and, because a basic program was compiled, the basic run time library "/blib" will be included during the link. Since more than one file was specified, the executable will be in a file called **core**.

cc basic.b csub.c c=basic.test
Same as above but executable (core) placed in **basic.test**.

cc *.c /mylib/fun.o +Lb
Compile all C programs in the current directory. Object modules (**.o**) will be produced for each C program. The object modules will be linked along with /mylib/fun.o creating an executable load module called **core**. Search Basic libraries.

cc *.o
Pass all object modules to the linker and produce an executable load module called **core**. Don't forget the **+8087** option if you wish inline 8087 code. You may also need to specify the **+L***c* option to search language dependant

libraries.

cc x=ofiles
Invoke the linker with the option **x=ofiles**.

Description:

CC is the QNX compiler interface. It 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 choice of parser to use is based upon the file suffix.

.c	**-> C compiler**	**/cmds/cc1**
.b	**-> Basic compiler**	**/cmds/cb1**
.a	**-> Assembler**	**/cmds/asm**
.y	**-> Yacc**	**/cmds/yacc**
.i	**-> Code generator**	**/cmds/cc2**

The parser must exist in order for you to use it. The CC command was designed to work with potential future parsers and may contain entries for parsers which are not yet available. If the suffix on a file is omitted, a scan for a file with each of the possible suffixes is made. Upon a match the appropriate parser is invoked. The suffixes are scanned in the following order.

.c -> .b -> .y -> .i -> .a

If only one source file is given, the linker will create an executable program in the current directory, with the same name as the input file, but with the suffix stripped off. If more than one file is specified, the executable will be placed in the file **core**. You may use the linker option **c=*name*** to force the destination of the executable.

The **-core** option will suppress the LINK phase. This is a heavily used option by developers which have many separate source files which they update and modify individually.

There are special system libraries for each parser which will automatically be included in the link when a source module for that parser is encountered. If only object modules are specified, then only the standard C libraries will be searched. You can include libraries using the **+L*c*** or **l=*library*** option which is passed to the linker.

cc *.o l=/blib (or use +Lb option)
or
cc x=ofiles l=/blib (or use +Lb option)

The code generator is capable of generating either native 8087 code or calls into a mounted shared library for floating point operations. The default is to emit calls to the shared library. This allows a program to run on a machine which does not contain an 8087 co-processor. One of two shared libraries may be mounted.

mount lib /config/float.slib (no 8087)
or
mount lib /config/float8087.slib (8087)

The second shared library is an order of magnitude faster than the first. Note that this approach allows you to distribute one program which can take advantage of the 8087, if it exists, or use software emulation if it does not. For really time critical operations, the overhead imposed by the shared library can be avoided completely by specifying the +8087 option, which will cause the code generator to produce native inline 8087 code. This switch will also link in the native 8087 math libraries. No shared library need be mounted but an 8087 MUST be present.

The QNX C compiler contains a limited C preprocessor which should handle 98% of all C programs. By building the preprocessor into the parser, it eliminates one phase, resulting in a faster compilation. You may use the full implementation of the C preprocessor by specifying the **+p** option. Very large programs may cause the C parser (cc1) to run out of memory due to a large number of **#defines**. Each define takes up memory. In this case the preprocessor will strip the **#defines**, giving the parser more memory to work with.

In some cases complex defines or macros may produce parser error messages which are difficult to understand. By using the **+P** (capitalized) option, the expanded output will be left in a file with the same name, but with the suffix capitalized. You may examine this file with the editor.

The full implementation C preprocessor also implements a number of built-in manifests including: i8086, qnx, quantum, __DATE__, __FILE__, and __LINE__. __DATE__ is a string containing the date the file was compiled. __FILE__ and __LINE__ are dynamic manifests containing the filename and linenumber of the source.

You may occasionally wish to examine the assembly code produced by the code generator, maybe in attempt at optimization or perhaps in tracking down a suspected code generator bug. The +Asm option will stop the compile after cc2, leaving a file with the suffix **.a**.

The -Stackcheck option will suppress the generation of stack overflow code by cc2. This typically results in slightly smaller and faster code. We recommend that you leave stack checking on for files that contain recursive functions.

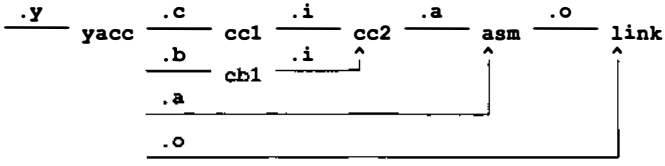
All arguments and options that are not listed above are assumed to be instructions for the linker and will be passed to the LINK command. Object modules (files ending in .o) are also passed directly through to the linker. The choice of upper case options is deliberate to avoid confusion with options known by the processors (parser, code generator, assembler and particularly the linker) invoked by CC. For example, you can change the default stack size by specifying s=*stacksize* which will be passed to the linker. The default of 2000 bytes may need to be increased for recursive programs or programs which place large arrays on the stack. If possible move arrays outside of functions to avoid having to override the stack size.

If you need to specify a parameter to any of the processors, you may use the +W*c,option*. Check the documentation on each processor to determine its options or type its name followed by a question mark (?).

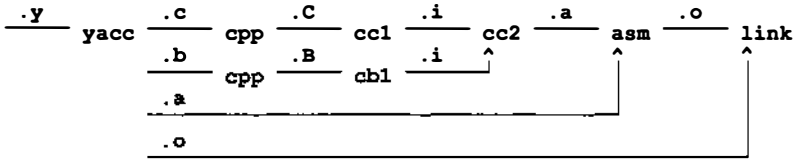
cc1 ?

The following charts illustrates the flow of control for the different parsers invoked.

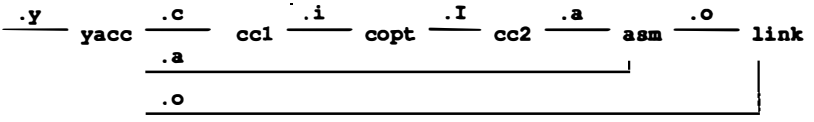
WITHOUT THE PRE-PROCESSOR



WITH THE PRE-PROCESSOR



WITH THE "C" OPTIMIZER



See Also:

LINK

CD

CD - Change current Directory

Syntax:

cd [*directory*]

Options:

directory - Name of new directory.

Examples:

```
cd /user/bill
cd 3:/expl
cd [4]3:/user/dan
cd
```

• *This is a local shell command* •

Description:

CD allows the user to define his working directory. All file names which do **not** start with a slash (/) or a drive number, are assumed to be *relative* to the current working directory. The default working directory is defined when the user logs into the system. Issuing CD with no arguments will restore the working directory to the initial login directory.

CD allows the user to position himself anywhere in the QNX hierarchical file system. If a QNX network is available, then this can be on any disk of any computer in the network. CD eliminates the need to always specify complete pathnames.

For example, if a user "bill" has a directory called "work" which contains several files he is currently working on, he can specify the name of one such file as "/user/bill/work/file1". This is the complete pathname of that file.

Alternately, he could set his current directory by typing:

cd /user/bill/work

He can now reference the file as simply "file1". Since the user "bill" is initially working at the directory "/user/bill" when he logs in, he could have typed:

cd work

to move himself down one level to the directory "work".

When he has finished working on the files under directory "work", he can move back up to his previous working directory by issuing:

cd ^

The PWD command can be used to display the current working directory.

Every task which runs in QNX has its own *current directory*. When one program (like the SHELL) creates another program, the *created* program will initially **inherit** the current directory of its *creator*. This new program may then change its current directory. Keep in mind however, that when this program terminates and control returns to the *creator*, the new current directory will be lost, since the original program has never changed its own current directory.

This fact is especially noticeable when SHELL files are invoked (BATCH files to DOS users). The SHELL file is controlled by a new SHELL task which has its own current directory (initially yours). If the SHELL file changes its current directory, then it will be changed only as long as that SHELL file is running. When it has finished, your current directory will be the same as it was before the SHELL file was executed.

See Also:

PWD

CHATTR

CHATTR - Change Attributes of a file

Syntax:

chattr *filename* options**

Options:

a=[+|-][rwaem]	- Change attributes of file.
a=[+|-][rcbm]	- Change attributes of directory.
g=*group*	- Change the group number of a file.
m=*member*	- Change the group member number of a file.
n=*name*	- Change name of file.
pg=[+|-][rwaem]	- Change file permissions (for my group).
pg=[+|-][rcbm]	- Change directory permissions.
po=[+|-][rwaem]	- Change file permissions (for other users).
po=[+|-][rcbm]	- Change directory permissions.
p=[+|-][rwaem]	- Same as combined **po=** and **pg=** (files).
p=[+|-][rcbm]	- Same as combined **po=** and **pg=** (directories).
s=[+|-][bmu]	- Change file status.
+date	- Change date to current date.
-busy	- Make un-busy (Caution: see text).

Examples:

chattr old n=new	- Change name of "old" to "new"
chattr main.c g=12 m=4	- Change ownership of file.
chattr core p=-wa a=+e	- Remove write and append permission from "core", add execute attribute.
chattr /user/bill +d	- Update date of file to today
chattr file.dat s=-b	- Unbusy a file (Caution: see text).

Description:

CHATTR allows the status, attributes, permissions, date, owner, and name of a file to be changed.

The status flags which can be changed on a file are

b - **BUSY**. Set when the file is opened for write. If the system crashes

without closing the file, this bit will be left set in the directory entry.

m- **MODIFIED**. Set whenever a file is modified (open for W, R/W or A) and cleared by the BACKUP command.

u - **USED**. This flag can only be cleared. It is the equivalent of zapping the file with the ZAP command.

The attributes and permissions which apply to files are:

r - **READ**. This capability allows files to be read or copied.

w- **WRITE**. This capability allows data to be written into a file. Old file contents will be lost.

a - **APPEND**. This capability allows new data to be written at the end of a file. The current contents of the file are not changed.

e - **EXECUTE**. This capability allows a file to be executed. All files created by the linker are created with execute permission. Any copies of an executable file will also need this permission before the file can be executed.

m- **MODIFY**. This capability allows the CHATTR command to change the attributes, permissions, name, owner, and date of a file. If this option is ever turned off, the attributes can never again be changed. Modify permission can only be turned off by specifying **a=-m**. Take care **never** to remove modify permission from a file which doesn't have write permission, or it will remain forever!

The attributes and permissions which apply to directories are:

r - **READ**. This capability allows directories to be read or listed.

c - **CREATE**. Allow new files to be created under this directory.

b - **BLOCK**. Prevent directory searches below this directory.

m- **MODIFY**. Same as for files.

Attributes of a file or directory apply to the owner of the file and the super-user (group 255). Permissions are those attributes which are permitted for other users. There are two types of permissions

Group - these apply to members of the same group
Other - these apply to members of other groups

Careful setting of permissions on directories and files allows a user to "protect" his files from other users of the system in a multi-user environment.

The owner of a file can be changed with the **g=** and **m=** options. These options will usually be used by the operator when creating initial user directories. The group and member number must be a number from 0 to 255. The owner (group.member) number is assigned to a user in the password file. Only the super-user can invoke

the **g=** option. The **m=** may be used by the super-user (group 255) or a group leader (member 255 of a group).

Filenames can be changed using the **n=** option. New names refer only to the lowest level of the complete pathname. For example, to change the name of the file "/user/joe/dir1/bill" (which has the name "bill"), one would use:

chattr /user/joe/dir1/bill n=fred

Valid filenames contain 1 to 16 characters, consisting of upper or lower case letters (they are distinguishable), numbers, underscores (_), dots (.), and dollar signs ($), provided that $ is not the first character.

The date and time of a file can be updated using the **+d** option, which changes the date and time stamp of the file to the current date and time. This is especially useful for upgrading files which were created or changed during a session where the user neglected to set the system date.

Busy files can be made unbusy with the **-b** option. The command CHKFSYS should then be run to verify that the unbusied file is consistent. If not, the file should be copied to another file and the original zapped using "chattr s=-u" or ZAP. An inconsistent file should **NEVER** be left on a disk and should **NEVER** be released by using FREL.

See Also:

CHKFSYS
ZAP

CHGRP - Change group (group number) of a file

Syntax:

chgrp *group_number file**

Examples:

chgrp 4 file1 file2

Description:

CHGRP changes the group number of the specified files to *group_number*. This number must lie between 0 and 255 inclusive. Group 255 is a the super group.

Note that this is a shell file which invokes the CHATTR command.

See Also:

CHATTR
CHOWN

CHKFSYS - Check entire file system for consistency

Syntax:

chkfsys *drive* [*options*]*

Options:

drive - Disk drive on which to check file system.
b=*file* - Fiie of bad blocks.(see DCHECK utility)
f=*/path* - Check the indicated pathname only.
-stats - Suppress statistics calculation and display.
+rebuild - Suppress error messages relating to the bitmap.
-fix - Don't fix anything.
-pause - Don't ask user if errors should be fixed.
-verbose - Don't print names of files being checked.

Examples:

chkfsys 2 - Check file system on drive 2.
chkfsys 2 +r - Check file system on drive 2 in order to rebuild the bitmap.

Description:

CHKFSYS will perform a consistency check of the file system on the requested drive. It will recursively walk the file structure visiting every file on the disk. During the walk checks are made on the directory entry of each file and the extents that make up the file. A bitmap is constructed in memory which is consistent with the block allocation of **all** files and directories on the disk. This bitmap is then compared to the one on the disk. If they differ the user is given the option of replacing the bitmap on disk with the one constructed in memory.

IMPORTANT: CHKFSYS should only be used when the system is idle. There should be NO open files when CHKFSYS is running.

CHKFSYS is usually used to recover blocks which were lost through the use of the ZAP and CHATTR command.

zap file
chattr s=-u file

In this case CHKFSYS will complain that there are blocks used in the bitmap which are in fact NOT used by any file. These blocks may be recovered by writing the reconstructed bitmap back to disk. CHKFSYS will attempt to read each of these blocks and will NOT mark bad blocks as available.

The **b=** option may be used to inform CHKFSYS of known bad (or marginal) blocks. Each line of the file should contain 1 hexadecimal block number. The block numbers must be sorted in ascending order.

If **b=** is not specified, then CHKFSYS will use the file "/bad_blks", if it exists on the drive being checked. This file is created automatically by the command:

dcheck +mark

CHKFSYS will tell you if any files are using blocks which are now known to be bad.

If CHKFSYS complains that a block is used by more than one file, it could indicate one of two problems. If the **b=** option was specified, or "/bad_blks" exists, then it probably indicates that the file uses a block which is bad and marked as used in the bitmap. If there are no known bad blocks, then a multiple allocation of a single block has occurred. In either case, the file should be saved on **ANOTHER** disk (if possible) and then zapped. CHKFSYS should then be run again to update the bitmap, after which the saved file may copied back onto the disk.

In general, whenever the bitmap is replaced, CHKFSYS should be run a second time to ensure that the file system is indeed consistent.

CHKFSYS may also be run after a system crash or power failure which has left some files busy. The files may be made unbusy using the chattr command

chattr s=-b file

which may then be followed by a CHKFSYS to ensure that no damage to the file system has occurred. QNX should be relatively immune to this type of damage. If CHKFSYS complains, you may copy the unbusied files to another disk, zap them, run CHKFSYS to recover lost blocks, then restore files. For example.

copy 2:/file1 1:/file1

zap 2:/file1

chkfsys 2

copy 1:/file1 2:/file1

If you are crashing the system often while debugging a program, you may wish to always save, zap and restore files and only run CHKFSYS occasionally to recover bad blocks. If you are sure that no disk I/O was in progress, it is probably not even necessary to zap the files.

If the **f=** option is specified, then only the indicated pathname will be checked. The pathname must start with a slash. This is not a complete check in that it cannot check for multiple allocations of the same block to more than one file. For example:

chkfsys 1 f=/user/gord/data

will check the file 1:/user/gord/data.

In the event of the complete loss of a file system due to the destruction of the root directory and the bitmap (first few blocks of the disk) you should refer to the technical note on file recovery in your QNX manual.

See Also:

DCHECK
ZAP
QUANTUM TECHNICAL NOTE ON FILE RECOVERY

copy 1:/file1 2:/file1

If you are crashing the system often while debugging a program, you may wish to always save, zap and restore files and only run CHKFSYS occasionally to recover bad blocks. If you are sure that no disk I/O was in progress, it is probably not even necessary to zap the files.

If the **f=** option is specified, then only the indicated pathname will be checked. The pathname must start with a slash. This is not a complete check in that it cannot check for multiple allocations of the same block to more than one file. For example:

chkfsys 1 f=/user/gord/data

will check the file 1:/user/gord/data.

In the event of the complete loss of a file system due to the destruction of the root directory and the bitmap (first few blocks of the disk) you should refer to the technical note on file recovery in your QNX manual.

See Also:

DCHECK
ZAP
QUANTUM TECHNICAL NOTE ON FILE RECOVERY

In this case CHKFSYS will complain that there are blocks used in the bitmap which are in fact NOT used by any file. These blocks may be recovered by writing the reconstructed bitmap back to disk. CHKFSYS will attempt to read each of these blocks and will NOT mark bad blocks as available.

The **b=** option may be used to inform CHKFSYS of known bad (or marginal) blocks. Each line of the file should contain 1 hexadecimal block number. The block numbers must be sorted in ascending order.

If **b=** is not specified, then CHKFSYS will use the file "/bad_blks", if it exists on the drive being checked. This file is created automatically by the command:

dcheck +mark

CHKFSYS will tell you if any files are using blocks which are now known to be bad.

If CHKFSYS complains that a block is used by more than one file, it could indicate one of two problems. If the **b=** option was specified, or "/bad_blks" exists, then it probably indicates that the file uses a block which is bad and marked as used in the bitmap. If there are no known bad blocks, then a multiple allocation of a single block has occurred. In either case, the file should be saved on **ANOTHER** disk (if possible) and then zapped. CHKFSYS should then be run again to update the bitmap, after which the saved file may copied back onto the disk.

In general, whenever the bitmap is replaced, CHKFSYS should be run a second time to ensure that the file system is indeed consistent.

CHKFSYS may also be run after a system crash or power failure which has left some files busy. The files may be made unbusy using the chattr command

chattr s=-b file

which may then be followed by a CHKFSYS to ensure that no damage to the file system has occurred. QNX should be relatively immune to this type of damage. If CHKFSYS complains, you may copy the unbusied files to another disk, zap them, run CHKFSYS to recover lost blocks, then restore files. For example.

copy 2:/file1 1:/file1

zap 2:/file1

chkfsys 2

CHMOD - Change mode of a file

Description:

Please refer to the documentation on the CHATTR command.

See Also:

CHATTR
CHGRP
CHOWN

CHOWN

CHOWN - Change owner (member number) of a file

Syntax:

chown *member_number file**

Examples:

chown 22 file1 file2

Description:

CHOWN changes the member number of the specified files to *member_number*. This number must lie between 0 and 255 inclusive. Member 255 is a group leader.

Note that this is a shell file which invokes the CHATTR command.

See Also:

CHATTR
CHGRP

CLOCK - Display Time and Date On Console

Syntax:

clock [*options*]*

Options:

-date - Suppress display of the date.
-hour - Suppress display of the hour.
-seconds - Suppress display of the seconds.
-ampm - Suppress display of the AM or PM.
-key - Suppress display of keyboard shift status(s).
a=*attr* - Select display attributes (hex).

Examples:

clock a=f104 & (white on blue, then inverse)

Description:

This command will maintain a real time display of the date and time in the top right hand corner of your console display. It will also display the current shift state of the CAPS LOCK key, NUM LOCK key and the ALT key. This command automatically adjusts its priority to level 15 so that it does not compete with other tasks.

The display attribute bits are defined as follows:

|eBBBxFFF|xxxxuihb|

b - enable blink
h - enable highlight
i - enable inverse
u - enable underline
FFF - foreground colour (if enabled)
BBB - background colour (if enabled)
e - enable colour
x - don't care

CLOCK

This command is included in your "/config/sys.init" file. Should you wish to remove the feature, you may edit "/config/sys.init" removing the invocation. The command may also be killed at any time by typing the command "slay clock".

Users with battery backup-up clock/calendar hardware may wish to use the RTC command to set the date. This command would be placed in your "/config/sys.init" file.

See Also:

DATE
RTC

CLRHOUSE

CLRHOUSE - A distributed database and name server

Syntax:

clrhouse **s**tart [**-p**oll] [**p**=*poll_sec*] **&**
clrhouse **p**oll [**n**=*node*] [**+m**e]
clrhouse **l**ocate [**n**=*node*]
clrhouse **s**top **n**=*node*

Examples:

clrhouse start &
clrhouse locate n=1
clrhouse stop n=4

Description:

CLRHOUSE maintains a distributed database for the naming of tasks. Like the TIMER task it may be required for some applications. Developers should read the technical note mentioned in the SEE ALSO section below.

START

You may run CLRHOUSE on up to 3 nodes in the network. Upon starting, each task will immediately poll each node in the network for it's list of attached names. It will then go into a slow poll mode to refresh this information. This slow poll period may be changed using the **p**=*poll_sec* options. It defaults to 10 seconds. You can suppress slow polling completely by specifying the **-p**oll option. The slow polling is not the major means by which the clearinghouse keeps informed of task names in the network. Each time a name is attached or detached, the clearinghouses are immediately informed. The slow poll is a safety net to handle extraordinary error conditions which might cause it to miss an attach or detach request from a task.

POLL

You can force all the clearinghouses to refresh their information on a node by using the POLL option. If you specify a node using the **n**=*node* option then the clearinghouses will immediately poll that node for it's list of names. The option **+m**e can be used as a short form for **n**=*my_node*. If you do not specify a node then all nodes in the network will be quickly polled. This can be used to quickly update the clearinghouses after a severe network disturbance (for example, a clearinghouse node was disconnected from the network) where you believe their

data is not current. It is doubtful that this option will be needed unless you elect not to slow poll.

STOP

You may kill the clearinghouse by using the SLAY command or the STOP option to CLRHOUSE. If you use the stop option you will have to identify which clearinghouse to stop using the **n=***node* option. The command

tsk info

will list the nodes which are running a clearinghouse.

LOCATE

When a node boots over the network it inherits the list of nodes which are running a clearinghouse. If you boot from disk you will not know this information. If the clearinghouses are slow polling you can wait until you are polled (thus learning each clearinghouse one by one) or you can execute the CLRHOUSE command with the LOCATE option. This will poll each node requesting it's list of clearinghouse nodes. The first node to satisfy the request will stop the poll and update your list. In a stable network this will typically be the first node polled. You can force the poll of a specific node only by specifying the **n=***node* option.

This command would typically be placed in the **sys.init.*nn*** file of a node booting from disk.

See Also:

NAME functions in the **C** compiler binder.
Technical note on **"Attaching and Locating Task Names"**

COMM - Modem communication handler

Syntax:

comm [*options*]*

Options:

a=*modem_answer_string* - Command to force modem to answer.
b=*baud* - Specify a baud rate to try. Multiple **b**= options can be given from lowest to highest. Default is: b=300 b=1200 b=2400.
B=*default_baud* - Default baud rate to select when the modem responds with "CONNECT" and not a baud rate. By default this is 300.
c=*login_command* - Command to execute for normal login.
d=*start_delay* - Delay in seconds after the connect before starting the selected login program. The default is 2 seconds.
f=*mail_command* - Command to execute for mail transfer.
i=*modem_init_string* - Initialization string for modem.
l=*log_path* - Name of directory for log files.
m=*greeting_message* - Initial greeting message to display.
M=*greeting_file* - Initial greeting file to display.
n=*net_address* - Mail network address to display on connect.
p=*pickup_on_ring* - Number of rings required for answer. This can only be used with the **a**= option.
q=*quiet_time_limit* - Number of minutes without input before hangup.
r=*modem_ok_response* - Modem response for command acceptance.
s=*bbs_command* - Command to execute for bulletin board system.
t=*modem_timeout* - Modem timeout for response (in 50 msec tics).
-autopar - Supress automatic parity.
+ATI_2400etc - Condition COMM to support this modem.
-CTS_check - Supress checking for CTS modem status.
-dropline - Supress dropline function.
+debug - Enable output of debug messages to log file.
+hayes - Hayes compatible modem.
+locked_baudrate - Force locked baudrate to the modem.
+pass_exception - Enable passing of hangup exception.
-prompt - Supress "Please type .... " message.
-recycle - Supress modem re-initialization after 10 minutes of inactivity.

+verbose - Enable more detailed log messages.

Examples:

```
ontty 3 comm b=1200 +h i=ATS0=0| a=ATA|
ontty 3 comm c=zarcon b=2400 +h i=ATI| "m=Welcome to zarcon2"
ontty 3 comm b=300 b=1200 b=2400
```

Description:

The purpose of COMM is to make a QNX system or network accessible by phone with a modem. The reason for a special utility is that, unlike a serial link to a terminal where the link settings are known, modems tend to operate with variable link settings. COMM is able to communicate with the modem to determine these settings and to detect when the modem hangs up the phone line so it can remove any tasks created during the call. COMM performs the following functions:

1. **Baud rate detection and selection.**
   - 300, 1200, 2400, etc.

2. **Parity and bits / character detection and selection.**
   - even, odd, mark, space, none, 7 bit, 8 bit.

3. **Initial welcome messages, a welcome line and/or a welcome screen.**

4. **The command to execute after baud and parity determination (eg: LOGIN).**

5. **Removal (killing) of all created tasks upon loss of carrier (hangup).**

If possible, your modem should be configured to provide the CD (carrier detect) RS232 signal to the computer so that COMM will be able to detect the modem status (on or off line). Likewise, the DTR (data terminal ready) RS232 signal should be connected from the computer to the modem, so that COMM and other programs can force a hangup if necessary. Your modem should be programmed to force a hangup and reset itself when the computer "drops" DTR. For a Hayes modem, this command is typically "AT&D2", but check the manual for your modem since this command may vary.

After COMM has answered the phone and determined the baud rate, COMM will wait for the delay specified with the **d=***start_delay* option (default of 2 seconds). This delay exists to avoid the burst of noise characters that often occur when two modems first connect. After the delay, if a single line message was specified with the **m=***greeting_message* option, it will be displayed. COMM will then perform various checks depending upon which command line options were specified. If a mail handler was specified with the **f=***mail_command* option, COMM will look for the protocol characters that indicate an incoming mail packet is arriving and start

the specified mail handler. Similarily, if a BBS (Bulletin Board System) program was named with the **s=***bbs_command* options, a request for the activation of the BBS program will be checked for. Note that any commands specified must have full paths, for example, the LOGIN default command is "/cmds/login".

The input sequences COMM checks for are:

..<Enter>	**- Execute the LOGIN, or c=***command* command.
<Escape>	- Execute the BBS specified with **s=***bbs_command.*
Mail Protocol	- The **f=***mail_command* mail handler will be started.

If COMM determines that a human caller is present, the file specified with the **M=***greeting_file* option will be displayed, after which either the LOGIN, **c=***command* or BBS system will be started. If no special commands have been specified, LOGIN or the command specified with the **c=***command* option will be started as a son task, without any delayed checking for alternatives.

If the user doesn't enter any characters for five seconds, COMM will default to either the BBS program (if specified), or a command (specified with the **c=***command* option), or LOGIN.

As part of the baud rate determination, COMM will post an environment variable called BAUD which is set equal to the baud rate for the modem connection. This variable is useful because many high speed modems operate with the serial link between the modem and the computer at a fixed, high speed (usually 9600 or 19200 baud) and then use hardware flow control to regulate the flow of data for slower connections. Since the true baud rate cannot be determined by the stty setting, COMM must post this information as an environment variable so that the tasks started by the dial-up user can determine the true transfer rate. As an example, QCP file transfers with fast modems benefit from having a packet size of 16K, rather than 2K. If the program detects that a high baud rate is in effect, it can set the "s=16000" option on QCP to provide faster file transfers.

In addition to setting the baud rate in an environment variable, COMM can also keep a log of modem events. The **l=***log_path* option can specify either a device where modem logs are to go or the name of a directory where log files are to be kept. If *log_path* contains a '$' character, it is assumed to be a device and data is written directly to that device. If the '$' character does not appear in *log_path*, it is assumed to be a directory path, in which case a '/' and a file name of the form:

881118 ***<-- Year, Month and Day***

is appended to create a filename for the day's log. A directory of the name "/commlog" is a good choice, since the COMM_REPORT utility will use this directory by default. Note that the logging directory must be created before COMM is started. The data written to the log file or device contains various messages. For example:

```
7:53:55 am [1]$tty22, Connect 2400/MNP
7:54:00 am [1]$tty22, Executing: '/cmds/login' (Login)
8:04:55 am [1]$tty19, Length: 17:47 (Caller Hung Up)
8:07:43 am [1]$tty19, Connect 1200/None
8:07:46 am [1]$tty19, Executing: '/cmds/login' (Login)
8:21:16 am [1]$tty22, Length: 27:21 (Call Done)
8:21:52 am [1]$tty19, Length: 14:08 (Call Done)
8:21:58 am [1]$tty22, Connect 14500/PEP
8:22:00 am [1]$tty22, Executing: '/cmds/login' (Login)
```

The LOCATE and WC commands can be used to extract various facts from these logs. For example, the command:

```
locate 2400 /commlog/881118 | wc
```

could be used to determine the number of 2400 baud calls for that day.

When the user has finished his call and exited from the program he was running (with LOGOFF, D or a hangup), COMM will remove the tree of tasks created by the user. If desired, COMM can be started with the +pass_exception option so that hangup exceptions, rather than kill exceptions, are passed to the user's tasks on hangup.

For an organized shutdown of COMM, the command:

```
slay comm u=4000
```

will cause COMM to wait for the current dial-up session to complete before terminating itself. This organized shutdown has some very important uses. If a dial-up system must be taken down for maintenance, instead of waiting for a chance when all the dial-up lines are inactive, or forcibly "evicting" people from the system, setting the "shutdown" exception will cause COMM to terminate itself on each line as the user on that line hangs up. Ultimately, there will be no COMM sessions running and the system will be idle. A shell script which uses the

```
slay comm +t
```

option can loop with a timeout until all the active sessions are done before starting a backup or other maintenance oriented function. When the backup is done, the COMM sessions can be restarted. If COMM is being used with the **a=*modem_answer_string*** option, the modems will not answer incoming calls while COMM is not present.

While COMM is operating, it makes itself an administrator task so that it cannot be accidentally "killed". If necessary, the SLAY command can be used to forcibly kill COMM. Note that any applications the dial-up user is running will also be killed.

slay comm u=8000

The **q=*quiet_time*** option can be used to limit how long COMM will tolerate a user on-line without any input. Assuming **q=*15*** was specified, after detecting 15 minutes without user input, COMM would beep the user and display the message:

Please respond!

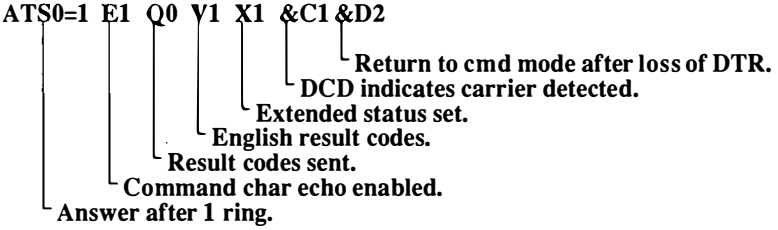
If another minute passed without user input, COMM would drop DTR, causing the modem to hangup and terminate the user session.

COMM can operate in one of two modes. The first mode is designed explicitly to work with Hayes compatible modems and the second is designed to work with non-intelligent modems that do not provide connect messages. If your modem is Hayes compatible, we recommend using COMM in the Hayes compatible mode.

Hayes Mode

The **+hayes** option configures COMM to understand the messages output by a Hayes compatible modem. COMM can be configured to initially program the Hayes modem to answer incoming calls and then wait for the CONNECT message to determine the incoming baud rate.

You can change the modem initialization string user by COMM with the the **i=** option. The default string is "ATI", but a suggested string could be:



With a 2400 baud (or better) modem, these settings can usually be programmed into the non-volatile RAM in the modem with the "AT&W" command. If so, you can set the modem initialization string to "ATZ|", causing the modem to program itself to the internally saved state. Also, the modem can usually be programmed to reset itself to the internally saved settings on loss of carrier, or hangup forced by DTR driven low by the computer.

The modem initialization string supports a number of special characters that are processed by COMM rather than being passed to the modem. These characters, and their actions, are:

| - Carriage return
\- - Ignored, usually part of a phone number
~ - 1 second delay
^ - Raise DTR
v - Drop DTR (forces a modem hangup and reset)
' - 100 millisecond delay

As the modem initialization string is processed, any characters not in this special set are passed through to the modem, unmodified. Using these special characters, modem initialization strings that perform many steps can be easily constructed. For example:

v~~~^~~~ATS0=1|

would force a hangup of the modem with DTR (with 3 second delays around each change of DTR) and then program it to answer after the first ring.

To accept incoming calls your modem need not necessarily be programmed for auto-answer. Another alternative is to provide COMM with an answer string. If the answer string is provided, the modem can be set to not auto answer (ATS0=0). What will happen is that the modem will emit "RING" statements when an incoming call is detected and COMM will detect the "RING" and provide the answer string (typically "ATA|") to force the modem to answer. This has the added advantage that your system will not answer the phone unless it is powered

up and the COMM task is ready for the incoming call. We recommend operating your modem in this mode.

If your modem answers with other than "OK" to commands, COMM can adapt to this with the **r=***modem_ok_response* command line option.

We also recommend you default your serial ports to 8 bits, no-parity and one stop bit for communication with other QNX systems.

The following are some typical invocations of COMM for a Hayes modem.

```
ontty $mdm comm +h b=2400
ontty 19 comm M=/nol/msg +h +l b=19200 i=ATS0=0| a=ATA| l=/commlog
```

Non-Hayes Mode

COMM can also handle non-intelligent, or non-Hayes mode, modems. When these modems answer the phone, COMM will begin to receive the incoming data that the remote user is typing. Starting at a previously specified baud rate (default of 2400), COMM detects this incoming data and prints the message:

Please enter two periods (..) and press <Enter>

If the baud rate is incorrect, any character typed by the user will generate a communication error causing COMM to switch its baud rate and redisplay the above message. The periods (..) and carriage return must be typed in order without any mix of other characters. Every three characters entered will redisplay the above message. Often, the sequence "ab.." seems to be most efective at generating enough exceptions for COMM to be able to determine the line settings. Space parity is treated as 8 bit, no parity by COMM.

After the caller has hung up, the baud rate is then switched back to the highest rate and the process starts over, waiting for the next call.

If your non-hayes modem has a feature to print a string when a call arrives, you should **TRY** to disable it since this may confuse COMM working in the non-hayes mode. This is especially true if the string contains either a carriage return or a period. You should also try to disable ECHO from the modem and any status messages.

This is a typical invocation of COMM for a non-intelligent modem.

```
ontty 22 comm M=/nol/msg b=300 b=1200 b=2400 l=/commlog
```

Recognizing the fact that COMM is a utility that many people would like to modify for custom applications, the source for COMM is available on the QNX Update System and is also shipped as a sample C program with the QNX C compiler. There are also a number of technical notes downloadable from the update system that detail how to configure various modems for use with QNX and COMM.

See Also:

LOGIN
LOGOFF
STTY

COPY

COPY - Copy files

Syntax:

copy *source_file destination_file* [*options*]*
OR
copy *file** *directory* [*options*]*

Options:

-date - Don't copy the date.
+pause - Pause before copying (to change diskettes).
+verbose - Display filenames as they are being copied.

Examples:

copy *.c /c_source - Copy all c files to the directory /c_source.
cp test.c $lpt - Copy file test.c to the line printer.

Description:

Copy is now identical in operation to the CP command. Please refer to the CP utility for details on it's operation.

See Also:

BACKUP
CHATTR
CP

CP

CP - Copy files

Syntax:

cp *source_file destination_file* [*options*]*
OR
cp *file** *directory* [*options*]*

Options:

-date - Don't copy the date.
+pause - Pause before copying (to change diskettes).
+verbose - Display filenames as they are being copied.
+newest - If the destination file is newer than the source, then do not copy; keep the newest of the two.

Examples:

cp *.c /c_source - Copy all c files to the directory /c_source.
cp test.c $lpt - Copy file test.c to the line printer.

Description:

The first form CP will copy a source file to a destination file. If no destination file is given, the standard output is used. If no source file is given, the standard input is used.

The second form of CP will copy multiple source files to a destination directory. This provides a convenient method of moving files in one simple operation. The SHELL special characters *, ?, and [...] are useful for selecting the set of files to be copied. If the error

LINE TOO LONG

is displayed, you will have to restrict the number of files copied and issue two or more commands.

cp *.[co] dir --> **cp *.c dir**
cp *.o dir

The following two commands achieve the same effect using the two forms

```
cp 2:/doc/report 1:/doc_backup/report
cp 2:/doc/report 1:/doc_backup
```

A dash (-) may be used to force the source input from standard input, or to force output to standard output.

CP copies as much of the source file as it can into memory, then flushes the memory buffer to the destination file. This is repeated until the entire file is copied.

The **+p** option allows the user to change diskettes (in a floppy only system) before actually transferring the files (so that the destination diskette can be inserted into drive 1 for example, where the commands disk which contains the CP command usually resides). CP will prompt the user for a carriage return before initiating the copy operation when **+p** is specified.

The **-d** option will use todays date for the destination file. The default is to copy the date from the source file to the destination file. The default form of the CP command requires that the destination file have "modify" permission, since the date of the destination file is being modified. If modify permission is missing, you will have to use the **-date** option.

See Also:

BACKUP
CHATTR

CRON

CRON - Schedule commands in background

Syntax:

cron [-logfile] [l=*logfile*] [t=*tablefile*] **&+**

Options:

-logfile	- Do not append status information to any log file.
l=*logfile*	- Use the indicated log file in place of the default logfile **/config/cronlog.**
t=*tablefile*	- Use the indicated table file in place of the default table file **/config/crontab.**

Examples:

timer &
cron -l t=/etc/crontab &+

Description:

The CRON command provides a convenient method for executing commands at specific times without operator intervention. It requires that the timer administrator be running in background.

Cron is a resident program that uses a list of times and program names contained in the file /config/crontab. This file has a single line entry for each program to be executed that contains the following six fields:

1. The minute in the hour	**(0-59).**
2. The hour of the day	**(24 hour clock).**
3. The day of the month	**(1-31).**
4. The month of the year	**(1-12) OR (a 3 letter monthname).**
5. The day of the week	**(0-6 where 0 is Sunday) OR (a 3 letter dayname).**

6. A command to be executed in the background

Any of the time fields may have:

a single value	**3**	**exact**
an upper and lower boundary	**10-15**	**range**
a series of values	**2,5,11,33**	**list**
a "*" to mean all values	*****	**all**

Each element of a list may of course be a range. The following examples illustrate some typical cron table lines. The indented comment text is NOT part of the cron table file.

15 * * * 1-5 who >>/etc/wholog
OR
15 * * * mon-fri who >>/etc/wholog

At 15 minutes after the hour on Monday thru Friday append the output of the who command to the file /etc/wholog.

0 1 * * * /cmds/diskuse

At 1 AM each day run the shell command file diskuse.

0 4 1,15 * * payroll_startup

At 4 AM on the first and the fifteenth run the program payroll_startup.

55 7 * * 1,2,3,4,5 ontty 3 comm

At 7:55 AM Monday thru Friday start the dialup access routine. This assumes a reboot every morning or multiple COMM programs will start to stack up.

0 0 1 jan * mail s %users <newyear_msg

On the first of january every year send mail to everyone in the file users.

The command which is scheduled to run will be run in **background**. Its standard input will be the device **$null** and its standard output and error will point to the console on which CRON was run in background.

Each time the cron table is read by CRON, or CRON schedules a program to run, a status message is appended to the cron logfile unless the **-l**ogfile option was specified. It is possible to run a command from CRON which prints then deletes this file once a week.

If a time in the cron table has already passed when CRON is started, that command will not be run. CRON reads the cron table file *once* (within 1 minute) when it is started. If you modify the table file you may force CRON to reread this file by setting a break exception on CRON. For example.

slay cron +break

CRYPT

CRYPT - Encrypt/Decrypt files

Syntax:

crypt [*key*] *<input_file >output_file*

Options:

key - Encryption key to use.
input_file - Source file to encrypt.
output_file - File to place encrypted output.

Examples:

crypt secret <table >table.x
ws "crypt qwerty <@ >@.x" p=*.c - Encrypt all C files in current directory

Description:

CRYPT reads from the standard input and writes an encrypted version of the data to the standard output. The KEY selects a particular transformation of the data. If no key is specified, CRYPT will prompt for a key from the terminal (without echoing it). CRYPT encrypts and decrypts with the same key. After executing these two commands

crypt abc <file1 >file2
crypt abc <file2 >file3

file3 will be identical to **file1**.

CRYPT implements a one rotor machine with a 512 byte rotor. Such an encryption scheme *can* be broken, however, the amount of work required is likely to be large and not generally known.

Since the encryption relies heavily on the key, small keys (3 characters or so) should be avoided.

CRYPT flushes all input to prevent users from recalling command lines in order to redisplay the key.

DATE - Display or set date and time

Syntax:

date [*day* [*month* [*year* [*hour* [*min* [*sec*]]]]]] [**am**|**pm**]

Options:

day - 1-32
month - 1-12 or jan-dec
year - 80-99 or 1980-99
hour - 1-12 or 0-23
min - 0-59
sec - 0-59
am - (used with 12 hour time)
pm - (used with 12 hour time)

Examples:

date
date 22 2 82
date 16 july 1982 4 30 pm

Description:

If no arguments are given, the DATE command allows the user to display the date and time on his terminal. When changing the date, day is entered first followed optionally by month and year. The current time can also be set by following the date with the hour, minute, and second. 24 hour or 12 hour time (am or pm) can be used. All arguments must be separated by spaces. Any missing arguments will default to the current value.

For details of setting the QNX date from a battery backed-up clock/calendar, see the RTC command.

See Also:

CLOCK RTC

DCHECK

DCHECK - Check a disk for bad blocks

Syntax:

dcheck *drive* [*options*]*

Options:

b=*num_blks*	- Number of blocks (in decimal) to check (default: all blocks).
+mark	- Mark bad blocks as used in the bitmap.
-verbose	- Don't display progress information.
+verbose	- Display every bad block on the disk.
-pause	- Don't wait for a disk to be inserted.
f=first_blk	- Start the check at this block number.

Examples:

dcheck 2 +m	- Check all blocks on disk 2 and mark bad blocks in the bitmap
dcheck 1 b=640	- Check first 640 blocks on disk 1
dcheck [3]1	- Check all blocks on disk 1 of node 3

Description:

DCHECK allows the user to verify that a disk has been formatted properly. The utility will do this by attempting to read every block on the disk. Any blocks which cannot be read will be displayed on the standard output (which may be redirected to a file). Bad block numbers are displayed in hex. A summary of the number of bad blocks will always be printed on the terminal.

If the number of blocks to check is not specified, then DCHECK will obtain the correct number from the file system and check ALL blocks on that disk.

DCHECK can be used to check any formatted disk, including disks which contain files. **These files will not be damaged.** If the disk has been initialized using DINIT, the **+mark** option should be used to remove any bad blocks from the disk allocation bitmap. This is especially true for hard disks.

IMPORTANT: DCHECK with the the **+mark** option should only be used when the system is idle. There should be NO open files when DCHECK is running.

When **mark** is specified, DCHECK will attempt to read the file /bad_blks from that disk. If found, DCHECK will read in the file which contains a list of all *known* bad blocks, in sorted order. When DCHECK finishes, it will re-create the file /bad_blks, after it has updated the bitmap. DCHECK will only *add* to this file, allowing *transient* or *marginal* disk blocks to be detected, and avoided, by repeatedly calling DCHECK:

```
dcheck 3 +mark
dcheck 3 +mark
etc.
```

The /bad_blks file is also recognized by the CHKFSYS utility.

DCHECK may be used to check a disk on another node. The disk may be indicated explicitly via a [*node*] prefix or implicitly by specifying a *remdisk* which you have mounted.

The file created by redirecting the output of DCHECK may be used as a data file for the CHKFSYS command. This allows you to detect which file (if any) contains bad blocks.

```
dcheck 3 >bad_blocks
chkfsys 3 b=bad_blocks
```

See Also:

CHKFSYS
DINIT
FDFORMAT
MOUNT

DCOPY

DCOPY - Copy entire disk to a second disk

Syntax:

dcopy [*source* [*destination*]] [*options*]*

Options:

source	- Source drive (default: 1).
destination	- Destination drive (default: 2).
+hard	- Allow copying to a hard disk.
-verify	- Don't verify after copying.
k=*size*	- Size of disk to copy in Kbytes (default: ALL blocks).
b=*nblocks*	- Size of disk to copy in blocks.
so=*offset*	- Offset on source disk in blocks.
do=*offset*	- Offset on destination disk in blocks.
+repeat	- Continue copying until user types BREAK.
-pause	- Don't pause for non-floppy disks.

Examples:

dcopy 1 2	- Copy all of disk 1 to disk 2.
dcopy 1 1	- Copy a disk using a single drive.
dcopy -v +r	- Make multiple copies of disk 1 without verifying.
dcopy 2 1 b=160	- Copy disk 2 to 1. Only copy 160Kbytes.
dcopy 3 1 so=640 k=320	- Copy the second 320Kbytes of drive 3 to drive 1.
dcopy 1 3 b=640 do=640 +h	- Restore the second 320Kbytes of drive 3 from drive 1.
dcopy [4]1 [6]1	- Copy disk 1 on node 4 to disk 1 on node 6.

Description:

DCOPY allows the user to make an *exact* copy of the diskette in the source drive (default 1) onto the diskette in the destination drive (default 2).

CAUTION: This utility is NOT meant for copying *files*. It is meant for copying exact disk *images*. Use CP, or BACKUP for copying *files*.

DCHECK will prompt the user to load the diskettes into the appropriate drive and type CR when ready. DCOPY will read as many blocks as it can from the source drive into memory, then write them onto the second drive. This will be repeated until the entire diskette has been copied.

Normally the data on the destination drive will be read back and compared with the memory buffer to ensure that the data has been copied correctly. This can be suppressed (to speed up disk copying) with the **-v** option.

The **b=** and **k=** options allows the user to make copies of only a portion of a disk. The **+r** option allows multiple disk copies to be made without the need to key in the DCOPY command every time. The **so=** option permits the first part of the source disk to be ignored. Offset is specified in blocks. The **do=** option allows an offset to be specified on the destination drive.

Note that a blank floppy diskette must be formatted before it can be copied to.

Although DCOPY is normally used to copy floppy diskettes, a ramdisk or a hard disk can also be copied to/from floppies. The **k=**, **so=**, and **do=** options allow an entire Hard disk to be copied onto several floppy diskettes. Command files consisting of several DCOPY commands could be built to make this more convenient. The **-p** option can be used to avoid the prompt message which results when one of the disks is not a floppy.

For your protection, copying to a hard disk is not allowed unless you specify the **+hard** option. This should protect against any typing errors.

You may DCOPY to disks on different machines. This may may be indicated explicitly via a [*node*] prefix or implicitly by specifying a *remdisk* which you have mounted.

See Also:

BACKUP
FDFORMAT
MOUNT

DDUMP

DDUMP - Dump the contents of a disk block

Syntax:

ddump *drive block* [**-**verbose]

Options:

drive - Disk drive number (1..F).
block - Disk block number in hexadecimal (1..?).
-verbose - Don't display as ASCII characters

Examples:

ddump 3 1 - Dump block 1 of disk 3
ddump 1 10A -v - Dump block 10A (hex) of disk 1
ddump [4]3 2 - Dump block 2 of disk 3 on node 4

Description:

DDUMP will display the contents of a disk block 16 hexadecimal bytes per line, with the ASCII equivalent (if printable) displayed to the right of the line. If the ASCII character is not printable, it is replaced with a dot (.).

The disk block is specified as a hexadecimal number.

Subsequent disk blocks can be displayed by typing *CR* for each new block. Any other typed character will terminate DDUMP.

DDUMP may be used to dump a disk on another node. The disk may be indicated explicitly via a [*node*] prefix or implicitly by specifying a *remdisk* which you have mounted.

See Also:

MOUNT
NACC

DEBUG - Invoke the system debugger

Syntax:

debug [*command line*]

Examples:

debug - Drop into the debugger.
debug test arg1 >$lpt - Debug the program "test"

Description:

If you wish to do source level debugging please refer to QDB in the C compiler manual. This is a low level debugger suitable for debugging interrupt handlers.

DEBUG allows the system debugger to be used, provided that it has been first mounted into the system. This should only be done once.

mount debug

The debugger is an absolute (hex) debugger which allows breakpoints to be set in user programs, memory to be displayed and edited, code to be disassembled, and I/O ports to be examined.

The debugger can only be invoked from the console. The debugger should never be used in a multi-user environment. It disables interrupts and freezes the entire system. It may be used to debug application programs as well as interrupt routines.

With no arguments, DEBUG will be entered with a 4K memory buffer at its disposal. If a command line follows the keyword **debug**, then that command will be loaded into memory before the debugger is called. In this case, the debugger will have its default code and data segments initialized to those of the command being debugged.

Breakpoints can be set on functions whose addresses have been determined from the map which can be generated by the linker.

cc test m=.map.
cc x=ofiles m=.map.

Similarly, the addresses of memory variables can be determined from the linker map and displayed in the debugger.

The debugger does not support line editing. If you make a typing mistake you may cancel the entire line by typing a Ctrl c. When entering addresses, etc. all data is assumed to be hexadecimal. Only the last 4 digits are valid (2 for byte data) so typing mistakes can be corrected by continuing to type the number such that the last 4 digits are the ones you want.

1b341d44 results in 1d44

If you type a simple DEBUG, perhaps to examine an I/O port, you may return back to the shell by typing the **g** command. If you debug a command, then your default code (@ @) and data (@) segment will be set up to point to your program. Note that the CS register displayed when you first enter debug will be pointing into the shared library, NOT your program. All breakpoints will be placed in your real code segment which the shared library will return to when you resume via the **g** command.

A WORD OF WARNING. If your program allocates considerable memory via the ALLOC, CALLOC, MALLOC or OPEN routines, then QNX may relocate your data segment without updating the default data segment memorized by the debugger when your program began execution. As a rule of thumb, the stack segment (**ss:**) will be your real data segment which should agree with the value printed for **ds=** when you type **?**. If not, you may change your default data segment using the **@** command. Note that the default data segment is ONLY used for displaying, examining and changing memory. Changing it does not affect the data segment of your program. It is there to save you from having to type in a segment prefix. When debugging multiple tasks, you may wish to change your default code and data segment many times for convenience.

Debug Command Summary

b *address*	Set breakpoint at *address*
b *seg*:*address*	
b r	Set breakpoint at return from C function
b	Display breakpoints

c *start finish destination*	Copy memory
c *seg*:*start finish seg*:*dest*	

d *address*	Display memory. CR ends display
d *seg*:*address*	and SPACE displays another line

D *address*	Disassemble memory
D *seg*:*address*	

e *address*	Edit Memory
e *seg*:*address*	

EDIT commands

hex number	Changes contents of memory
'c	Changes to ascii character *c*
CR	Leaves edit
SPACE	Goes to next byte
=	Redisplays this byte
^	Displays previous address
.	Displays 8 bit IO port
,	Displays 16 bit IO port
:data	Output to 8 bit IO port
;data	Output to 16 bit IO port
r	Goes to short relative address
R	Goes to long (16 bit) relative address
/	Goes to the address found in memory

f *seg*:*start finish data*	Fill memory with data
f *start finish data*	

g *address*	Go to *address* (begin execution)
g *seg*:*address*	
g	Continue execution (or exit debug)

=	Continue execution, but reset a breakpoint at the same address
l *start finish data* **l** *seg*:*start finish data*	Locate data in a range. Up to 8 bytes may be searched for.
r *reg*	Examine contents of register reg : ax bx cx dx si di bp es ds cs ss
r *reg data*	Change contents of register
s	Single step from last instruction. You may type the space bar to step through your code, one instruction at a time.
s *count*	Execute *count* instructions or until next breakpoint
t *n*	Display last *n* C stackframes
T *n*	Display last *n* instructions executed while single stepping
u *address*	Unbreak (remove break) at this *address*
u	Remove all breakpoints
V	Trap all Protected-mode interrupts.
?	Display registers
@ *segment*	Define default data segment used by **d e f c** commands
@ @ *segment*	Define default code segment used by **D b u s g** commands
@ i *mask*	Define interrupt *mask* which will be restored when program resumes.
@ p *data*	Change stack pointer
@ s *data*	Change stack segment

See Also:

LINK
MOUNT
QDB *Symbolic Debugger*

DEF_SERVER

DEF_SERVER - Define id of global name server

Syntax:

def_server *node task_id* [+local]

Options:

node - Node number where the server task resides.
task_id - Task-id of the server task.
+local - Only change server on local node.

Examples:

def_server 3 1 - make TASK_ADMIN on node 3 a global name server.

Description:

The DEF_SERVER command defines the *node* and *task_id* of a global name server task. This may be a user defined task or a TASK ADMINISTRATOR on any node. Global names are used by administrator tasks which provide network wide services. The SPOOLER is an example of such a task. This command may be executed on any machine and all machines in the network will be updated with the information.

DEF_SERVER will attempt to update the *global* name server information on every node in the network, unless the **+local** option is used. If it is, then only your node will change its value for global name server.

This command is only useful with the networking version of QNX.

The default name server in a QNX network is task-id 0001 on node 1. Most users will NOT need to change this.

See Also:

CLRHOUSE
SPOOL
TSK

DEFPIPE - Define temp files for pipes

Syntax:

defpipe *path*

• *This is a local shell command* •

Description:

The DEFPIPE command allows you to change the pathname where temporary pipe files are placed.

defpipe my_pipe

The default is

defpipe /tmp/..#n#$

DIFF

DIFF - Difference between two files

Syntax:

diff *file1 file2* [*options*]*

Options:

+editor - Creates a script of line edit commands which can convert *file1* into *file2*. (see the LED command)

+matching - Produces a script of matching lines in the two files rather than the difference.

+raw - Causes a character by character comparison of the two files. A file match or mismatch statement will be issued if necessary indicating the characters (and character number) where the mismatch occurred. The **-v** option may not be used with **+raw**. This option must be used when comparing binary files such as commands.

-blanks - Causes multiple blanks or white characters to be ignored in line comparisons.

-case - Causes case distinction to be ignored.

+heading - Displays the names of the files being compared. This may be useful without any other option or the **+m** and **+r** options.

-null - Causes null lines to be ignored. This option should not be used with the **+e** or **+r** options.

r=*resync* - Changes the minimum number of lines required to match before the two files resynchronize. The default is two, the new value must between one and one hundred.

s=*max_diff* - Changes the maximum number of lines the files may differ by at any time before being declared incomparable. The default is one hundred.

t=*string* - Define terminator string to print between sections of text.

-verbose - Disables the "The two files compared equal" message.

-write - Suppress "w" command in LED script created by the +e option.

Examples:

diff version1 version2
- Creates a meaningful script of differences between the files.

diff old new +e >chgs
- Compares the original file to the new version creating changes which will convert the old file into the new version. To recreate the new version use:

led old <chgs

diff core1 core2 +r
- Allows text or non-text files to be compared character by character.

diff file1 file2 +m
- Displays the similar text segments between the given files.

Description:

DIFF

DIFF compares two files line by line searching for differences between them. It can illustrate these differences by producing a meaningful script of edit-like commands. The commands are of the form:

range1 C range2
[< *file1_text*]*
[> *file2_text*]*

Where:
range1 is line number range in file 1
range2 is line number range in file 2
C is the LED command **a**ppend, **d**elete or **c**hange
< indicates a line of text affected in file1
> indicates a line of text affected in file2

In many situations, several versions of a file may exist. You may reduce file space requirements by saving diff files rather than multiple similar copies. When you update a file, DIFF **+e** can summarize the session as a series of line editor commands. Typically a file containing the LED commands will be much smaller than the updated version of an original file. Thus, if several versions of a file are necessary, a great deal of disk space can be saved by keeping one original file and several difference files.

If similarities between the files are required, the **+m** option will display the files corresponding line numbers followed by similar lines of text.

When it is necessary to compare files character by character (binary files), instead of as lines of text, use the **+r** option. The output will state if the files are equal. If not, the character position and corresponding characters will be given.

See Also:

LED
LOCATE

DINIT - Diskette Initialization

Syntax:

dinit *drive* [**+h**ard] [**k**=*size*] [**p**=*/prefix*] [**+s**uppress] [**-p**ause]

Options:

+hard - Required if the disk is a hard disk.
k=*size* - Size of diskette in Kbytes.
p=*/prefix* - Name to give the root of the file system on that disk (default is "/")
-pause - Don't pause for diskette to be loaded.
+suppress - Suppress the writing of the root directory.

Examples:

dinit 3 +hard - Initialize a hard disk in drive 3.
dinit 1 - Initialize a disk in drive 1.
dinit 1 p=/user - Initialize disk in drive 1 to have a root name of "/user".
dinit 5 -p - Initialize disk in drive 5 without pausing (perhaps a ramdisk).
dinit [3]1 - Initialize disk in drive 1 on node 3.

Description:

DINIT will initialize a formatted diskette so that it may be used in the QNX file system environment. The default values are determined from the current configuration of the specified drive. This configuration can be changed with the MOUNT command.

The prefix (if specified) is the name of the "root" of the file structure on that drive. The default prefix is "/" which allows any directory names to be placed at the highest directory level on that diskette (eg. the drive could contain the directories /cmds, /lib, /expl, etc...). If a diskette is known only to contain one directory at the highest level (eg. /user), then this prefix can be given. The prefix must start with a slash (/). In practice, this option is rarely used.

A diskette which is given a prefix other than "/" cannot be accessed without giving the exact name of the disk. A user could name a diskette with a unique name known only to himself which would prevent other users from looking at the files on that diskette.

The **-p** option can be used to bypass the prompting for a diskette to be loaded.

In the case of a hard disk the DINIT should be followed by a DCHECK to remove any bad blocks from the disk allocation bitmap.

```
dinit 3 +hard
dcheck 3 +mark
```

The **+hard** option is required if the disk is not a floppy or ramdisk. This option is there to protect you against accidental typing errors which might attempt to DINIT your hard disk.

The **+suppress** option is only used after a disaster which has destroyed the first few blocks of your disk. Please read the technical note on "Recovering Your Hard Disk".

The **k=** option will rarely be required. It may be used to create a bitmap which is smaller than the physical disk. This will prevent QNX from touching blocks at the end of the disk.

```
dinit 1 k=252    - 28 tracks * 2 heads * 9 sectors
                   reserve inner 12 tracks
```

DINIT may be used to initialize a disk on another node. The disk may be indicated explicitly via a [*node*] prefix or implicitly by specifying a *remdisk* which you have mounted.

See Also:

DCHECK MKDIR
FDFORMAT MOUNT

DIR - Display directory tree

Syntax:

dir [*directory*] [*options*]*

Options:

+count - Display a count of the number of files in each directory.

-files - List only directories, not files.

+modified - Display any directories that contain modified files with the enhanced video attribute (or with a preceding '*' character for output devices without enhanced mode). The only files that will be listed while this option is in effect will be those that are modified. Thus, a command like

```
dir 3:/ +m
```

will create a display of the complete directory tree and only those files that have been modified since the last backup will appear within the tree.

-sort - Do not sort the output.

d=*max_dirs* - Maximum number of sub directories allowed in a single directory. The program will halt if there are too many. The default is 60.

f=*max_files* - Sets the maximum number of files that will be printed within a single directory. The error message "too many files" will appear if more are present. The default is 650.

l=*levels* - Limit the depth of the directory tree. Default is 9999.

p=[^]*pattern* - Specify a pattern to select which files are listed. This pattern match facility applies only to file names and not to directory names and is applied only to files that make it through the flag settings, such as the **+m**odified flag. The '^' character can be used to negate the pattern match.

m=*memory* - Amount of memory (bytes) used to store directory and file names while processing. The default is 25 Kbytes.

w=*line_width* - This option can be used to manually specify the output width. The default width is 80. This is useful when printing a directory with **+f**iles to a 132 column printer.

Examples:

```
dir
dir 3:/ w=132 >$lpt
dir +m +c l=2
```

Description:

This utility displays a hierarchical view of the directory system. If no directory is named, the directory tree from the current directory down is displayed.

The output of this command uses the tcap library for the line drawing characters and can thus be used to print directory trees on the printer or attached terminals as well as on the screen.

See Also:

CD	LS
FILES	PWD

DMARK - Mark bad blocks on a disk

Syntax:

mark *drive* [*cylinder,head*]*

Options:

drive	- Which QNX drive is to be marked.
cylinder	- Which Cylinder is bad. Values range from 0 to T-1.
head	- Which head, or platter is bad. Values range from 0 to H-1.

Examples:

dmark 3 40,4 250,2 500,0 - mark track 40, head 4
and track 250, head 2
and track 500, head 0

dmark 3 20,1 >3:/bad_blks

Description:

If your hard disk was shipped with a sheet of paper containing a list of bad tracks, then you may use the DMARK command to explicitly mark the blocks on these tracks as un-usable.

DMARK will display a list of QNX block numbers on the screen corresponding to the bad blocks. This may be re-directed into a file, which can be used by CHKFSYS to update the **bitmap** on that disk.

If DMARK is used immediately after using DINIT on a *new* hard disk, you could direct the output into the file /bad_blks on that disk. The DCHECK and CHKFSYS utilities will then maintain that file as new bad blocks are discovered.

The DMARK command takes a list of cylinders and heads. Each pair is separated by a space, with the cylinder and head separated by a comma.

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

DMARK

DMARK is best used in conjunction with DCHECK, which will find any blocks which have failed since the disk was originally manufactured.

See Also:

CHKFSYS
DCHECK
DINIT
FDFORMAT
MOUNT

DOPEN - Display open devices

Syntax:

dopen [*node*]

Options:

node - The node whose devices you wish to query. The default is your local node.

Return Value:

Number of devices open
-1 on a critical error.

Description:

DOPEN will print out the names of any tasks which have a device open for read. It does **not** list tasks which have a device open for write only. This command is frequently used to determine who has the modem port open.

DOPEN only lists the devices open on one node at a time. It defaults to your current node. You can specify a node using the *node* option.

dopen 4

See Also:

FOPEN

DREL

DREL - Release Directory

Syntax:

drel *directory*

Examples:

drel /test
drel /user/bill/cat1
drel 2:/user
drel [2]3:/user

Description:

DREL will remove a directory from the disk. Only empty directories can be released. If the DREL command verifies that the directory is empty, then the space used by the directory will be reclaimed.

If a directory is to be removed which is at the top level (root), it is usually necessary to specify the correct drive prefix as in the last two examples.

See Also:

FREL
RM
RMDIR

DUMP - Dump the contents of a file in hexadecimal format

Syntax:

dump *file* [*start_offset* [*end_offset*]]

Options:

start_offset	- First byte to display (decimal) default: 0.
end_offset	- Last byte to display (decimal) default: last byte of the file.

Examples:

dump test.o	- Dump an entire file.
dump core 0 120	- Dump the first 121 bytes of "core".
dump $tty3	- Dump data from $tty3 (At least 80 chars need to be sent before being displayed).

Description:

DUMP will display the exact contents of a file, 16 bytes per line (hexadecimal), with the ASCII equivalent (if printable) displayed to the right of the line. If the ASCII character is non-printable, it is replaced with a dot (.).

DUMP is useful for displaying non-text files such as object files and core files.

The start and end offsets are specified as decimal numbers.

See Also:

P
SPATCH

DUMP_IBM - Make a hardcopy of Graphics Screen

Syntax:

dump_ibm [*options*]*

Options:

device_name	-	Name of printer to use if not $lpt.
+h*igh_res*	-	dump high resolution screen (640 x 480)
+m*edium_res*	-	dump medium resolution screen

Examples:

dump_ibm	-	Make hardcopy of graphics screen.
dump_ibm [7]$lpt	-	Make hardcopy on node 7's printer.

Description:

This command will make a hardcopy of the IBM colour graphics screen onto an IBM compatible graphics printer.

It is typically called from other programs (such as BAR) to print the graphics screen since issuing the command by hand would alter the display.

See Also:

BAR
MOUNT

EC - Execute a shell file

Syntax:

ec *shell_file_name*

• *This is a local shell command* •

Description:

The EC command causes the shell to temporarily take its input from the indicated file. Since the execution of commands in the file is performed by the current shell, it is possible to execute commands like CD, PATH, PRI, PROMPTT etc..., which affect the environment of the shell which executes them.

This is especially useful in the password file where you will often see a line such as

ec user.init

EC commands cannot be nested.

See Also:

SH

ECHO

ECHO - Echo arguments to standard output

Syntax:

echo [*arguments*]*

Examples:

echo Hello world!!!
echo abc >$tty3

Description:

ECHO will echo its arguments to the terminal. ECHO is used within command files to display progress information. The last argument will be followed by a carriage-return/line-feed.

Unlike TYPE and STYPE, the output of ECHO can be redirected since it is a executable program and not a SHELL command.

See Also:

STYPE
TYPE

EO - Execute On

Syntax:

eo [*file*] "*command*" [*options*]*

Options:

+concat	- Each line in the input file is concatenated and separated by a space. The indicated command is called only once with multiple arguments rather than many times with a single argument
-error	- Don't stop if a command returns back a non-zero status. This includes a command stopped via break.
o=*offset*	- Skip this many characters at beginning of each line before executing.
p=*pattern*	- Skip any line which does not match this pattern (before applying o=).
+repeat	- This is only used in conjunction with the +concat option. If during concatenation of the input lines into multiple arguments the maximum line size of 256 characters is exceeded, then the command will be executed with as many items on the line which will fit. This process is repeated until there are no input lines left or line greater than 256 characters is read.
+verbose	- Display commands as they are being executed.
dc=*del_char*	- This character when present in *command* will delete the last character of the current. pathname (default is a backquote)
pc=*path_char*	- This character when present in *command* will be replaced by the current pathname. If omitted, then the argument will be appended after the command. Note that the *path_char* is consumed by the first argument when used with the +concat option. (default is an at-sign @)

Examples:

eo my_files "list -b" +c

```
                                    - List all files named in my_files
eo cfiles "ed @" +v
                                    - Edit all files named in cfiles
files -v p=*.o | eo "frel" +c +r
                                    - Release all files ending ".o"
```

Description:

EO, like WS, is a utility which increases the power of existing QNX commands. It takes a command and a file containing a list of filenames with one filename per line. The command will be executed once for each line in the specified file, with the line being appended as an argument to the command. Given a set of filenames, EO allows a convenient mechanism of executing a command across all those files. The file list will typically be generated using the editor or from the output of the FILES command.

```
files >cfiles p=*.c -v
eo cfiles "crypt key <@ >@.z"
```

If the *file* of filenames is omitted, then the filenames will be read from the standard input. This allows EO to be used as a filter in a pipe. The above command could be shortened to a single step as

```
files p=*.c -v | eo "crypt key <@ >@.z"
```

Some commands are capable of accepting multiple arguments. In these cases it is more efficient (and sometimes necessary) to execute the command once with many arguments rather than many times with one argument. The **+concat** option will take each input line and concatenate them together, separated by spaces. The command will then only be executed once. For example

```
eo my_files "size" +c
```

The **+repeat** option handles the case where the number of input lines when concatenated will exceed the maximum line size of 512. In this case the command will be executed with as many items on the line which will fit. This process is repeated until there are no input lines left, or a line greater than 512 characters is read.

See Also:

FILES
WS

EXPL - Explain

Syntax:

expl [*command* [*subcommand*]*] [*options*]*

Options:

+qnx	- Force QNX display character set and escape sequences to be used.
m=*menu*	- Go immediately to indicated menu item.
p=*lines*	- Pause after this many lines. default is 24. p=0 disables pausing.

Examples:

```
expl inform m=B
expl led command summary p=0 >$lpt
```

Description:

The EXPL command provides a convenient method of indexing into the directory of files found under the directory "/expl".

The explain files provide detailed information about commands. If the required explain file cannot be found, but a file called "index" is present in "/expl", EXPL will ask if an explain index is wanted. If the user responds with 'y', then the file /expl/index is printed (if it exists). Typing "expl index" on the command line will also print this file.

The disk containing the directory "/expl" must be inserted into one of the disk drives before using the EXPL command. EXPL searches for an explain command as follows.

expl mail	
/expl/mail	***look here first***
/cmds/mail.hlp	***then here***
/expl/mail/expl	***then here***
/expl/index	***look here last***

In many instances, the user may only wish to be reminded of the syntax of a command. All QNX commands explain their own syntax when the command is followed by a single question mark.

expl ?

You may use lines containing a formfeed (Ctrl-L) to cause EXPL to pause for input. Upon receiving a carriage return the screen will be cleared and EXPL will continue.

The expl command will strip all escape sequences which apply to the system console when output is redirected. The **+qnx** option can be used to force the sequences to be printed when the user is logged in to QNX via a modem on a PC using QTALK.

It is possible to construct a file which implements a single level menu using EXPL. The file will consist of a menu followed by two formfeeds on a line by themselves. Following this will be one section terminated by two formfeeds for each menu section. Each section may contain several single formfeeds to cause EXPL to pause at appropriate places. A formfeed may be entered using ED by typing a Ctrl-L.

FF - **one formfeed**
text
. - **this section contains a menu**
FF FF - **two formfeeds**
text
. - **menu item '1' or 'a' or 'A' or F1**
FF FF - **two formfeeds**
text
. - **menu item '2' or 'b' or 'B' or F2**

Please refer to the file **/expl/inform** for an example of a menu driven explain file.

FBACKUP

FBACKUP - Archive files to floppy disk(s)

Syntax:

fbackup [*drive*] **IN**it max-numr-files [**c**=360K | 720K | 1.2M | 1.4M]
["**v**=*vol_name*"]
[**f**=*format_command*] [**-f**ormat]

fbackup [*drive*] **FI**les [*arch_dir*] [+summary] [+|-**v**erbose] [*options*]*

fbackup [*drive*] **NA**me

fbackup [*drive*] **SA**ve save_spec* [**+all**] [**-clr**] [**l**=*levels*] [*options*]*

fbackup [*drive*] **RE**store [*disk_dir*[,*arch_dir*]] [**-c**reate] [*options*]*

fbackup [*drive*] **VE**rify [*disk_dir*[,*arch_dir*]] [*options*]*

save_spec: *disk_dir*[,*arch_dir*] **x**=*index_file filename*[,*arch_dir*]

options: **+p**ause **-v**erbose **-m**ulti_sector **+l**ist-only **+w**rite
pf=[]*file_pattern* **pd**=[]*dir_pattern* **pp**=[]*path_pattern*
+before **d**=*dd-mm-yy* **t**=*hh:mm:ss* (Use digits)
+Force (allow use of non-floppy disks)
g=*group* **m**=*member*
e=*error_file*
"**v**=*volume_name*" (Use quotes if name contains spaces)

Description:

The FBACKUP command is used to archive **large** files to one or more floppy disks. This command was created to solve the need for saving files which are larger than a floppy disk (such as files used in most data base applications. Each time a file is saved it is appended to the end of the archive which may span many diskettes. Earlier versions of the same file will NOT be overwritten. You may restore any version of a file on the archive.

The FBACKUP command should not be used as a general substitute for the BACKUP command. Performing an archive of all files which changed each day will rapidly consume large quantities of disks due to duplication.

When saving a smaller number of files, FBACKUP is dramatically faster than BACKUP and does not suffer the problems of disk overflow. FBACKUP will prompt for new disks as required. If FBACKUP is being used with the +Force option that allows archiving to hard disks, the CRON utility should also be investigated, as CRON can cause this backup to automatically occur overnight.

There are many options for this command and to properly use it, the *Floppy and Tape Backup* technical note at the back of this manual should be read.

See Also:

BACKUP CRON TBACKUP
Floppy and Tape Backup Technical Note

FDFORMAT

FDFORMAT - Format floppy diskettes

Syntax:

fdformat *drive* [*options*]*

Options:

f=*first_track*	- First track to format (default: 0).
h=*heads*	- Number of heads (default 2).
l=*last_track*	- Last track to format (default: last track on drive).
n=*sectors/track*	- Number of sectors per track (8, 9, 10, ...).
s=*stagger*	- Stagger factor (sector interleave) default stagger is 3 for n=8, 4 for n=9,10, 2 for n=15.
t=*tracks*	- Number of tracks on diskette (40 or 80 for diskettes).
-stagger	- Suppress the staggering of sectors.
+patch	- Write the diskette size information indicated by **h=, n=, t=** onto block one of the diskette.
-pause	- Suppress the prompt to load the disk.
+other	- Not a QNX style disk ("other") - do not put the QNX size information in block 1.
b=sector_base	- block number to start sectors at. Usually 1.
+hard	- Allow formatting of an AT hard disk.
+abort	- Abort on error.
+360k	- Assume 360K floppy (h=2, n=9, t=40).
+720k	- Assume 720K floppy (h=2, n=9, t=80).
+1.2meg	- Assume 1.2M floppy (h=2, n=15, t=80).
+1.4meg	- Assume 1.4M floppy (h=2, n=18, t=80).

Examples:

fdformat 2	- Format diskette in drive 2 using defaults.
fdformat 1 +1.2	- Format 1.2M diskette in drive 1

overriding defaults.

fdformat [3]2 t=80 - Format 80 track diskette in drive 2 of node 3.

fdformat 3 +h s=6 - Format partition mounted as disk 3 with a stagger factor of 6.

fdformat 3 +h s=6 f=1 l=152 - Format first 153 tracks of hard disk 3 with a stagger factor of 6. Skip track 0.

Description:

FDFORMAT allows diskettes to be *physically* formatted using the floppy disk controller hardware. New diskettes must be formatted before they can be read or written to by the disk hardware. In addition, a newly formatted diskette will typically have to be initialized using the DINIT command before it can be used by the QNX file system, unless DCOPY is being used to make copies of already initialized diskettes.

FDFORMAT will format the diskette according to how the drive is currently MOUNTed. You can override the defaults by specifying parameters to the command. Floppy disk drives have three physical parameters of interest. The number of sides (single or double), the number of sectors (8, 9, 10 or 15) and the number of tracks (40 or 80).

The **t=***tracks* option allows you to format a diskette with 40 tracks or 80 tracks. Note that 80 track floppy drives are mechanically different than 40 track drives. You can read a floppy diskette formatted with 40 tracks on an 80 track drive by double stepping, but you can NOT format or read an 80 track diskette on a 40 track drive.

The **n=***sectors/track* option allows you to format a diskette to hold more or less information than the default size. For example a 360K floppy diskette (40 tracks, 2 heads, 9 sectors/track) can be formatted to hold 320K or 400K by specifying **n=8** or **n=10** sectors/track. This option is independent of whether the drive is single sided or has 40 or 80 tracks. You should use high quality floppy disks if you use 10 sectors/track. A value of **n=15** may only be used on special high performance drives which may contain 1.2 Meg (2 heads * 80 tracks * 15 sectors/track).

The following table may clarify things.

		Sectors/Track			
Sides	**tracks**	**8**	**9**	**10**	**15**
1	**40**	**160K**	**180K**	**200K**	**+300K**
2	**40**	**320K**	***360K**	**400K**	**+600K**
1	**80**	**320K**	**360K**	**400K**	**+600K**
2	**80**	**640K**	**720K**	**800K**	**+1200K**

*** - Default if no parameters are specified**
\+ - Requires high performance AT disk drive and special high capacity floppy diskettes.

This formatting information is encoded on the diskette and is queried each time a file is opened. This allows QNX to read differently formatted diskettes in a single drive without having to remount the disk. This encoded information overrides those that are specified by the MOUNT command. Diskettes formatted with VERY old versions of FDFORMAT (QNX 1.1), or perhaps other operating systems will not contain this encoded information. In this case the defaults specified by the MOUNT command will be used. You may use the **+patch** option to place the mount information on a diskette which for one reason or another does not contain it. The disk will NOT be re-formatted.

The **-stagger** option disables the staggering of sectors formatted on the diskette. By staggering sectors by 3, a program will have two sector times (about 12 msec) to process one sector of data and still be able to read the next sequential sector on the same revolution of the disk. Although this slows down program load time from the disk, it can significantly increase program throughput!

FDFORMAT may be used to format a floppy disk on another node. The disk may be indicated explicitly via a [*node*] prefix or implicitly by specifying a *remdisk* which you have mounted.

Hard disks may be formatted if the **+hard** option is specified and the hard disk driver supports formatting.

See Also:

DCOPY
DINIT
MOUNT

FDISK

FDISK - Create QNX disk partition

Syntax:

fdisk *drive*

Options:

drive - Disk drive to partition.

Examples:

fdisk 3

Description:

FDISK allows you to partition a hard disk. The partition information matches that used by DOS. It is kept on the first physical block on the disk.

IMPORTANT: FDISK must only be used when the file system is idle. No other users or programs can have open files when this utility is used.

To create a QNX partition for the first time you must first mount a hard disk driver. For example:

mount disk 3 /drivers/disk.xt

This is explained in a technical note in your QNX manual. You do not need to specify an offset or size. You should now execute the FDISK command from the floppy.

fdisk 3

and partition your disk. QNX does **NOT** automatically mount any partition. Once you have created the partition, you should remount the disk using this partition. For example:

mount disk 3 /drivers/disk.xt pa=qnx

Specifying the **pa=** option will cause the MOUNT command to read the partition block, locate the first QNX partition, then mount the drive with the correct offset and size. You will probably wish to place this mount command in your sys.init so that it is executed each time you boot. If you are running the QNX-DOS file system task, you may also wish to mount a DOS partition as another drive. For example:

mount disk 4 d=3 pa=dos

Note that you should NOT remount the driver. The **d=** option indicates that the existing driver which has already be mounted for disk 3 should be used.

It is important to realize that the FDISK command only displays and updates the partition information on the disk. It does **NOT** directly affect your access to the drive. You must still issue the MOUNT command separately.

FDISK is a full-screen, interactive program which is fairly self explanatory. When fdisk is invoked, it will display a screen similar to that shown below:

```
Ignore Next Prev Change Delete Mount Boot Unboot Save Quit

        OS          start       End           Number          Boot
      name type   Cylinder   Cylinder   Cylinders   Blocks

--> 1.  dos (  1)       0        139        140       19039
    2.  qnx (  7)     140        279        140       19040    *
    3.  qny (  8)     280        419        140       19040
    4.  qnz (  9)     420        639        220       29920


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

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


First cylinder is 0    Last cylinder is 639

Disk is 44,564,480 bytes    H=8  T=640  N=17
```

The Possible commands are displayed along the top line. They can be selected by typing the first letter of the command, or by moving the cursor to the appropriate command (with the arrow keys), and typing Enter.

The commands are described below:

BOOT
Make the selected partition the boot partition.

CHANGE
Change the selected partition. You must input an OS type, a start cylinder and an end cylinder.

DELETE
Delete the selected partition.

IGNORE
This menu item does nothing. It is a safeguard against accidental carriage returns which select the current menu item.

MOUNT
Display the parameters which you would give to the MOUNT utility for this partition.

NEXT
Select the next partition. The down arrow key may also be used.

PREV
Select the previous partition. The up arrow key may also be used.

QUIT
Leave the FDISK command. Remember to Save first, if you have changed anything.

SAVE
Save any changes made to the partition back to the disk. Unless this command is executed, your changes will be ignored. This allows you to quit without saving if you mess things up.

UNBOOT
Clear the boot indicator from the selected partition.

NOTE: If you wish your disk to contain both QNX and DOS partitions, be sure to create the DOS partition FIRST (using DOS)

See Also:

MOUNT

FILES - List Files

Syntax:

files [*directory*] [*options*]*

Options:

+busy	- Display only files which are busy.
+directory	- Display directories instead of files.
+current	- Only display files at the current level.
+modified	- Display files which have been modified since the last backup.
-sort	- Don't sort filenames.
-used	- List files which have been FRELed or ZAPped.
-verbose	- List only the name of each file.
+verbose	- List detailed information about each file.
+totals_only	- Only print totals of blocks and files.
t=hh:mm:ss	- Only print files which have a date later than this time on the given date.
d=dd-mon-yy	- Only print files which have a date later than this date.
+Before	- Applies to the **t=** and **d=** options.
a=[& | ^]attr_list	- List files whose attributes correspond to *attr_list* where *attr_list* is one of **maewrc**.
p=*pattern*	- Only display files whose name matches this pattern.

Examples:

files	- List files at current directory.
files /cmds -s	- Unsorted list of files in /cmds.
files +v	- List ALL information on files at current directory.
files 3:/ a=&rw	- List all files which have the READ and WRITE attributes set.
files 3:/ a=^w	- List all files which are not WRITEable
files +d -v >index	- List directories, don't sort, then place output in the file index.
files 3:/ p=*.c +v	- List files on drive 3 which end in **.c**.
files 3:/ d=10-06-90	- List files with dates later than 10-June 1990.

FILES

Description:

The FILES command allows the user to list the files at or below any point in the file structure. If no directory is specified, then the current directory is listed. The FILES command will walk the entire tree structure recursively, displaying all files at the specified (or current) directory followed by all files at lower levels. The **+c** option prevents the recursive walk of the file "tree", causing only those files at the current level to be displayed.

Normally, file names will be sorted before being displayed. This feature can be turned off by specifying the **-s** option, in which case file names and directories will be displayed in the order they are found. When sorting, files at the current directory will be displayed first, followed by any files at lower levels in the directory.

Directories are just special files in the QNX file system. The Files command will normally not print directories. If the user wishes to examine the directories which exist at this and lower levels, he can use the **+d** option. The **-s** and **+c** options apply to directories as well as to files.

If desired, only files whose filename matches a *pattern* may be displayed. The pattern is specified using the **p=** option. A pattern can be any valid filename, and may include the following *wildcards*:

*	matches any run of characters
?	matches any one character
[ ]	will match any of the characters enclosed in the square brackets

An example pattern which will match any file which ends in ".c" or ".h" is "p=*.[ch]".

The +v option can be used to display the information about each file which is kept in the directory. One line of information will be displayed for each file, with the filename displayed last. The following information will be presented:

Blk - Number of disk blocks in the file (512 bytes per block)
X - Number of Extents in the file (indicates degree of disk fragmentation). A contiguous file will consist of one extent.
Loc - Starting location of the file (hex block number)
Grp - Group number of file owner. (1 ... 255)
Mem - Member number of file owner. (1 ... 255)
Attr - Attributes of the file. These are the access methods permitted for the owner of the file (or the super-user and privileged commands)
Perm - Permissions. These are the access methods available to users other than the owner of the file. G-perms apply to other members of the same group. O-perms apply to to all other users.

Date - Date that the file was created or last changed.
Time - Time that the file was created or last changed.
Name - File name (relative to the specified directory). As deeper levels of the file structure are displayed, the complete hierarchical pathname is displayed.

Attributes and Permissions for FILES may be:

r **READ** permission. The file can be read.
w **WRITE** permission. This file can be written to. User programs can therefore change the file contents and even delete the file (since null files are removed)
a **APPEND** permission. The file can be written to, but only new information can be added to the end of the file. The file cannot be deleted.
e **EXECUTE** permission. This file can be executed. If the file contains executable code, it can be executed directly. If instead it contains text, then the file will be treated as a command file, and the text lines will be given to the shell for interpretation.
m **MODIFY** permission. This allows the permissions and attributes of the file to be changed (using CHATTR)

Attributes and Permissions for DIRECTORIES may be:

r **READ** permission. The directory can be read (listed).
c **CREATE** permission. New files may be created under this directory.
b **BLOCK** access. Prevent any files or directories below this directory from being accessed.
m **MODIFY** permission. This allows the permissions and attributes of the directory to be changed (using CHATTR).
d **DIRECTORY**. The file is a directory.

If a file is BUSY, it will be flagged in the attribute column with a 'B' character.

The FILES command is a very useful tool for creating command files. A typical use of this function could be in the generation of a command file which is to delete all of the files at and below the current directory level. The user could start by typing:

files -v >command

which would create the file "command" containing a list of all the files at and below the current directory level. He could then edit this file and substitute the beginning of every line with "frel ", and save the file away:

led command "*s/^/frel /" w q

He could then execute the command file by typing:

sh command

This example might be better accomplished using either the wild card syntax of the shell or the WS command.

frel *
or
ws "frel @" l=12

A better use for such a file is on the creation of an index file for commands like LIST and LINK which support the **x=***index* option. These commands expect to be provided with an index file of the form created by the FILES command.

See Also:

CHATTR
DIR
LED
LS

FOPEN

FOPEN - Display open files

Syntax:

fopen [*node*] [+userid]

Options:

node - The node whose files you wish to query (default is your node).
+userid - Display a userid instead of the program which has a file open.

Return Value:

The return status is the number of files open
or -1 on a critical error.

Description:

FOPEN will print out the names of any tasks which have a file open. It not only indicates the program which opened the file, but also the open mode and the current location within the file. The location can be used to judge a programs progress in processing a file by invoking FOPEN several times. The location is shown as two numbers as follows:

current block / last block

This command can be used in a shell script to check if any files are open before shutting the system down.

```
fopen
if ne #? 0000 type "You have #? files open. Do not shut down the system."
```

FOPEN only lists the files open on one node at a time. It defaults to your current node. You can specify a node using the *node* option.

```
fopen 4
```

See Also:

DOPEN

FREL - Release File

Syntax:

frel *file* [*file*]*

Examples:

frel junk
frel stuff oldjunk test.o
frel [2]1:/config/sys.init

Description:

FREL will remove a file from the disk. The space used by that file is reclaimed for use by other files, and the directory entry is removed. One or more files can be specified on the command line.

If the file exists at the root of a disk, or if the same directory exists on multiple drives, then the filename should be preceded by the correct drive prefix to remove ambiguity. For example, use:

frel 3:/test

to remove the file "test" from the top level of the disk in drive 3.

See Also:

DREL
RM
RMDIR

FSTAT

FSTAT - Display file status

Syntax:

fstat *file** [+xtnts]

Options:

file	- A QNX filename.
+xtnts	- Display the disk extents which are allocated to this file.

Description:

In its simplest form, FSTAT can be used to display the **date** of one or more QNX files.

fstat /user/luc/test.c
fstat *.c

If the **+x** option is used, then FSTAT will also give a complete list of all the disk blocks which are allocated to this file. This option is a useful debugging tool.

See Also:

DIR
FILES
LS

GREP - Search a file for a pattern

Description:

Please refer to the documentation on the LOCATE command.

See Also:

LOCATE

KBD - Redefine keyboard layout

Syntax:

kbd *kbd_type* [**f**=*user_file*]
kbd list [**f**=*user_file*]

Options:

kbd_type - Keyboard type to use
list - List all keyboard types
f=*user_file* - alternate file to use (default: ***/config/kbd.dbase***)

Examples:

kbd 102.GE
kbd 84.FR f=/config/kbd.db.ours
kbd list
kbd list f=/config/kbd.db.ours

Description:

The KBD command remaps the character codes that are produced by your keyboard, using a binary data file (***/config/kbd.dbase***). You will only need to run this command if you are using a non-US keyboard. Users who require this command will likely execute it in their *sys.init* files.

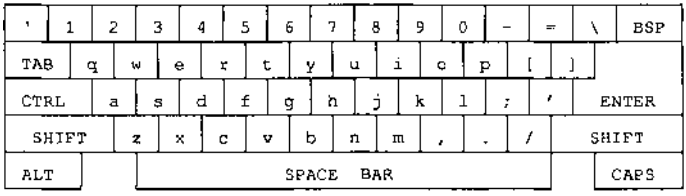
To find out what keyboard remappings are available, use the **list** option. The naming convention for a keyboard remapping is *number_keys.country_code* where *number_keys* is the number of physical keys on your keyboard, and *country_code* is typically a two letter abbreviation for a particular country.

First, you need to figure out which keyboard you have. Specifically, you need to determine *number_keys*. The simplest method is to count the number of key caps on your keyboard. Following is a brief description of each keyboard, and a picture of the default character mapping for it.

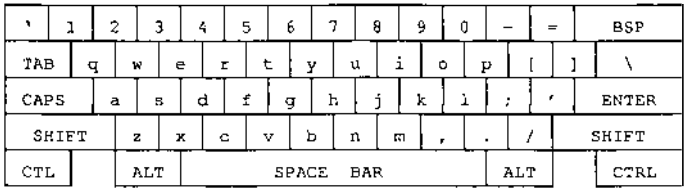
KBD

Below is the IBM PC/XT 83 key keyboard. It has ten function keys along the left side, and an integrated numeric/arrow keypad on the right side.

Below is the original IBM PC AT 84 key keyboard. It has ten function keys along the left side, and an integrated numeric/arrow keypad on the right side.



Below is the "enhanced" 101 key keyboard. It has twelve function keys along the top, and separate numeric and arrow keypads on the right side. It was first available on the IBM PC AT, and later used on the IBM PS/2. This keyboard is only available with the US keyboard layout.



Below is the "enhanced" 102 key keyboard. It has twelve function keys along the top, and separate numeric and arrow keypads on the right side. It was first available on the IBM PC AT, and later used on the IBM PS/2. This keyboard is only available with non-US keyboard layouts.

Once you have determined which physical keyboard you have, you need to determine which character remapping to use with it. Following is a current alphabetical list of the abbreviations for *country_code*, and their meanings.

BE Belgian
CF Canadian French
DA Danish
DU Dutch
FR French
GE German
IT Italian
LA Latin American
NO Norwegian
PO Portugese
SE Swedish
SI Swiss
SP Spanish
UK United Kingdom
US United States

If you do not specify an alternate file using the **f=** option, the file ***/config/kbd.dbase*** will be used. You may create an alternate file by copying the original ***/config/kbd.dbase*** file, and possibly modifying the copy using the KBD_EDIT command, which is available from the update service. The **f=** option allows a user to avoid having their modified keyboard layouts overwritten by an updated ***/config/kbd.dbase*** file from QNX Software Systems Ltd. The keyboard remappings in ***/config/kbd.dbase*** should be accurate, and thus this option should rarely be used.

KILL

KILL - Kill a task

Syntax:

kill *task-id*

• *This is a local shell command* •

Description:

The indicated tasks will be killed. You may obtain the tasks identity with the TSK command. KILL is supported by the shell to allow you to kill a task even when there are no free task descriptors in the system to create a new task. The escape sequence #& refers to the last background task you created.

kill #&

If the first argument starts with a plus sign (+) it is taken as a system exception to set on the remaining taskid's specified. For example

kill +0001 050c -set exception hangup on task 050c

Refer the to chapter on MULTI-TASKING in the QNX manual for more information on exceptions.

You may wish to refer to the SLAY command to remove a task by it's name rather than it's task id.

See Also:

BREAK
SLAY

KILL_VCS - Kill all virtual circuits to a node

Syntax:

kill_vcs *node_id*

Options:

node_id - Node number of crashed node.

Examples:

kill_vcs 3 - Kill virtual circuits to node 3

Description:

The KILL_VCS command will kill all virtual circuits between your node and a node which has crashed. This allows *manual* recovery without using the poller. Please refer to the QNX manual on setting up your network.

If a poller is running somewhere on the network, then this command should not be needed.

See Also:

ALIVE
POLL
TSK

LED

LED - Line Editor

Syntax:

led [*file* [*commands*]]

Examples:

led
led junk
led test.c *s/bill/joe/ w q

Description:

The editor is a line oriented text editor which supports a number of powerful pattern matching facilities. Only text files can be edited, which means any files containing regular ASCII characters on lines which are terminated with a newline character (hex 1E). Null characters (hex 00) are not allowed within the file.

Lines are numbered sequentially starting from 1. Line numbers change dynamically as new lines are created and lines which are not needed are deleted. Lines can be referred to by specifying the line number explicitly, or relative to the current line (which is referred to by the character "."). A line can also be referred to by specifying a pattern of characters which is found on that line. Some commands support ranges of lines, which may begin and end with line numbers, relative line numbers, or lines containing patterns.

Files are edited by first reading the entire file into a memory buffer. This provides for a very fast editor, and minimizes the dangers of having files open while editing. Unfortunately, very large files cannot be handled by this editor. It is recommended that large files be divided into several smaller files before editing.

Files can be inserted anywhere into the editing buffer with the "r" command, and portions of the buffer can be written to any file (default being the current working file).

Features supported by the editor are:
- insertion before a line
- append after a line
- delete line(s)
- write a group of lines to a file
- append a file after the current line
- character substitution on a range of lines
- global command execution on a group of lines
- join two lines
- print a group of lines
- change current line
- location of lines containing patterns
- moving a range of lines to another position
- copying a range of lines to another position

Editor Command Syntax:

The format of an editor command is typically of the form:

linerange ***command*** *parameters*

where:

command	is the command character (described below)
parameters	are arguments to the command
linerange	is the range of lines which are to have the command applied to them.

linerange can be:

line
or *line,line*
or *

line can be:

lineno
.
$
/pattern/
line+nn
line-nn

The character "." represents the current line. The character "$" represents the last line. The character "*" represents ALL lines which is the same as "1,$". The character "&" represents the lines from the current line to the current line plus page size (ie. one screen).

Patterns represent a line which is found to contain that pattern. Patterns enclosed in slashes (/) mean the next line AFTER the current line which contains the pattern. Patterns enclosed in question marks mean the closest line BEFORE the current line which contains the pattern, scanning backwards. In both cases, scanning wraps around the beginning and end of the file and will terminate when the current line is again reached.

Example *line* numbers and *lineranges* are:

3	- line 3
1,10	- lines 1 to 10
.	- the current line
.+25	- 25 lines past the current line
.-10,.+5	- all lines from 10 lines before the current line to 5 lines past the current line
/joe/	- the next line which contains the string "joe"
1,/loop:/+4	- all lines from line 1 to 4 lines past the next line containing the string "loop:"
1,$	- all lines
*	- all lines

Editor Patterns:

The text editor allows pattern arguments to be used when locating line numbers, and when substituting strings.

Several metacharacters are used when specifying a pattern. these are:

^	- represents the null character at the beginning of a line
$	- represents the null character at the end of a line
.	- matches any character
[*ccc*]	- where "*c*" can be any characters, will match any one of the enclosed characters. (eg. [abc] will match any of "a", "b", or "c"
[*c*-*c*]	- where "*c*" is any character, will match any of the characters in the range of ASCII characters. (eg. [a-z] will match any letter)
[^*ccc*]	- where "*c*" is any character, will match all characters EXCEPT those in the square brackets
@(*nn*)	- which represents the null character before character position *nn*.

Any of the above pattern characters may be followed by a star (*), which means zero or more occurrences of that pattern character. (eg. "a*" will match a null character, a, aa, aaa, etc. ... ".*" is useful in matching a run of any characters)

Preceding any character (including the metacharacters) with a backslash (\) will indicate that that character is to be taken literally. (eg. '*" will match the character "*", and "\\" will match the character "\").

All other characters in a pattern are treated literally (ie. they will match only that character.

Examples:

/abc/	- will match "abc" anywhere in a line
/^abc/	- will match "abc" at the beginning of a line
/abc$/	- will match "abc" at the end of a line
/.abc/	- will match any character followed by "abc"
/.*abc/	- will match any run of characters followed by "abc"
/^.*$/	- will match everything on a line
/[0-9]*/	- will match the largest run of characters consisting only of numbers (which may be the null string)
/[0-9][0-9]*/	- will match a number
/[^123]/	- will match the largest run of characters which does not contain the digits 1, 2 or 3

In string substitutions, two patterns are specified separated by delimeters. (eg. s/fred/bill/ or s,ed/,edit/,). The special metacharacter ampersand (&) is used to represent the portion of the line which was matched by the first pattern. This allows some very powerful editing operations.

For example:

s/[0-9][0-9]*/<&>/ - will turn the line "123 men on 3 horses" into the line "<123> men on <3> horses"

s,^.*$,copy & dir/&, - will turn the line "junk.c" into the line "copy junk.c dir/junk.c"

When these pattern substitutions are combined with the global command (g), some very powerful editing operations can be performed.

Editor Command Summary:

LED

[*line*]	**a**	- Append new input after the line. End with a line containing a single "."
[*range*]	**c**	- Change these lines to new input. End with a line containing a single "."
[*range*]	**d**	- Delete these lines
	e *file*	- Edit new file...delete all lines, read in new file, and change current file name.
	f [*file*]	- Change current file. If no file, then display current file name.
[*range*]	**g**/*pattern*/*commands*	- Globally execute these ed commands on this group of lines.
[*line*]	**i**	- Insert new input before the line End with a line containing a single "."
[*line*]	**j**	- Join this line with the next line.
[*range*]	**k***line*	- Make a copy of these lines behind the indicated line.
[*range*]	**m***line*	- Move these lines behind the indicated *line.*
	oc1*char*	- Defines the command-line prompt character to be this *char.*
	oc2*char*	- Defines the input prompt to be this *char.*
	oc3*char*	- Defines the character to print when a null-line is entered.
	od	- Dual-case is recognized in patterns.
	odr	- Dual-case not recognized. Upper and lower case are the same.
	of	- Display special symbols in expanded form \hh where "hh" is the hex value of that character.
	ofr	- Turn off expansion of special symbols.
	op*number*	- Define page size (for "&").
	os*number*	- Auto-save into file "autosave" after this many lines has been entered in input mode.
	ov	- Display error messages upon error.
	ovr	- Turn off error messages. Only display error number.
[*range*]	**p**	- Print these lines.
	q	- Quit if file not altered.

	qq	- Quit anyway.
[*line*]	**r** *file*	- Read the file and put in buffer AFTER this line.
[*range*]	**s**/*pattern*/*new*/	- Substitute all occurrences of this pattern for the new *pattern* within this line range.
[*range*]	**s***number*/*pattern*/*new*/	- Substitute the Nth occurrence of the *pattern.*
[*range*]	**v**/*pattern*/*commands*	- Execute the ed *commands* on all lines EXCEPT those with this *pattern.*
[*range*]	**w** [*file*]	- Write these lines to the file
	&	- Print a page (size defined by "op" command).
[*line*]	=	- Display this line number (eg. ".=").
	CR	- Display next line.

LINK

LINK - Link object files

Syntax:

link [**o**=*new_object_file*] *objfile** [*options*]*

Options:

+i	- Intel addressing mode (default).
+m	- Motorola addressing mode.
+v	- Verbose.
m=*mapfile*	- Create a map and put in *mapfile*.
o=*obj_file*	- Create a new object file.
u=*sym*	- Use the symbols found in *sym* but do not include the code or data.
s1=*hhhh*	- Segment 1 starts at *hhhh* (hex).
s2=*hhhh*	- Segment 2 starts at *hhhh* (hex).
s3=*hhhh*	- Segment 3 starts at *hhhh* (hex).
s4=*hhhh*	- Segment 4 starts at *hhhh* (hex).
s1=**$***n*	- Segment 1 starts AFTER segment *n*.
s2=**$***n*	- Segment 2 starts AFTER segment *n*.
s3=**$***n*	- Segment 3 starts AFTER segment *n*.
s4=**$***n*	- Segment 4 starts AFTER segment *n*.
s=*stack*	- Increment to minimum stack size.
c=*loadfile*	- Put output file into *loadfile* (default is "core").
x=*index*	- Link the object files in *index*.
l=*library*	- Search directory *library* for any unresolved functions and variables. A file named "*library*/directory" must be found which lists symbols defined in each object file.
n=*namelen*	- Maximum length of a name (default: 66).
b=*bufsize*	- Size of *symbol* buffer (default: 32000).

Examples:

```
link /lib/entry.o test.o
link x=edfiles m=.map. l=/lib s=4000
link o=ed.o x=ofiles +v
link u=rom.map rom.o m=map s1=ffc0 s2=1000 s3=$2
link o=$null test.o +v
```

Description:

The routine CC should be used instead of LINK for linking programs to run on QNX. CC will invoke LINK with the proper arguments to create a load file for the QNX environment. Any arguments to CC are passed on to LINK. The same is true for any extra arguments passed to the CC command. LINK may be called directly to link programs for environments other than QNX.

The Linker will link together object files which are created by one of the compilers or the assembler. A load file is produced which is in a form to be executed directly (by typing its name on the command line). All external references are resolved, and warning messages are printed if any object module references a symbol which is not defined in any other module.

The default values for the 4 segments which are allowed by the assembler are such that if no overriding values are specified, the command will execute properly. Overriding segment starts and offsets are typically only used when linking together object files which are not to be executed as a command. For example, a program which is to be placed in ROM would need to be linked outside of the default memory segments.

The **x=** option accepts a file list in the form which would be created by the FILES command, allowing convenient linking of commands which consist of a large number of object files. Therefore the format of an index file is with one object filename per line.

The **l=** option invokes a search for unresolved symbols under the specified directory. A file named "directory" is used to define the contents of this library of object files. The format of this file is one object file name on a line by itself, followed by any number of lines which begin with a TAB character followed by one symbol which is defined in that object module (see the file "/lib/directory" as an example). Any number of object files may be described in the library file. Multiple **l=** options are allowed so that many libraries can be searched. All library references must be forward. Refer to the documentation on QNX compilers in your C binder for more information.

Program execution begins with the first address in segment 1 which is typically the code segment.

By default, all commands are given a 1K stack by the operating system. The **s=** option in the linker allows additional stack space to be allocated.

The linker will create a map file if the **m=** option is used. This map is useful when debugging programs. The address (hex) of all functions and variables is included in the map. Typically the map should be sorted with the SORT command if it is to be

used in debugging a program.

```
sort map_file s=1,3 +r        (sort by name)
  or
sort map_file s=1,2 +r        (sort by address)
```

The linker can take a collection of object files and combine them into one large object file using the **o=** option. It must immediately follow the LINK command. The **s1=**, **s2=**, **s3=** and **c=** options are not used.

```
link o=ed.o main.o command.o output.o input.o +v
```

Note that this option may be used to determine the size of an object module. In this case the output is thrown away.

```
link o=$null obj_file.o +verbose
```

There is no limit to the number of object files which can be linked together provided that the resulting command will fit within the address limitations of the processor (64K code + 64K data). **If the code segment is exceeded, you will need to refer to the the technical notes in the C Compiler binder.**

LINK maintains its *symbols* in an allocated buffer. This buffer defaults to 32,000 bytes, but may be changed with the **b=** option. If LINK complains that it has run out of symbol space (for very LARGE programs), try again with a bigger buffer. If memory is tight on your machine, you may wish to decrease this value. Valid sizes are 2000 to 65,520.

See Also:

ASM	CC
SORT	*Technical Notes (C)*

LIST - List files on the line printer

Syntax:

list [*options*]* [*file*]*

Options:

b=*banner*	- Name of banner on first page.
d=*device*	- Name of $device to use for output (default: taken from prt.init).
i=*file*	- Printer configuration file (default: /cmds/prt.init).
l=*page_length*	- Define page len in inches (default 11).
n=*name*	- Name to print in header (default: filename).
o=*offset*	- Offset from left hand margin.
p=*port*	- Define semaphore port (default 16).
s=*vert_spacing*	- Vertical spacing (lines per inch).
w=*line_width*	- Characters per line.
t=*tab*[,*tab*]*	- Define tab stops.
x=*index_file*	- File containing list of files to print.
+**n**umberlines	- Print line numbers with listing.
+**r**eset	- Reset page number to 1 for new files.
+**g**ap	- Feed a gap for a new banner.
+**a**ssembly	- Set tabs for assembly source program.
+listing(assembly)	- Set tabs for assembly listing.
+**e**mphasized	- Turn on emphasized printing.
-**e**mphasized	- Turn off emphasized printing.
+**d**ouble_print	- Turn on double printing.
-**d**ouble_print	- Turn off double printing.
-**p**agination	- Turn off pagination (and headers).
-**h**eader	- Turn off header (date and page no.).
+**b**anner	- Print a banner page.
+**u**ser	- Print user numbers on each page.

Examples:

list test.c
list s=8 w=80 o=10 report

```
list +a "n=ASM Files" x=assembly_index
list +g
list +b -p -h w=80 s=6 +e good_document
files +v | list &
```

Description:

LIST provides a means of printing paginated output on a line printer. The date, time, filename, and page number are printed at the top of every page. More than one file can be specified on the command line. Index files may also be used which contain a list of files to print with the current option settings. More than one index file can be specified. The format of an index file is that of the output of the FILES command (with the **-v** option).

Tabs are expanded into spaces by the LIST program. The default tab stops are at every 4th character position. This can be changed with the **t=** option. If more than one tab stop is given, then LIST will assume that tabs past the last specified tabstop will have the same spacing as the gap between the last two given tabstops. For example "t=4,7" will put tab stops at position 4, 7 and then every 3 positions afterwards since the difference between the last tabstop (7) and the previous one (4) is 3 spaces.

For printers which support different line spacings, the **s=** option allows the user to select the line spacing (eg "s=6" for 6 lines per inch, "s=8" for 8 lines per inch). Some printers also support different character widths. Character widths can be changed using the **w=** option. Examples are "w=80" for 80 characters per line and "w=132" for 132 characters per line (on an 8 1/2 inch page).

Some printers (such as the MX-80) allow special print modes such as emphasized printing (use **+e**) or double printing (use **+d**).

If listings are to be placed in binders, a margin is often desired on the left side of the page. The **o=** option will offset all output the specified number of spaces to the right to leave such a margin.

When used with the banner option (**+banner**) the first page of a listing contains a "banner" which gives the current date and time as well as the name of the first file being listed. A different banner can be specified with the **b=** option. In this case the LIST program will always feed enough spaces after the last file has been printed to allow the user to remove his listing without needing to feed the paper himself. The resulting gap is used by the next LIST command for its banner. This mechanism works well provided that the user only uses the LIST program to generate output on his line printer. To initially position the paper at the correct position for printing banners, the user should position the paper to the top of a new page, then issue a "list +g" command which will feed the proper number of spaces. Subsequent LIST

commands will then be positioned properly. This procedure is especially useful for printers which don't support formfeeds.

The LIST program also provides a means of turning off the printing of the top-of-page header (**-h**), and/or the automatic spacing between pages (**-p**). This allows the user to print documents which don't require these features, but still be able to specify line spacings, print modes, etc.

The LIST command will automatically wait for other LISTs to finish before printing. This automatic spooling is especially useful in multi-user installations where many users wish to print files at the same time. So long as everyone uses LIST to print their files, the listings will not be mixed together.

The characteristics of the line printer are found in the file "/cmds/prt.init". This file name may be over-ridden using the **i=** option. If no configuration file exists, then the LIST program assumes that it is talking to a printer which prints 80 characters per line at 6 lines per inch. An example format of the file "/cmds/prt.init" is shown below for the EPSON MX-80 printer. All numbers are decimal unless followed by the character 'h' or 'H' in which case they are hexadecimal. These numbers often represent ASCII characters. Note that the comments in the following example should not be included in the file.

$lpt	name of printer (may be overridden using **d=**)
12	form-feed character (0 if not supported)
132	default width for listing (characters per line)
8	default spacing for listing (lines/inch)
80	default width for headers
80	default width for banners
6	default spacing for banner page
80	WIDTH #1 (characters per line, 8 1/2" page)
20 18	string to print to cause this line width
132	WIDTH #2
20 15	string for width #2
0	END of widths
6	SPACING #1 (lines per inch)
27 50	string to print for spacing #1
8	SPACING #2
27 48	string for spacing #2
0	END of spacings
27 69	string to print for +e
27 70	string to print for -e
27 71	string to print for +d
27 72	string to print for -d

LIST

See Also:

P
SPOOL

LOCATE

LOCATE - Locate patterns of characters in a file

Syntax:

locate *"pattern"* [*file* | **x=***file*]* [*options*]*

Options:

c=*context*	- Number of lines to print for each match.
b=*bytes_to_skip*	- Number of bytes to skip before starting the locate.
w=*pattern*	- After a match continue output while following lines match *pattern.*
x=*file*	- Index file containing list of files to search.
+dualcase	- Distinguish between upper and lower case.
+count	- Exit with number of *finds.*
-verbose	- Don't display filename and line number.

Examples:

```
locate array test.c
locate INV main.c support.c x=index junk.c +d
locate "([a-z][a-z]*)" *.c -v
```

Description:

LOCATE will find patterns of characters in a given file (or files). The name of the file, and line number will be displayed along with the entire line, for each line in the file which contains the pattern. Patterns are the same as those used in the LED command.

Index files may be used to avoid typing long strings of file names on the command line. The format of these index files is the same as the output format of the "files -v" command.

Consider a directory called /application which contains several C source programs, and perhaps several object files as well. To find all the C source files which contain a call to the function "sort", the user could use the following:

```
files /application -v p=*.c >index
locate sort( x=index
       OR
locate sort( *.c
```

This will produce one line of output for every occurrence of the pattern "sort(" in all these files. The LOCATE command will also supply the filename and line numbers where these patterns occur. The WS command is often used with locate for the same purpose.

ws "locate sort(@" p=*.c

Normally, upper and lower case letters are treated identically. The user can specify **+d** which will then distinguish between upper and lower case letters.

The **-v** option turns off the display of filename and line number at the beginning of each output line.

The **w=** option permits simple lookup databases to be implemented with simple text files created by the editor. For example, a file could be created with keywords followed by lines of descriptive text which are indented by one or more spaces. To print the text associated with a keyword "QNX", one could issue the following command:

locate ^QNX "w=^ " -v

The LOCATE COMMAND can be used in SHELL scripts to count the number of occurrences of a string in a file. The **+**count option is useful for this.

The **c=** option allows some *context* to be printed around each matching line. For example, specifying **c=5** would print each line containing the pattern, along with the next 4 lines (5 lines of *context*). The **c=** and **w=** options cannot be used at the same time.

See Also:

FILES
LED

LOCKER - Implement record locking in QNX

Syntax:

locker [options]* &

Options:

s=*cache_size*	- Number of 1K blocks to allocate for the cache. Default is 64, minimum is 32, maximum is 1000.
p=*port*	- Port to attach to. Used for timing purposes to age the cache.
b=*buffers*	- Maximum data segment buffers to be used.
f=*files*	- Max number of simultaneous open files.
+*debug*	- Internal use only.
+*sync*	- Force O_SYNC option on all OPEN()'s. (See the C compiler manual for details)

Description:

LOCKER implements a record locking facility in QNX. To make use of it, applications must have been written specifically to use LOCKER. LOCKER is usually started from the sys.init at boot time, and left running always. The *b=buffers* option us used to set the number of data segment buffers to be used by LOCKER (max approx 30). The larger this value the less room there is for locks, task entries etc.

If you need to terminate LOCKER, use the SLAY command:

slay locker

Although no QNX Software Systems applications are using LOCKER (as of January, 1988), this may not always be true. However, a number of 3rd party applications (notably database products) are using locker. If you have any doubt as to whether or not you should run locker, check the documentation for the applications you wish to run, or contact the vendor or QNX Software Systems' technical support.

LOCKER requires the presence of TIMER. Before starting LOCKER, you **must** invoke TIMER (it is used to age the cache). For example:

timer &
locker s=100 &

See Also:

TIMER
Record Locking in QNX in the C manual Technical Notes

LOGIN

LOGIN - Log-in to QNX

Syntax:

login [*userid* [*password*]] [+stack]

Options:

userid	- Name of user: (Max 16 characters).
password	- Unique password for that user. (Max 20 characters).
+stack	- Stack logins, freeze current user.

Examples:

login
login bill secret
login john +s

Description:

LOGIN allows a user to log-in to a QNX system. QNX will invoke LOGIN on the console automatically when the system boots. LOGIN will also be invoked if a user types Ctrl-Z on any idle terminal or console.

If invoked with no arguments (such as with automatic invocation by QNX), LOGIN will prompt the user for his *user-id* and *password*. The user will be greeted with a message of the form:

QNX version x.xx (Release x)
Copyright © QNX Software Systems Ltd. 1982, 1993
Login:

The *user-id* supplied by the new user is checked against all valid *user-ids* found in the file '/config/pass'. If a match is NOT found, then LOGIN will re-issue the message:

Login:

If the *user-id* DOES match one of the entries in the password file, then LOGIN will prompt the user for a *password*:

Password:

This *password* must match the one found in '/config/pass'. If it does not, then LOGIN will re-issue the login message. If the *password* was found to be correct, then LOGIN will create the user. The user's number (group.member) is assigned from the password file, as is his home directory. An initial command is also executed automatically, if one exists in the password file. Finally, LOGIN will create a SHELL on the terminal for the user to use.

Once a user has successfully logged-in, he may use LOGIN to log-in to another user. If **+stack** is not specified, then LOGOFF will be called automatically. If a valid *user-id* and correct *password* are given to LOGIN, then that user will be logged-in without prompting for these values. If both fields aren't supplied (or are incorrect), then LOGIN will issue the login prompt and wait for a valid *user-id*.

See Also:

LOGOFF
SH
QNX Manual

LOGOFF

LOGOFF - Terminate a QNX Session

Syntax:

logoff [-z]

Options:

-z - Don't print message regarding Ctrl-Z.

Examples:

logoff
logoff -z

Description:

LOGOFF is called by a user to terminate his session with QNX. All resources which were used by the user will be reclaimed by the operating system and his accounting entry will be deleted.

If the **-z** option is not specified, a message of the form:

--------- Type a Ctrl z to login ---------

will be displayed to remind the next user of that terminal to type a Ctrl-Z to log in.

See Also:

LOGIN
SH
QNX Manual

LPS - List Postscript Laser Printer Filter

Syntax:

lps [*file*]* [*options*]*
options: **+align +(crt) +duplex +Duplex +header +landscape**
+Landscape +pchars +verbose +quad +Quad +octal +Octal
a=*alignfactor*[440] **A**=*align_char* **c**=*columns*[1] **f**=*font*[4]
i=*ignored_pages* **k**=*copies* **l**=*maxlines* **n**=*pages_to_print*
o=*offset* **O**=*offset* **p**=*cur_psize*[12] **r**=*rightmargin*
s=*extra_spacing* **sx**=*x_scale* **sy**=*y_scale*

Options:

+align - Turn on auto-alignment feature.

+(crt) - For each letter entered, map that letter when enclosed by parentesis into the symbols (• c)->©, (• r)->® and (• t)->™.

+duplex - Print two pages on one physical page in landscape mode. The logical pages are scaled, rotated and translated to fit.

+Duplex - Same as **+d** except the page offset is applied to the logical page printed on the right side of the physical page.

+header - Display the filename as a header on each page printed.

+landscape - Rotate the logical page to print sideways on the physical page. If more than one column is printed the page offset is only applied to the first column.

+Landscape - Same as **+l** except the page offset is applied to each extra column printed when **c**=*columns* is specified.

+pchars - Support the codes for the PC line drawing characters. These characters are defined in the file **/config/pcfont.ps**.

+verbose - Print a comma (,) for each logical page skipped and a period (.) for each logical page printed. Remember that several logical pages may be printed on one physical page.

+quad - Print four pages on one physical page in portraite mode. The logical pages are scaled and translated to fit.

+Quad - Same as **+q** except the page offset is applied to the logical pages printed on the right side of the physical page.

+octal - Print eight pages on one physical page in landscape mode. The logical pages are scaled and translated to fit.

+Octal - Same as **+o** except the page offset is applied to each logical page.

a=*alignfactor* - Set the width of the alignment character in 100 * points. The

default is 440.

A=*align_char* - Two character hex value which represents the alignment char. The default is the backquote (‘), hex 60.

c=*columns* - Select the number of columns to place on a physical page. Each logical page starts a new column.

f=*font* - Select the font to start printing with. It defaults to Times Roman.

i=*numpages* - Ignore (skip) this number of logical pages. Note that several logical pages may print on one physical page (ie: +dual, c=columns ...).

k=*copies* - Number of copies to print for each page. The default is 1.

l=*maxlines* - Force a new logical page after receiving this number of lines. Useful when text is pre-formated without formfeed characters.

n=*numpages* - Only print this number of logical pages. Note that several logical pages may print on one physical page (ie: +dual, c=columns ...).

o=*offset* - Specify a page offset (left margin) in points. Defaults to 36 points (1/2 inch).

O=*offset* - Same as **o**= except the offset is applied to all columns and not just the first column.

p=*points* - Select the pointsize to start printing with. This set the pointsize for a logical (full size page). The **+d**ual and **+q**uad options scale it. It defaults to 12 point.

r=*rightmargin* - Specify a right margin in points. Used for justification only, lines are not broken. Defaults to 468 points (6 1/2 inches) from left page border.

s=*spacing* - Specify extra points (1/72 inch) of space between printed lines.

sx=*scale* - Floating point number to scale charcater width. Do not use with the +d or +q options.

sy=*scale* - Floating point number to scale character heigth. Do not use with the +d or +q options.

Examples:

```
lps text o=72 >$mdm
lps text +d >$mdm
lps text +q >$mdm
lps text +l c=4 f=0 p=8
lps text +l c=4 f=0 p=8 sx=.75
```

Description:

The LPS utility takes a text file and converts it into a set of postscript commands which will display it on a laser printer supporting postscript. There are a number of command line options which provide considerable flexability in the formating of the output. The utilty also recognizes a large set of escape sequences which can be embeded within the text to further enhance your output. For example, there are sequences to changes fonts, pointsize, attributes and underline to mention only a few. These escape sequences allow word processors and the QNX list command which do not support postscript to be used with a postscript laser printer. To find which fonts are supported by LPS type.

lps ?

Note: Your printer may support fewer or more than those fonts listed.

The size tables for all these characters are kept in the file **/config/PSwidths.** Programmers may use these tables for their word processors. It is structured as

```
struct font_entry {
    char font_name[32]
    unsigned font_widths[256]
    } font_file[NUM_FONTS];
```

with each value containing the characters width at pointsize 1000. Keeping the width as scaled integers saves the application from having to use floating point which is much slower. LPS also uses the following two files.

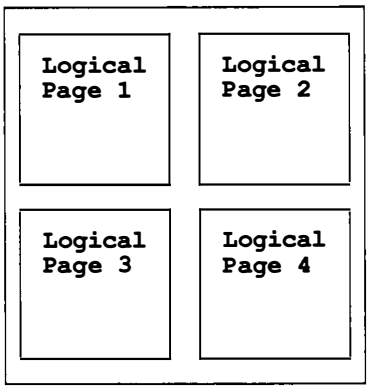
/config/Prolog.ps - **Pre-pended to each job.**
/config/LineChars.ps - **A line drawing character set.**

If you are familure with Postscript, you may edit and modify these files. In particular you may wish to add to the line drawing character set.

LPS takes as input a series of logical pages and from them produces one or more physical pages. A logical page is what you would consider to be a full page of text. The end of a page is usually marked by a formfeed character or the end of file. If your text does not separate it pages with formfeeds but instead expects the printer to start a new page every *nn* (say 66) lines then you can specify a **l=66** parameter to LPS to cause it to paginate based on this number. If your pointsize is too large for a logical page to fit on the physical page LPS will itself break the page when you run off the bottom. This allows you to take a completely unformated piece of text and run it through LPS without any options knowing that it will be paginated properly reguardless of the pointsize used. If you text contains formfeeds and you select too large a pointsize LPS will paginate on page overflow then again when it sees the formfeed character resulting in a second page which contains only a few lines of text. In this case you should reduce your pointsize.

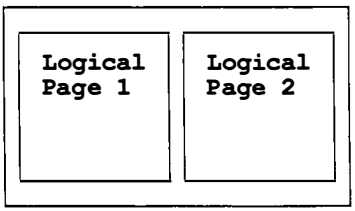
LPS

Unlike a daisy wheel or dot matrix printer which maps one logical page to one physical page, the LPS utility can map several logical pages to one physical page. It does this by scaling the text, and then translating it to different area's of the page instead of starting a new page. For example the +quad option puts 4 logical pages onto 1 physical page by scaling the x and y size by 0.5 and treating the physical page as 4 small pages by translating the output. The mapping looks like this.



Physical Page

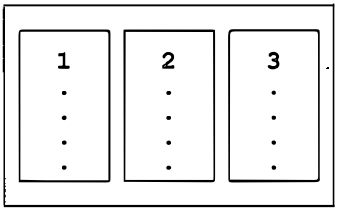
This has two advantages, for drafts it saves paper and on a 7 page/minute laser printer you can often achieve 20 logical pages/minute throughput. Each logical page displayed will also look the same as that produced when you go for a full page master of each logical page. The +dual option rotates the page and placed 2 logical pages on one physical page. Each logical page is the same size as used in our manuals. When producing a master for a printer it is best to produce pages at full size and have the printer reduce them when he makes the plates. In the case of pages of the size you are now reading this reduction has the side effect of increasing you laser printers resolution from 300 dots/inch to over 400 dots/inch.



Physical Page

For those with good eye sight there is also a +octal mode which prints 8 logical pages per physical page. The output is still quite readable (heh, heh) and if your laser printer can keep up, you can print at an effective rate of 8 times your laser printer's page output rate.

You can also place mutiple logical pages on the physical page by using the **c=***columns* option which will divide the page into the indicated number of columns. Each logical page will start at the start of the next column. In this case you have to select a pointsize which fit the text in the columns. Scaling is NOT performed automatically as it is with the dual and qaud options. You can also specify landscape mode to rotate the page. The options of **+l** and **c=3** would produce a physical page of.



The **+dual** and **+quad** options start the logical pages which are printed on the right side of the physical page exactly at in the middle. The page offset is only applied to the left side. If you specify a capital letter (**+D**, **+Q**) then the offset will be applied to the right side as well. In the case of dual printing this provides for a margin if you wish to cut the page in half and place it in a binder.

When producing tables with a variable width font they seldom line up. Each character has a differrent width. You can force alignment by adding a backquote. The **+align** option can be used to automatically add them for you. However, it makes certain assuptions and guesses which may not be what you want. When enabled, for each line LPS reads it will add them after a run of two or more spaces not preceeded by by a ., ? or ! and around dashes (-) and o's (o) which are often

used for points.

aiaiaia - Without alignment char on any of these lines.
axaxaxa - Without alignment char on any of these lines.

aiaiaia - With alignment chars on each side of the dash (-) and the start of text on each these lines.
axaxaxa - With alignment chars on each side of the dash (-) and the start of text on each these lines.

Please note that a space is a very narrow character while a capital A is very large. Alignment is based upon an average which assumes that all characters are about the width of a small a. This will be true on average but fails badly when presented with all capitals. The line.

```
AAAAAAAAAAAA 'point one
```

will actually slide the text "point one" to the left resulting in an overprint as follows:

```
AAAAAAAAAApoint one
```

This can be corrected by simply increasing the number of spaces before the point.

```
AAAAAAAAAAAA          'point one
```

which produces

```
AAAAAAAAAAAA          point one
```

You can use simple shell files to interface utilities like LIST to your laser printer. For example, the following shell file is suitable for producing listing of C programs on your laser printer. To conserve paper it prints with the +dual option and uses the courier font. Note that you will have to create a list initialization file which uses the escape sequences supported by LPS.

```
list #* >/tmp/#n#t.lst i=/config/ps.init
lps /tmp/#n#t.lst f=0 +d >$mdm
```

If this file was called LLIST you can use it in place of LIST.

llist file1.c file2.c ...

Your laser printer is going to be very popular and you will want to share it amoung many users. This is most easily acomplished using the SPOOLER and SPOOL command. If your printer is attached to $mdm you should run the spooler as follows.

spooler "c=lps >$mdm" t=/tmp/ &

You may wish to place this command in your system initialization file. You can now submit files using the SUBMIT command.

spool su file ***any lps options***

In the example of the LLIST shell command above you would replace the second line with LPS to a SPOOL SUBMIT command.

Escape Sequences

In the following tables some of the escape sequence take one or more ascii digits as parameters. They will be represented as italic *d*'s.

d **- A single digit.**
dd **- Up to 2 digits.**
ddd **- Up to 3 digits.**
dddd **- Up to 4 digits.**

If several digits are expected you can supply fewer as long as you terminate them with a non-digit. For example if *dd* was expected then input would be interpreted as

1xyz - Read a 1 followed by "xyz"
12xyz - Read a 12 followed by "xyz"
123xyz - Read a 12 followed by "3xyz"
123xyz - Read a 12 followed by "3xyz"

You may also enter a leading plus or minus sign to indicate a relative rather than an absolute change.

ESC p 12 - Set pointsize to 12 points.
ESC p +3 - Add 3 to current pointsize.

ESC p -3 - Sub 3 from current pointsize.

There are two classes of escape sequences, single character and muti-character. The multi-character sequences always start with an ESC code (hex 1b, dec 27). You will note that LPS supports the escape sequences supported by the system console. A file which highlights, underlines ... text on the console will do the same on the laser printer. Inverse is mapped to italics and blink to larger characters. The console colour escape sequences are ignored.

Note that center, fully justify and right justify are easily remembered by there first letter with a control key. The text following these codes extendes to another code or a code which forces a repositioning of the text (typically a newline).

Single Character Sequences

Name	Ctrl	Hex	Action
TAB	ctrl i	09	Move to next tab stop. Tabs fixed every 4.
RS	ctrl ^	1e	Move to start of next line.
CR	ctrl m	0d	Move to begining of this line.
LF	ctrl j	0a	Move down a line (same column).
ETX	ctrl c	03	Center following text between margins.
ACK	ctrl f	06	Fully justify following text between margins.
DC2	ctrl r	12	Right justify following text between margins.
ETB	ctrl w	17	Force a current position (moveto command) to printer.
EOT	ctrl d	04	Ignored and striped from the text.
FF	ctrl l	0c	End of a logical page (formfeed).

The following will create a line with three parts. One part left justified, one part centered and one right justified. Text in < >'s represent a single character.

Utilities<ETX>© QNX Software Systems<DC2>LPS<RS>

Multi Character Sequences

Sequence	Action
ESC (	Italics on.
ESC)	Italics off.
ESC <	Bold on.
ESC >	Bold off.
ESC [	Underline on.
ESC]	Underline off.
ESC {	Increase pointsize 1/3.
ESC }	Decrease pointsize 1/3.
ESC #	Pattern mask on.
ESC :	Pattern mask off.
ESC d	Subscript (move down 1/4).

ESC f *dd*	**Set font.**
ESC p *ddd*	**Set pointsize in points.**
ESC u	**Superscript (move up 1/4).**
ESC m *ddd*	**Set right margin in points.**
ESC ‘	**Display the quote char.**
ESC g *ddd*	**Set gray level. 0 (black) to 100 (white).**
ESC z *d*	**Add zing to text. 0 to 3 supported.**
ESC z 4 *cmd* .	**Use postscript command to show text.**
ESC o *ddd*	**Set page offset in points.**
ESC a *d*	**Set auto-align on (*d*=1) or off (*d*=0).**
ESC F *d*	**Expand mask (Esc #, :) to right margin if *d*=1.**
ESC M *ddd*	**Manual feed the next *ddd* sheets of paper.**
ESC S	**Save font, pointsize, attributes and zing.**
ESC R	**Restore from last save (ESC S).**
ESC b *dd dd*	**Draw a box. If gray level != 0 then it is filled.**
ESC y *ddd* , *ddd*	**Move by points by x (col) and y (row). This is usually done as a relative move.**

The following sequences impliment the console cursor movement escape sequences. To be useful you should use the courier fixed point point. Proportional fonts may not line up as expected.

Cursor Movement Sequences

Sequence	Action
ESC A	**Cursor up.**
ESC B	**Cursor down.**
ESC C	**Cursor right**
ESC D	**Cursor left.**
ESC H	**Cursor home.**
ESC Y *cc*	**Direct cursor addressing.**

For special situations you may pass postscript commands directly to the laser printer by enclosing them in the follow escape sequence.

SYN SYN *postscript commands* SYN

In conclusion, the following escape sequence can be used to create the shaded blocks with which each of our utilities start with. You will have to supply a 2 digit number for you right margin (*rmargin*) and the text to be outlined (*your text*). Both need to be provided twice. The right margin is provided in character positions NOT points. The last two lines of the escape sequence vary depending on wheather you wish an odd or even page.

ESC S ESC B	**Save state and position**
ESC g 095 CR ETB ESC b *rmargin* +04	**Draw shaded rectangle**
ESC g 000 ESC b *rmargin* +04	**Draw box outline**
ESC A	**Position**
Odd page (right justify)	
ESC f08 ESC p +20 ESC g 100 DC2 *your text* CR	**Put text in white**
ETB ESC g 000 ESC z 1 DC2 *your text* ESC R	**Put text in outline**
Even Page	
ESC f08 ESC p +20 ESC g 100 *your text* CR	**Put text in white**
ETB ESC g 000 ESC z 1 *your text* ESC R	**Put text in outline**

See Also:

SPOOL
SPOOLER

LS - List directory

Syntax:

ls [*directory*] [*options*]*
+modified **-s**ort **p=**[^]*pattern* **+u**nused **-d**ir_off **+d**ir_on
+age_sort **+S**ize_sort **+r**everse_sort **+s**ize **+b**locks **-A**ll
c=*columns* **-c**olumns_off **+f**ile_list **+h**orizontal **+v**erbose
+executable **-e**xecutable **+M**odified_only **+c**lear_screen
w=*column_width* **+t**x_time **b=***baud_rate* **-m**odified **l=***line_length*

Examples:

ls +v
ls /cmds
ls /user/bill/dir1 -s
ls +M p=*.c +c

Description:

LS displays a sorted list of all the files and directories which reside at the current (or specified) directory level. Directory names are preceded by a plus sign (+) indicating that there is a substructure below it.

There are an almost unbelievable number of options:

Options:

+modified - Highlight files that have been modified since last backup. These files are shown with enhanced video mode or with a leading star (*) if enhanced is not available. The other files are also shown.

-modified - Show only files that have not been modified since last backup. Setting this flag also supresses the display of directories.

+Modified - Show only those files that have been modified since last backup. Setting this flag also supresses the display of directories.

-sort - Supress all sorting of the output. Files appear in the order in which they exist in the directory file.

+unused - List all file entries which are unused. These files may be

recoverable if they were recently deleted.

-All - Don't list all files. Files beginning with a '.' character will be excluded from the file list.

+directories - Directories on. List only directory files in the output.

-directories - Directories off. Do not include directory files in the output.

+age_sort - Sort by age (most recent first) rather than by file name.

+Size_sort - Sort by file size (largest first) rather than by file name.

+reverse_sort - Reverse the sort order to get reverse alphabetic, oldest first, or smallest first, as appropriate.

+size - Show file sizes. This mode is active whenever verbose mode is turned on. If verbose is off, then the file size display must be selected with this flag.

+blocks - Show block counts. If file sizes are being displayed, this flag will cause the file size to be displayed in blocks rather than in bytes.

c=*columns* - Used to manually set the number of columns in the output. If the number of columns is set to a large number, then a stream of file names will result. A column setting of 0 has no effect.

-columns - Turn all multi column output off. This forces the output to be displayed as one column. This has the side effect of asserting horizontal output mode, which forces the '\n' after each file name is output.

+file_list - Display a list of files, excluding any directory files, modified notation or file sizes in one column. This command is synonymous with "ls -v -c -d +h" and is used as a short hand notation for it. This option is useful to prepare lists of files for other commands to process. Executing "ls +f p=*.[ch]" is a quick way to get a list of all the source files in a directory.

+horizontal - Horizontal file list. This causes the file names to be listed across the columns rather than down the columns.

+verbose - Display the output complete with directory name, file sizes and a usage summary.

+executable - Executable files on. List only executable files in the output.

-executable - Executable files off. Do not include executable files in the output.

+clear_screen - The screen is cleared before the directory is displayed.

w=*width* - Used to manually set the width of the columns in the output. Changing the column width causes the number of columns displayed to also be recalculated. This recalculation can be supressed by manually setting a column count with the **c=** option. A column width of 0 is quietly ignored.

l=*line_length* - Used to manually specify line length. This is useful for sending the output of this command to a printer that may have a 132 or 240 character line. Also useful for squeezing

the output onto a 40 column screen if necessary.

+tx_time - Used to cause the transmission time of a file to be displayed. The default baud rate used for this calculation is 1200 baud. The **b=** option can be used to specify a baud rate. While this mode is active, directory display is turned off.

b=*baud_rate* - Used to manually set the baud rate used for transmission time calculations. If a **b=** option is specified on the command line, the time display mode is automatically selected without **+t** having to be specified.

p=[^]*pattern* - List all files matching the *pattern* described. This pattern match is applied to the files that make it through the flag settings. For example, if the **-d** option is set, even directory files which match the pattern will not be included in the output. The '^' character can be used to negate the pattern match. Character ranges can be specified in the form *[a-e]*. This would be expanded to *[abcde]* for matching purposes. Multiple ranges and the '^' option also work. Filename patterns can be any valid filename, with the following "wildcard" characters:

*	**will match any character, or run of characters**
?	**will match any one character**
[ccc]	**will match any of the characters in the brackets**

Some examples of filename patterns are:

p=*.*	**any file with a dot (.) in it.**
p=*.[ch]	**any file ending in 'c' or 'h'**
p=?a	**any two character filename ending in 'a'**
p=[a-e]*	**any file starting with 'a','b', 'c', 'd' or 'e'**
p=^*.o	**any file which doesn't end in '.o'**

See Also:

CD FILES
DIR PWD

MAKE

MAKE - Maintain and recreate files

Syntax:

make [*target* | *option* | *macro_definition*]*

Options:

f=*filename*	- Use *filename* as the description file. The default is **makefile** or **Makefile** in the current directory.
+**p**rint	- Print the complete set of macro definitions, rules and dependency descriptions.
+**i**gnore_errors	- Ignore error codes returned by invoked commands.
-**v**erbose	- Do not print command lines before executing.
-**e**xecute	- Print all command lines (even those beginning with an "@") but don't execute them.
+**t**ouch	- Change the dates of the target files instead of executing the usual commands.
+**u**nconditional	- Ignore the file dates (assume every target file and dependent is out of date) and recreate everything.
d=*directory*	- Change directory to *directory* before doing anything else. This allows MAKE to be used recursively in other directories.
+**d**ebug	- Debug mode. Print out detailed information on files and times examined. (This is usually used when the description file is not performing as desired).

All other arguments are treated as target files to be "made", unless they contain an "=" sign, in which case they are a macro definition (see the section on **macros** for more details).

Examples:

```
make
make -e f=my_makefile
make +t x.o y.o core
```

Description:

MAKE is used to maintain up-to-date versions of target files by ensuring that all of the files on which the target file depends exist and are up-to-date. If any of the dependents have been modified more recently than the target file, it is (re)created using specified commands or internal rules. MAKE performs a depth-first search of the dependencies, examining the date and time that the files were last modified.

Usage

MAKE reads a description file that defines the way that the components of a program depend upon each other, and what commands must be executed to recreate each component.

If no **f=***file* option is given, the files **makefile** and **Makefile** in the current directory are tried in order.

Description Files

A description file consists of the following information:

- **macro definitions**
- **dependency information**
- **executable commands.**

The basic form of a simple description file is:

```
dependency line
    command line
    command line
            .
            .
```

This may be repeated as many times as needed.

The syntax of a dependency line is one or more target file names, followed by a colon (:), followed by the names of zero or more files on which the target(s) depend. This can be expressed as:

target **[*target*]* : [*dependent*]***

Some examples of dependency lines are:

```
x.o   : x.c  header.h  /dir/file.h
core  : a.o  b.o  c.o
```

If you wish to use a full path name for one of the targets, the colon between the drive and path must be escaped using a backslash (\), such as:

3\:/user/... : dependents

Zero or more command lines follow dependency lines. Command lines consist of a tab, optionally followed by an "@" sign or a "-" or both (in any order), followed by an executable shell command. The printing of a specific command is suppressed if the command line begins with an "@" sign, unless the **-e**xecute option is specified. MAKE normally terminates if any command returns an error code, unless the **-i**gnore_errors options is specified, or the command line begins with a "-". The "@" or "-" is **not** part of the command.

The syntax of a command line can be expressed as:

tab[**@**|**-**]**command_line*

Because each command line is passed to a separate invocation of the shell, care must be taken with certain commands (e.g. **cd** and shell control commands such as **path**) that have meaning only within a single shell process. The effects of these command are forgotten before the next line is executed. However, you can place multiple commands on the same line, separated by a semi-colon (;). For example:

cd my_dir;cc myprog

All commands on a line such as the one above, will be executed by the same shell, so **cd** and **path** commands will take effect, for the other commands on that line.

The comment convention is that a sharp (#) and all characters on the same line after a sharp are ignored. A comment **cannot** be placed at the end of a command line. Blank lines and lines beginning with a sharp (#) are totally ignored. If a noncomment line is too long, the line can be continued by using a backslash (\) as the last character of the line. In that case, the backslash, the new line, and all following blanks and tabs are replaced by a single blank.

A target can appear on more than one dependency line, with the dependents on the additional lines adding to the previously defined dependents. Although not as useful, any commands related to the additional dependency lines also add to the end of the list of previously defined commands for that target file.

An example of this is:

```
x.o y.o: a.h
x.o: b.h
```

which is the same as:

```
x.o: a.h b.h
y.o: a.h
```

The following description file example says that **print_setup** depends upon three object files: **print_setup.o**, **small.o**, **normal.o**; each of the objects depend upon their source files and, in addition, **print_setup.o** and **small.o** depend on the header files **setup.h** and **print.h**.

```
# Description file for print_setup program

print_setup:    print_setup.o small.o normal.o
    cc print_setup.o small.o normal.o

print_setup.o: print_setup.c setup.h print.h
    cc -c print_setup.c

small.o:        small.c setup.h print.h
    cc -c small.c

normal.o:       normal.c
    cc -c normal.c
```

To make the file **print_setup**, assuming that the above description file is in a file with the name **makefile** or **Makefile**, you would enter:

```
make print_setup
  or
make
```

The second example will use the first target mentioned in the description file (print_setup). It is also possible, though less common to type:

```
make small.o
```

to only remake a single object file.

As more features of MAKE are described, you will be shown how to simplify this description file.

Macros

MAKE has a limited form of macro processing that includes string substitution. Macros are defined in the description file by:

macro_name = value_string

Macros may also be defined in the argument list passed to MAKE. Any argument that contains an "=" is assumed to be a macro definition, with the exception of the **f=** and **d=** options. Special care must be exercised when defining macros in the arguments. The following guidelines should be followed:

1) No spaces are allowed between the first '=' and the macro name or the value.
2) When the macro definition includes imbedded blanks, the entire macro definition should be enclosed within quotes.
3) The macro name is all of the characters before the first '=' and the value is all of the characters after the first '='.

For example:

make CC=cppo "LFLAG=+v l=mylib"

This command will define two macros CC and LFLAG with the following values "cppo" and "+v l=mylib".

Macros may be used in any of the dependency, command or macro definition lines (resulting in nesting of macros) and are recognized whenever $(*macro_name*) appears in the input. If the macro name is one character the parentheses are not required. The following are all valid macro invocations:

$(CFLAGS)
$2
$(xy)
$Z
$(Z)

The last two invocations are identical. A **$$** is a simple dollar sign ($). Care must be taken when using macro nesting; recursion (repeatedly nesting the same macro) is not tested for.

The following is an example of a use of nested macros:

```
HEADERS1 = x.h y.h z.h
HEADERS2 = a.h b.h c.h
ALL_HEADERS = $(HEADERS1) ($HEADERS2)
```

There are a number of macros built into MAKE which are initialized when MAKE is started. They may all be redefined within a makefile or on the command line. Most of these initialized macros are used for the transformation rules (discussed later), but two of them deserve special attention: **CWD** and **SHELL**.

The macro **CWD** is initialized to the current working directory in effect after MAKE processes its command line options (to give the "d=" option time to take effect). It is occasionally useful within command lines.

The macro **SHELL** is initialized to the default shell in use by the user. However, it may be reset to indicate which shell should be used for interpreting command lines when executed. For example, if you are using shell (user interface) other than the default shell supplied with QNX, it may not be able to interpret certain command sequences (such as "#u" - your userid) used in a particular makefile, which expects that you are using the default shell. To condition make to always use a particular shell (for example, the QNX default) instead of the user default, include a line like:

```
SHELL = /cmds/sh
```

The previous description file example can now be simplified to:

```
# Description file for print_setup program

OBJECTS = print_setup.o small.o normal.o
HEADERS = setup.h print.h

print_setup:    $(OBJECTS)
    cc $(OBJECTS)

print_setup.o: print_setup.c $(HEADERS)
    cc -c print_setup.c

small.o:        small.c $(HEADERS)
    cc -c small.c

normal.o:       normal.c
    cc -c normal.c
```

Invocations of macros in dependency lines are expanded immediately. Macros in macro definitions are expanded when used. Macros in command lines are expanded immediately before the command is executed.

A macro that has never been assigned, expands to a null string on invocation.

Macro Redirection

In QNX, it is frequently useful to have a list of file names stored in a file (an index file). For example, a list of object files to be linked together may be stored in a file called **ofiles**. However, this same list of objects may be needed by MAKE, so they can be made before linking. To avoid the need to maintain the list in the index file and the MAKE description file, it is possible to have the contents of a macro read from a file. The syntax for this is:

macro = @filename

An example is:

OBJECTS = @ofiles

Suffixes, Implied Dependencies and Internal Macros

The only thing MAKE knows about file name suffixes, is that they begin with a ".". To compensate for this lack of understanding, it has an internal table of suffixes for files which are frequently used with MAKE. As MAKE examines the file dates and times, it also matches the suffixes in the table against the file names to try and extrapolate from the dependencies in the description file to other (implied) dependents. The information extracted and built is placed in a series of internal macros.

The suffix table has the form of a dependency line where the target is **.SUFFIXES** and the dependencies are the suffixes. Naturally, there are no command lines for **.SUFFIXES**.

The default suffix table looks like:

.SUFFIXES: .o .c .c~ .h .h~ .a .a~ .y .y~ .b .b~ .exe

where the suffixes represent:

.o	**Object file**
.c	**C source file**
.c~	**C source file in an archive**
.h	**Header file**
.h~	**Header file in an archive**
.a	**Assembler source file**
.a~	**Assembler source file in an archive**
.y	**Yacc-C source grammar**
.y~	**Yacc-C source grammar in an archive**
.b	**BASIC source file**
.b~	**BASIC source file in an archive**
.exe	**DOS .exe file**

As QNX supports more languages, this table will be extended. MAKE will list the internal tables, macros, etc. if **make +p** is entered.

Suffixes ending with a tilde (~) exist to support a source code revision control system, that may appear in the future.

The order of the suffix list is important, since the list is scanned from left to right. New suffixes can be appended to the table placing an entry for **.SUFFIXES** in the description file, such as:

```
.SUFFIXES: .xyz
```

This will add the new suffix ".xyz" to the usual list. A **.SUFFIXES** line without any dependent suffixes deletes the current list. It is necessary to do this to remove suffixes from the table, or to change the order of search. An example is:

```
.SUFFIXES:
.SUFFIXES: .o .c
```

By reducing the number of suffixes in the table, it is possible to improve the performance of MAKE when working with very large systems. When it is possible to define new rules, this facility will be useful for adding suffixes for new languages which MAKE is not already programmed to handle.

The current target file name is placed in the macro **$@**, and the stem (the portion of the target file name, less the suffix as found in .SUFFIXES) is placed in **$***. If no suffix was matched to the target file name, **$*** contains a null string ("").

Once a stem has been identified, in turn, each of the suffixes are concatenated to the stem and the resulting filename is matched against the dependents specified in the description file, to try to find the name of the "source" dependent. If an explicit "source" dependent isn't found, the same procedure is tried against the file names in the same directory as the target file in an attempt to find an implicit "source" dependent. If a "source" dependent is found, its name is placed in the macro **$<**, else it will expand to a null string.

As MAKE checks each of the explicitly stated dependents, it places the names of the dependents younger than the target file in the macro **$?**.

The following table summarizes the internal macros:

$@ - The target file name.
$* - The target stem (the file name less the suffix).
$< - The "source" dependent (the stem plus a suffix).
$? - The explicitly stated dependents, younger than the target.

With the search for implied dependencies and the internally maintained macros, it is now possible to reduce the example above to:

```
# Description file for print_setup program

OBJECTS = print_setup.o small.o normal.o
HEADERS = setup.h print.h

print_setup:            $(OBJECTS)
    cc $(OBJECTS)

print_setup.o small.o:  $(HEADERS)
    cc -c $<

normal.o:
    cc -c $<
```

Notice that the source file names are never mentioned, but are implied by the suffixes.

When MAKE scans the arguments it is passed, the options specified (with the exception of **f=**, **+p** and **d=**, the target names and macro definitions) are placed in the internal macro **$(MAKEFLAGS)**. This is for use with hierarchical makefiles. See the section on **Recursive Makefiles** for details.

Extensions to $*, $@, and $<

To the list of internally generated macros, the following related macros have been added: **$(*F)**, **$(*D)**, **$(@F)**, **$(@D)**, **$(<F)**, **$(<D)**. The "D" refers to the directory portion of the single letter macro. The "F" refers to the file name part of the single letter macro. They are useful for maintaining source and objects in different directories, or in conjunction with hierarchical makefiles, as in:

```
$(MAKE) $(MAKEFLAGS) d=$(<D) $(<F)
```

An alternative to the above, would be the line:

```
cd $(<D);$(MAKE) $(MAKEFLAGS) $(<F)
```

Both of these examples will cause MAKE to change directory to the directory **$(<D)** before reading the description file and making the target **$(<F)**. The second example (with the **cd** command) will temporarily change directory at the shell level. The current working directory will revert to the original directory MAKE is being run in at the end of the line. See the section on **Recursive Makefiles** for details on MAKE invoking MAKE.

Transformation Rules (Implied Commands)

If a "source" dependent (the macro **$<**) was identified and there are no commands associated with the current target (the macro **$***), a transformation rule is selected based on the suffixes of the target and dependent file names. The name of the rule is made by concatenating the suffix of the source to the suffix of the dependent (i.e. the rule to transform a *.c* file to a *.o* file is **.c.o**).

Rules have the syntax of a dependency line and command line(s), where the target name is the rule name and the command line(s) are the commands to be executed if the rule is invoked. Naturally, a rule has no dependencies. MAKE version 2.1 and beyond will allow rules to be redefined and new rules to be created, but this facility is not yet supported.

The **.c.o** rule is expressed as:

```
.c.o:
   $(CC) $(CFLAGS) -c $<
```

This means, execute the command in the **$(CC)** macro (default: "cc") with the arguments in the **$(CFLAGS)** macro (default: ""), as well as the argument "-c" (don't produce a core file) on the file whose name is in the macro **$<**.

An example of a complex description file follows. This is the file used to maintain the MAKE command. The only transformation rule it uses is the **.c.o** rule described above. The file contains:

```
#
# Create the make command in the current directory
#

CFLAGS =    +optimize

SOURCES =   main.c make.c misc.c read_file.c input.c rules.c definitions.c
HEADERS =   manif.h struct.h extern.h
OBJECTS =   main.o make.o misc.o read_file.o input.o rules.o definitions.o
INSDIR =    /cmds

core:           compile_date.o
    $(LD) $(LDFLAGS) $(OBJECTS) compile_date.o
    @patch core +p

install:
    copy core $(INSDRIVE)$(INSDIR)/make

print:          $(SOURCES) $(HEADERS)
    list w=80 +r $?
    chattr $@ +d s=+m

compile_date.o: $(OBJECTS)
    $(CC) +p -c $<

$(OBJECTS):  $(HEADERS)
make.o:      /lib/task_msgs.h
misc.o:      /lib/io.h /lib/lfsys.h /lib/dev.h
rules.o:     /lib/account.h /lib/task_msgs.h /lib/systids.h
```

The following output results from typing **make** in a directory containing only the source and description file:

```
MAKE: cc -c definitions.c
MAKE: cc -c rules.c
MAKE: cc -c input.c
MAKE: cc -c read_file.c
MAKE: cc -c misc.c
MAKE: cc -c make.c
MAKE: cc -c main.c
MAKE: cc +p -c compile_date.c
MAKE: cc main.o make.o misc.o read_file.o input.o rules.o
definitions.o compile_date.o
```

Although none of the source files were explicitly mentioned in the description file as being dependencies of the target **core**, MAKE found them using its suffix rules and issued the needed commands. The printing of the **patch core +p** command was suppressed by an "@" sign.

The "print" and "install" entries in the description file are useful maintenance sequences. The "print" entry prints only the files changed since the last **make print** command. A short file *print* is maintained to keep track of the time of the printing. The **$?** macro in the command line then picks up only the names of the files changed since the file date and time of *print* was changed (via **chattr +d**).

In the "install" sequence, MAKE can be installed into different directories or on specific drives as follows:

```
make install INSDRIVE=1:
  or
make install INSDIR=/free
```

The following is a listing of all the internal macros and default rules:

```
#               LIST OF SUFFIXES

.SUFFIXES: .o .c .c~ .h .h~ .a .a~ .y .y~ .b .b~ .exe

#               PRESET VARIABLES

MAKEFLAGS=-e
MAKE=make
YACC=yacc
YFLAGS=
LD=cc
LDFLAGS=
CC=cc
CFLAGS=
AS=asm
ASFLAGS=
GET=co
GFLAGS=
CWD=3:/man/util
SHELL=/cmds/sh

#               INTERNAL RULES

.c.o:
                $(CC) $(CFLAGS) -c $<
.c~.o:
                $(GET) $(GFLAGS) $<
                $(CC) $(CFLAGS) -c $*.c
                rm $<
.h~.h:
                $(GET) $(GFLAGS) $@
.a.o:
                $(AS) $(ASFLAGS) $< $@
.a~.o:
                $(GET) $(GFLAGS) $<
                $(AS) $(ASFLAGS) $< $@
                rm $<
.y.c:
                $(YACC) $(YFLAGS) +d $<
.y.o:
                $(YACC) $(YFLAGS) +d $<
                $(CC) $(CFLAGS) -c $*.c
                rm $*.c
```

```
.y~.o:
        $(GET) $(GFLAGS) $<
        $(YACC) $(YFLAGS) +d $<
        $(CC) $(CFLAGS) -c $*.c
        rm $*.c $<
.b.o:
        $(CC) $(CFLAGS) -c $<
.b~.o:
        $(GET) $(GFLAGS) $<
        $(CC) $(CFLAGS) -c $<
        rm $<
.o.exe:
        $(CC) $(CFLAGS) +D $<
.c.exe:
        $(CC) $(CFLAGS) +D $<
.c~.exe:
        $(GET) $(GFLAGS) $<
        $(CC) $(CFLAGS) +D $<
        rm $<
```

This table may be reproduced, with the output appearing on the standard output, by the following command:

```
make +p f=$null
```

Given all of the above capabilities, the description file for the print_setup program can be reduced to the following:

```
# Description file for print_setup program

OBJECTS = print_setup.o small.o normal.o
HEADERS =setup.h print.h

print_setup:            $(OBJECTS)
    cc $(OBJECTS)

print_setup.o small.o:  $(HEADERS)
```

Because **normal.o** is dependent only on its source file, it needn't be mentioned as a target anywhere in the description file. Its source file will be found by the use of the suffix table and will be made (if necessary) using the transformation rules.

Alternate Implied Source Directories

By default, MAKE expects that the source dependent be in the same directory as the target. However, this is not always the case. It is sometimes useful to have the source "scattered" across a number of directories and have MAKE search these directories to find the source. The alternate directories are specified using the pseudo-target **.ALT_DIRS**, as follows:

.ALT_DIRS: ver2.0 ver1.2 ver1.1 ver1.0

Placing the above line in your makefile will cause MAKE to search the alternate directories in specified order for an implicit source dependent after first searching the directory of the target. This example will give you some ideas as to how MAKE may be used for simple version control. (The current directory is the most recent source. If the source isn't here, start looking back at the earlier version archives).

If you are working with C source in the above scenario, you must tell the CC command where to place the object files it compiles (they default to the same directory as the source). If you specify the following C compiler flag:

O=$(@D)

for example:

CFLAGS = O=$(@D)

then when C source is compiled to object, it will automatically be placed in the "target" objects directory (usually the current directory).

Recursive Makefiles

Another feature of MAKE is the ability to have make invoke itself. This is useful if the description file is too complicated (or too large). If a command line contains the sequence "$(MAKE)" anywhere, the line will be executed even if the **-**execute flag is set. Since the **-**execute flag can be carried across invocations of MAKE via the **$(MAKEFLAGS)** macro, the only command that will actually be executed is the MAKE command itself. For testing purposes, **make -e** can be executed and everything that would have been done will be printed, including output from lower level invocations of MAKE.

An example of a command line invoking another level of MAKE is:

$(MAKE) $(MAKEFLAGS) f=makefile2

It is also occasionally useful to have a target dependent on a file which is maintained by a makefile in another directory. The usage of the **d=** option to facilitate this is described in the section on **Extensions to $*, $@ and $<.**

See Also:

CC

MKDIR

MKDIR - Make directory

Syntax:

mkdir *directory* [*size*]

Options:

size - Initial size of directory (default: 10 entries)

Examples:

mkdir dir1
mkdir 3:/cmds/backup 20

Description:

MKDIR will create a directory with the given name. If a simple name is given, the directory will be at the current directory level. If the name starts with a slash (/), then the complete pathname from the root of the file system is specified. A drive number followed by a colon (:) can precede the pathname which forces the directory to be made on the given drive. Otherwise, the directory will be made on the first drive it finds which matches the prefix. For instance, the second example above will create a directory "/user" on drive 3. Without the "3:", the directory would be created on the first drive searched, which is often a ramdisk.

Space is initially reserved for 10 files if no size is given. Note that directories, like files, will grow so that this size parameter does not limit the size of a directory. Directories can contain any number of files, but may become fragmented if the initial size is not made large enough which often implies a speed penalty when opening files.

mkdir 2:/cmds 100

See Also:

DREL FREL RMDIR RM

MORE - Display a file or stdin by pages

Syntax:

more [<stdin | *filename* | **x**=*filename*]* [*options*]*
options: **+n**umbers **+r**aw **-t**ab

Options:

filename	- Name of file to print (default: keyboard)
x=*filename*	- Name of index file that names files to browse
+numbers	- Display line numbers
+raw	- Don't interpret escape sequences
-tab	- Don't expand tab characters

Description:

MORE is a file browser that uses the high speed terminal display routines to display a file on the console. While the file is being viewed on the console, it is also being stored in a large, dynamically allocated buffer. This buffer allows text that has scrolled off the screen to be "rolled back" for later review. Note that only the last 60K (approximately) of viewed text can be reverse scrolled.

After the first page of output has been displayed, various commands are available. The most used of these commands would be those that cause the display of more text to be started and stopped. This can be done a number of ways. The space bar can be pressed to start and stop output, the XON / XOFF (**^S** and **^Q**) characters can be used, the space bar, or the large plus key. Once the scrolling has been stopped, a menu appears at the bottom of the screen to prompt for further commands.

F1 or **'?'** - These keys can be pressed to display a help menu.

Left and **Right** cursor keys - These keys allow the text to be scrolled to the left and right.

Up and **Down** cursor keys - These keys allow forward or reverse scrolling a line at a time. New text from the file will be loaded and displayed if viewing past the end of buffered text is attempted.

Control **Up** and **Down** - Depressing the control key while typing the up or down arrows will cause the text to jump up or down by four lines. This is intended to mimic the action of the same keys in the QNX editor.

PgUp and **PgDn** keys - These keys allow viewing text in a forward or reverse direction a page at a time. New text from the file will be loaded and displayed if viewing past the end of buffered text is attempted.

Home - This key restarts the display at the top of the buffered file. If the amount of text buffered has exceeded the buffer size, then the earliest text still in the buffer will be used as the starting point for the display.

End - This key causes the text in the last page of the buffer to be displayed. Only if the actual end of file has been reached, will the text at the end of the file be displayed. If the end of file has not yet been reached, pressing the space bar will cause output to resume at that point.

'n' - This key causes the currently displayed page to be redisplayed with line numbers along the left column.

Digits **'1'** through **'9'** cause the text to be advanced by the corresponding number of lines.

'F' or **'/'** - MORE will prompt for a pattern to find within the buffered text. Once the text has been found, the screen will be updated such that the located text appears in the top line of the screen. Executing the find again will cause the next pattern to be found. The pattern describes a template which the filename must fit. Two special wildcard characters **'*'** and **'?'** may be used in defining the pattern. **'*'** will match any run of characters (or none at all), whereas **'?'** will match any single character. In addition, any characters enclosed in square brackets ([ ]) indicate that any one of those characters will match. The escape '\' character can be used to match the wildcard characters. A '' character can be used as the first character in the string to indicate that all names NOT matching the pattern are to return a match condition.

'r' - This key toggles "raw" mode on or off. MORE will by default attempt to display escape sequences in a file with the appropriate display attributes. For example, an Escape followed by a '<' character would turn **BOLD** on. See the operating system manual for other escape sequences. With raw mode turned on, the actual escape characters will be displayed rather than interpreted into the appropriate visual modes.

Tab or **'t'** - This key toggles the display of tab characters in the viewed text.

Escape key, **'e'** or **'x'** - These keys exit the file browser.

Enter key or **'q'** - These keys advance the file browser to the next named file if multiple files were specified. If only one file was specified, this key exits the browser.

The **'z'** key flips the screen through the various screen resolutions. For example, on an system equipped with an EGA display card and the appropriate graphics driver mounted, the zoom command will toggle the screen between 80 by 25 line mode and 80 by 43 line mode. When an exit from the MORE command is performed, the original display mode is automatically reselected.

Note: Browsing multiple files from a single invocation of MORE causes the displayed line numbers to count from one for the first file and continue from that point for each successive file.

See Also:

P
CAT

MOUNT - Mount a new disk drive or new memory

Syntax:

mount disk *drive*[**d**=*driver_num*] [*load_file*] [**+v**]
[**s**=*size*] or [**pa**=*os_type*]
[**h**=*heads* **t**=*tracks* **n**=*sectors/track*]
[**p**=*physical_drive*] [**c**=*ctl_addr*] [**i**=*interrupt*]
[u=user_data] [w=write_precomp_cyl]
[+large_xtnts] [-large_xtnts]

mount cache **d**=*drive_number* **s**=*size*[**k**]
mount xcache **s**=*size*[**k**]
mount bmcache **d**=*drive_number*
mount ramdisk *drive* [*load_file*] [**s**=*size*] [**-v**] [**+v**]
mount remdisk *drive* **n**=*node_number* **d**=*drive_number*
mount console [*device_number*] *new_name*
mount lib *load_file* [**+v**]
mount file *load_file* [**+v**]
mount debug [*load_file*] [**+v**]
mount mem **m**=*segment* **s**=*size*
mount float
mount

Options:

load_file - Specifies the name of a disk driver load file which can be used to control this drive. If none is specified, the floppy drivers will be used. In the case of MOUNT DEBUG, *load_file* is the name of the debugger to use which defaults to "/cmds/sys.debug". If MOUNT FILE or MOUNT LIB is used, then *load_file* is the name of the file to load.

s=*size* - Specify the storage capacity of the disk if different from the default value specified in the driver load file. This option is a short form useful when mounting special floppy diskettes. It is also used when mounting special memory. Always use the **h=, t=** and **n=** options (and never **s=**) for hard disks. The letters "k", "K", "m", M", 't' or 'T' may follow the size to indicate sizes in Kbytes, Megabytes or Tracks. Otherwise, the number refers to the number of 512 byte disk blocks (or 16 byte memory blocks for MOUNT MEM). The postfix **M** should be avoided since it provides only a very course resolution of size. The postfix **T** refers to the number of

tracks under one head. It is often referred to as cylinders as well.

m=*segment* - Starting segment of new memory being added. (top 16 bits of 20 bit memory address). This will not work in protected mode.

p=*drive* - Use this physical drive rather than drive 1.

pa=*os_type* - Use the partition information to determine the correct offset and size to mount the disk. You may still need to specify the correct **t=**, **h=** and **n=**. The os_type may be the characters **qnx, dos** or a number between 1 and 255. The characters **qnx** are the same as the number 7 and **dos** is the same as the number 1. To use this option you must have set up a partition using the FDISK command.

c=*ctl_addr* - I/O control address used by driver. This option will rarely be used since this information is in the supplied driver.

i=*interrupt* - Interrupt used by driver. This option will rarely be used since this information is in the supplied driver.

-verbose - Don't display any messages.

+verbose - Display extra information about where the driver or file is mounted.

u=user_data - Internal use.

+large_xtents - used with QNX versions < 2.06

-large_xtents - used with QNX versions < 2.06

w=*write_precomp_cyl*
- used as options to specify your own user data space when writing a disk driver.

Examples:

mount disk 2 t=80 h=2 n=8 - Mount drive 2 as a 640K floppy.

mount disk 2 s=640k - Also Mount drive 2 as a 640K floppy

mount disk 3 /drivers/disk.xt pa=qnx
- Mount drive 3 as a hard disk using the driver found in the file "/drivers/disk.xt". Mount the QNX partition.

mount ramdisk 5 s=128k - Mount drive 5 as a high speed memory disk with 128Kbytes of storage (data is stored in RAM instead of on a disk). (Remember to DINIT the ramdisk before using)

mount mem m=D000 s=128k - Add 128K of memory starting at hex address D000.

mount debug - Load the system debugger into memory.

mount console $con2 - Mount another console

mount xcache - Mount a 2k extent cache (default)

mount cache d=3 s=48k - Mount a 48k cache on disk 3

mount bmcache d=3	- Mount a bitmap cache on disk 3
mount lib /config/qdb.slib	- Mount a shared library
mount float	- Mount a floating point library
mount	- Query mounted disks and devices

Description:

MOUNT allows the user to add new disk drives and custom disk drivers into the system. In addition, memory which is not normally recognized when the system boots may be included. The MOUNT command may also be used to load the system debugger into memory, or to load any core file into memory.

If MOUNT is executed without any arguments it will print out what disks are mounted in the system, all devices which are present, and any mounted libraries.

When QNX boots for the first time, it assumes that the setting of the DIP switches within the PC correctly indicate how many floppy drives are connected to the system, and how much memory is installed. All floppies are initially assumed to be the same size as drive 1 unless you are on an AT, in which case the setup information in the CMOS ram is used. MOUNT can be used to change the "sizes" of the remaining disk drives if they differ. As in the first and second example, the number of disk blocks which can be stored on a particular drive can be changed. Only users which have a non-standard disk drive need to remount their floppy drives.

In the case of a floppy, MOUNT provides two pieces of information,

1. The physical characteristics of the floppy drive. This consists of two parameters, the number of tracks (40 or 80) and the number of heads (1 or 2). The number of sectors/track is a function of the floppy diskette, NOT the drive.

2. The default diskette format to be assumed for NON-QNX diskettes and QNX diskettes formatted with early versions of the FDFORMAT command (QNX 1.1).

The FDFORMAT command places a table in block one of the diskette which describes how the diskette was physically formatted. This information will always override that specified by the MOUNT command. In the case where a drive is mounted with 80 tracks and a diskette with 40 tracks is inserted (which contains an override table) the system will perform double stepping to correctly read the diskette.

Disk drives other than floppy disks require that a special driver be written to handle them. The MOUNT command can be used to include these drivers into the operating system. Special disk drivers are normally found on the boot disk under the directory /config.

MOUNT

The MOUNT command allows a user to create up to 15 logical disks. Only the first 8 may be real physical disks. The others may be REMOTE or adopted disks. For example, the filename "3:/tmp/test" is found on LOGICAL drive 2.

A hard disk can be partitioned into several logical volumes. The first logical volume on the hard disk must be mounted first, specifying the name of a disk driver file. Subsequent logical volumes on that physical disk can then be mounted by specifying that the same driver is to be used (**d=**). The following will mount a QNX and a DOS partition. The DOS partition would be accessed by a QNX-DOS file system task (DFS).

```
mount disk 3 /drivers/disk.xt    pa=qnx
mount disk 4 d=3                 pa=dos
```

If a driver is capable of supporting more than one physical drive, then the **p=** option can be used to force a particular physical drive to be used. For convenience, the "MOUNT DISK" form of the command with no driver specified will use the floppy driver, implicitly using the same physical drive as logical drive. Thus "mount disk 2" will mount a logical disk 2 on physical floppy drive 2.

The **mount mem** form of the command is useful in systems where non-standard memory cards are being used which may require some initialization before being included into the system. In systems with large amounts of memory, the system may take a long time to power-up due to the automatic memory check which is performed by the firmware. Some people prefer to set the DIP switches to indicate some smaller amount of memory which reduces the time required for power-up. The MOUNT command can then be used to install the extra memory into the system. However, it is recommended that if this practice is followed, regular memory diagnostics should be performed to safeguard against memory failure. On 8088 based PCs or XTs, it can be useful to install an extra 128K of ram above the video card (at D000) and an additional 64K just below the video card (at A000) for a total of 832K (640K + 128K + 64K). Many PCs have this RAM already installed on the motherboard, just waiting to be used. The commands

```
mount mem m=d000 s=128k

mount mem m=a000 s=64k
```

will "mount" this memory into QNX, ready for use by tasks or ramdisk.

The **mount debug** form of the command is used to load the system debugger into memory. The name of a different debugger file may be given as an option. Memory will be allocated for the debugger which will remain resident in memory.

The **mount file** and **mount lib** form of the command are provided to allow user compiled and linked programs to be loaded into a block of memory which is allocated by the operating system. How the file is mounted depends on the first byte of the linked loadfile. The LINKER will by default create loadfiles with a first byte equal to 1. This byte can be changed with the PATCH utility. MOUNT interprets this byte as follows:

1 : Common Code and Data (invalid in ATP) 2 : Not used 3 : Data Only 4 : Code Only 5 : Common Code and Data (CS:0000 = DS) 6 : Split Code and Data (CS:0000 = DS)

Type 4 is recommended for mounted libraries consisting of pure code (only seg 1 used in the assembler). Type 6 is recommended for libraries requiring local data (MOUNT will put the data segment to use in byte 0000 of the code segment). The **+verbose** option causes MOUNT to display the memory segments it has allocated for loading.

Mounting a cache on a disk will usually speed up accesses to that media. A good size for a cache is 48K. Mounting an xcache will cause the file system to cache the linkage information which connects the extents within files. As a rule of thumb, each xcache entry requires 16 bytes. Thus, only a few Kbytes of xcache can store many extent headers. The FILES command will display how many extents each file has, allowing you to judge how large an xcache might be useful. Xcache can significantly increase performance for large data base files.

Normally the MOUNT command will be used in system initialization, and may be conveniently included into the file "/config/sys.init" which is automatically executed when QNX is booted.

Only SUPER users can use the MOUNT command with options.

See Also:

FDFORMAT
FDISK
NACC
PATCH
SEARCH
SH
STTY

MSORT - Merge sort utility

Syntax:

msort [*file*] [*options*]*

Options:

f=*field*[,*field*]*	- Define sort fields
l=*max_lines*	- Maximum number of lines to buffer in memory
+descending	- Sort in descending order
+replace	- Replace file with sorted output
r=*output_file*	- Place sorted output in **output_file**
p=*directory*	- Pathname of a directory where temporary files are to be placed. The default is the current directory.
-verbose	- Suppress sort and merge phase messages

Where:

field is *field_offset* . *field_width*
left column is 0

Examples:

msort inventory f=0.10,10.4 +r
msort data f=10.5,4.3 r=sort_data p=/tmp

Description:

MSORT will sort files based on fixed fields in fixed records. Each record (line) is terminated by a record separator (hex 1E, '\n' in C programs) and all records **MUST** be the same size.

The sort is accomplished in two phases. The sort phase reads and sorts the input file in large chunks, appending the sorted chunks into two temporary files. The merge phase merges the sorted chunks from the two files into larger chunks in two other files. This is repeated until only one chunk remains. Unless **-verbose** is specified a dot will be printed as each chunk is processed in each phase.

Sorting is based on the ASCII values of the characters in the file. The order and definition of the fields is specified with the **f=** option. The offset is based at zero which is the left hand column. The width is based at one which will sort a single character.

The **+r** option causes the output of the sort to replace the contents of the original file. The output may be redirected to another file using the **r=** option. If neither the **+r** nor **r=** options are used, the output will go to the standard output which defaults to the screen.

The sorted output will normally be in ascending order. This can be reversed to descending order by specifying **+d**.

The **l=** option will limit the number of records MSORT will attempt to buffer in memory. It can be used to keep MSORT small at the price of efficiency. If not specified, MSORT will attempt to buffer the maximum number of records which will result in a data segment of close to 64K.

During the sort and merge phase, MSORT will create temporary files with the names

MSORT.*n.nid.tid*

where: *n* is 0, 1, 2 or 3
nid is the nodeid of machine MSORT is running on
tid is the taskid of MSORT

These files will be removed if MSORT terminates normally. Users with a large ramdisk may wish to place the temporary files there to increase performance. This may be accomplished using the **p=** option.

See Also:

SORT

MV - Move files

Syntax:

mv *source_file destination_file*
OR
mv *file* directory*

Examples:

mv *.c /c_source	- Move all c files to the directory /c_source.
mv test.c good.c	- Rename the file test.c to good.c

Description:

The first form of MV will move a source file to the destination file. This form of MV allows the file to be renamed as it is moved. If the directory of the destination file is the same as directory of the source file, the file is simply renamed in place.

The second form of MV will move multiple files to the specified directory. This provides a convenient method of moving multiple files in one simple operation. The SHELL special characters *, ?, and [...] are useful for selecting the set of files to be copied. If the error

LINE TOO LONG

is displayed, you will have to restrict the number of files moved and issue one or more commands.

mv *.[co] dir --> **mv *.c dir**
mv *.o dir

The utility WS is also useful for getting around this problem. For example:

ws "mv @dir" p=*.[co]

The following two commands achieve the same effect using the two forms of MV:

mv /test/file /release/file - destination is a file
mv /test/file /release - destination is a directory

For all forms of MV, if the directory the source file is in is the same as the final directory, then MV acts only to rename the file. If the destination directory is on another device (e.g. moving across a network or from floppy to hard disk), MV acts by copying the contents of the source file, then deleting it. If the directory is on the same device, the file is logically moved, with no data actually being copied.

See Also:

CP

NACC

NACC - Set network access to disks, devices and CPU

Syntax:

nacc [**cpu**] [$*device*]* [*drive*]* [+read] [+write]
[-read] [-write]

Options:

cpu - Control access to CPU.
device - Control access to $device.
drive - Control access to disk drive.
+read - Allow read access.
-read - Disallow read access.
+write - Allow write access.
-write - Disallow write access.

Examples:

nacc cpu +write	- Allow non-superusers on other nodes to execute commands on this node.
nacc 1 $lpt $mdm -w -r	- Remove network access from drive 1, $lpt and $mdm.
nacc 2 +read	- Allow network access to drive 2.

Description:

NACC may be used to control network access to your local disks, devices and CPU. If you remove both read and write, then network access will be denied. The NACC command **only** affects network access. It has no effect on requests which originate from the local machine.

Allowing CPU write access allows non super-users to execute tasks remotely on your machine. CPU read access is not currently defined. The default at boot time is to disallow everything.

Network access permissions do not apply to super-users (group 255).

If a non super-user tries to execute a task on a remote node, he must have previously allowed read / write access to his console. This is because his remote task will try to access his console from across the network. For example, if the command:

[4] mount

is executed from node 2, the user on node 2 must first have executed the command

nacc $con +r +w

to grant the mount command running on node 4 permission to display its output on the user's console from across the network.

See Also:

ALIVE
MOUNT
SEARCH
STTY

NET - Query machines on the network

Syntax:

net [*node_id*] *options**

Options:

node_id	- Only display information on this node.
-clear	- Don't clear the screen first.
+horizontal	- Horizontal display option.
+vertical	- Vertical display option.
+repeat	- Repeat until a key is typed.
p=*priority*	- Priority to run net (used with repeat).
s=*node_id*	- Start display at the indicated node.
e=*node_id*	- End display at the indicated node.

Examples:

```
net
net 3
net s=6 e=15 +v
net -c +r
```

Description:

NET may be used to examine the resources of machines connected together in the local area network. NET may only be used with networking versions of QNX.

In the full screen display modes (**+h** or **+v** option), many nodes are displayed and, if **+r**epeat option is specified, continuously updated on the screen. Pressing any key will cause NET to exit from repeat mode.

The **s=** option may be used to start the display at a particular node.

Note that NET will use the TCAP database to define the terminal characteristics for full screen mode. If the terminal has not been defined, NET will display:

Can't find terminal type (see TCAP utility).

You must then define or set the terminal type using the TCAP command. Not using the **+h** or **+v** option removes the need for the TCAP database.

NET displays the system resources which are free (out of total available) for each node in the network. Free memory and available tasks virtual circuits are displayed in all display modes.

The default, non full-screen mode of NET will also display the QNX version number, free memory, CPU speed, number of ttys on that node, number of signaling ports, maximum number of open files and the CPU flags for that machine (same definition as for the TSK command).

See Also:

TSK
WHO

NETBOOT

NETBOOT - Service boot requests from the network

Syntax:

netboot [*options*]* &

Options:

+**b**roadcast - Broadcast boot packets.
+verbose - Print boot requests on the screen.
f=*boot_file* - Default boot file.
r=*retries* - Number of retries on a network error.
s=*slow_down* - Pause between dumping boot packets on the network.

Examples:

```
netboot f=os.2.10pcat &
netboot +v &
netboot +b s=3 &
```

Description:

NETBOOT accepts boot requests from machines which wish to boot over the network. It must be running as a background task on the boot machine before another (booting) machine can boot. Upon receiving boot requests it supplies boot records to the requesting machines.

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

os. ***n.n r type***

where *n.n* - is the version number (eg: 2.1)
r - is the release number (single digit 0 .. 9)
***type* - is a hardware machine type**
pcat - IBM PC, AT, PS/2 or compatible
atp - IBM AT, PS/2 or compatible runing in protected mode
hv - HP Vectra (before the ES or GR series)
hvp - HP Vectra running in protected mode (before the ES or GR series)

The following are typical boot files.

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

The name of the file to boot from is set in a non-volatile ram on your network card. If the boot file name in the card is left blank NETBOOT will default to the file specified with the **f**=*boot_file* option.

Unless requested, NETBOOT will only boot one machine at a time. If you select the +**b**roadcast option, NETBOOT will send boot requests to all machines in the network. This allows more than one requesting machine to boot in parallel. There is a danger in using broadcast. It will place a load on machines which are already running and it is possible for a very fast server to overrun the network input buffer of a slow machine which is trying to boot. To avoid this you may have to to slow down NETBOOT with the **s**=*slow_down* option. You will have to experiment but we recommend numbers between 1 and 100. The +**b**roadcast option should be avoided unless you really have a need for it.

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

NETSIZE

NETSIZE - Configure Network Size

Syntax:

netsize [**f**=*floppy*] [**h**=*harddisk*]

Options:

f=*floppy* - Floppy disk drive to read from. Default = 1.
h=*harddisk* - Hard disk drive to write to. Default = 3.

Examples:

netsize
netsize f=2
netsize f=[2]1 h=[5]3
netsize h=[2]3

Description:

NETSIZE is used to configure the size of the network. It does this by prompting the user to insert each *Boot disk* and *Network Expansion disk* purchased into the indicated floppy drive. For each floppy disk inserted, NETSIZE copies the network size information to the indicated hard disk. The defaults are to read from floppy disk 1 and to write to hard disk 3, but these can be otherwise specified with the **f**=*floppy* and **h**=*harddisk* options. If the local disk drive is not compatible with the media (for example, 3.5" and 5.25" disks), a network remote floppy drive can be specified as:

$ netsize f=[3]1

This causes NETSIZE to read from floppy drive 1 on node 3 and to write the network size information to the hard disk 3 on the local machine. NETSIZE should be run such that it writes to the *3:/netdisks* directory on each hard disk in the network that a machine will be booting from.

Every QNX *Boot* disk is worth 1 node on the network and *Network Expansion* disks are worth various numbers of nodes. NETSIZE makes it possible to to combine these various disks into a larger network license. If you currently have a 10 node network, and wish to add 5 more nodes, all you need to do is to purchase 5

more network cards and a network expansion disk which is good for 5 nodes. Re-running the NETSIZE utility, and inserting the new network expansion disk when prompted will automatically upgrade your hard disk to reflect the new network size. Rebooting QNX from this hard disk will cause the new network size to take effect.

The BACKUP and TBACKUP utilities will copy the files stored within the */netdisks* directory but the copied files are not useful for network configuration. You should specify that TBACKUP and BACKUP not copy the contents of the */netdisks* directory (at least for the restore). If you are restoring from a backup to a hard disk, you will need to re-run NETSIZE with your various disks to have the node boot from its own hard disk and also participate on the network. This also protects the network from network workstations booting from floppy disk and accessing the network, since only a hard disk boot from a disk where NETSIZE has been run can participate on the network.

For a first time installation, the *3:/netdisks* directory is automatically created by the INSTALL utility before it attempts to run the NETSIZE utility. If you are manually running the NETSIZE utility it is important that the *3:/netdisks* directory be manually created first. Manually creating this directory helps to ensure that the local */netdisks* directory will be updated, and not another one on the network.

If at any time things appear to not work, ZAP the *3:/netdisks* directory on the hard disk, use MKDIR to create a new *3:/netdisks* directory and re-run NETSIZE. Also run CHKFSYS to reclaim the disk space lost by using ZAP.

See Also:

Network Installation section of the QNX Manual

NETSTATS - Display network statistics

Syntax:

netstats [*options*]*

Options:

n=*node* - The node whose statistics you wish to query. The default is your local node.
+clear - Clear the statistics after returning their current values.
+report - Generate report style output
+all_nodes - Display statistics of all nodes on the network.
+monitor - Enables monitering of network.
o=*offset* - Offset in minutes for displaying logged information.
d1=*dd-mon-yy* - Start date for displaying logged information.
d2=*dd-mon-yy* - End date for displaying logged information.
t1=*hh:mm:ss* - Start time for displaying logged information.
t2=*hh:mm:ss* - End time for displaying logged information.
d=*monitor_depth* - Monitor sample size. Default is 1000 samples.
m=*monitor_mask* - Mask of bits to monitor (hex).

Description:

NETSTATS will print out network statistics on a node in the network. These values can be used to detect network errors and network load experienced by a node. Unless you use the +clear option, the counters used for the statistics will continue to increase in size over time. You must be a super-user to clear the counters.

The following information will be displayed:

Min Packet Queue: 97

Each outgoing message is placed in an outgoing packet queue. Messages will be lost if this queue goes to zero. This field displays the minimum number of free entries the queue has experienced.

Packet Queue Overruns: 0

Each time the packet queue goes to zero and a message is lost this number is incremented. This number should remain at zero.

Network Rx Packets: 24,795
The number of packets transmitted by this node.

Network Tx Packets: 22,257
The number of packets received by this node.

Reconfigurations: 32
How many times the physical network has reconfigured since your node was booted. This value will increase every time a new node enters the network, and should not be considered an error. If this value increases when no node enters or leaves the network, it could indicate faulty networking hardware or noise on the cables (possibly caused by passive hubs).

Network Tx Errors: 7
This value indicates the number of packets which were corrupted while transmitting, or which were sent to non-existent node-ids. Non-zero values should not necessarily be considered an error, since QNX will retry several times. Every network reconfiguration COULD cause this value to increase. A poller will cause this value to increase every time a non-existent node is polled.

Network Tx Timeouts: 0
This value indicates the number of times packets were unable to be sent to another node which has a working network card, but whose software is not responding. This should be considered an error unless a poller is running on your machine.

Network Tx Aborts: 1
This value indicates the number of times QNX gave up when trying to transmit a packet. This should be considered an error unless the poller is running on your node.

Network Rx Errors: 0
This indicates how many bad packets were received. This value should always be zero.

Network Rx Duplicates: 0
This value indicates how many duplicate packets were received and rejected. There is a possibility of receiving a duplicate packet whenever a new node enters the network, or whenever the network is noisy. Non-zero values are not necessarily an error, but could indicate faulty hardware if excessive.

Network Monitor Mode

The **Network Monitor Mode** can be very useful for examining and diagnosing your network. It will keep track of a series of events related to your node in rela-

tion to the rest of the network. Here is a list of the events which are logged :

Event	Mask Bit
TX_Retry	0001h
TX_Failed	0002h
RX_Duplicate	0004h
RX_Invalid	0008h
Reconfig	0010h
RX_Failed	0020h
Timeout	0040h
Netboot_begin	0080h
Netboot_end	0100h
Net_poll_fail	0200h
Net_poll_down	0400h
Net_poll_up	0800h

Each entry is time stamped. To enable logging, enter the following at the command line (or put in your *sys.init.nn* file):

netstats +monitor

Then at any time in the future, you may query the current in memory log of events by entering :

netstats

The ***m=monitor_mask*** option can be used to select (mask) which events you wish to monitor/log. See the above table for the mask bit definitions.

The following options can be used for displaying the logged information :

o=*offset*
d1=*dd-mon-yy*
d2=*dd-mon-yy*
t1=*hh:mm:ss*
t2=*hh:mm:ss*

See Also:

NETTEST

NETTEST - Check data transmission between two nodes

Syntax:

nettest *node* [*min* [*max*]]

Options:

node - The destination node to test. You local node is always the source.
min - The minimum number of 16 bit words to send. Default 1.
max - The maximum number of 16 bit words to send. Default 500.

Description:

NETTEST will send data between your local node and a specified destination node checking the data at both ends. To accomplish this, NETTEST creates a remote task on the destination node. If you do not specify the minimum and maximum number of 16 bit words to send they default to 1 and 500. Messages of random size between these two values are generated for the test.

NETTEST should run without errors for millions of messages. Errors can occur when cables are disconnected from machines connected to a passive hub.

See Also:

NETSTATS

ONTTY

ONTTY - Create task on another tty

Syntax:

ontty *ttynum command-line*
ontty *ttyname command-line*

• *This is a local shell command* •

Description:

The ONTTY command executes the indicated command on the indicated tty. The command is detached from your terminal and inherits your accounting information and current directory. After mounting a console you may wish to create a SHELL or LOGIN on that console.

```
mount console $con2
ontty $con2 sh
   OR
ontty $con2 login
```

If this command is combined with the syntax of remote task creation, then the remote node should precede the ONTTY. You may also specify the node number as part of the device name.

```
[4] ontty $con2 ed file

ontty [4]$con2 ed file
```

These will create the editor on $con2 of node 4. The editor will attempt to read the indicated file using the search order on node 4.

If you know the device number you may directly reference it.

```
ontty 3 login
```

See Also:

MOUNT

OSCONFIG - Change operating system parameters

Syntax:

osconfig *filename* [+|-dos]*

Options:

filename - The name of the configuration file.
+dos - Enable the reserving of base memory for DOS
-dos - Disable the reserving of base memory for DOS

Examples:

osconfig 3:/config/sys.cfg
osconfig 3:/config/sys.cfg.4 +dos

Description:

OSCONFIG allows you to specify a configuration file which lets you change some of the operating system defaults such as the number of open files supported.

An operating system may boot from 1 of three sources.

1. **Floppy**
2. **Hard Disk**
3. **Network**

When booting over the network, the NETBOOT command attempts to open a file by the name

/config/sys.cfg.*nn*

where *nn* is replaced by the node number of the booting machine. If successful, the operating system will use the parameters in this file, otherwise it will use it's defaults.

Booting from disk is a little different. The configuration file must be explictly bound into the operating system using the BOOT command. This is specified as a **c=3:/config/sys.cfg.*nn*** option to the BOOT command. You may select any file name, however, we suggest you use the same convention as that used by NETBOOT. On a single machine you may omit the ".*nn*".

/config/sys.cfg - **Single machine.**

/config/sys.cfg.*nn* - **Network machine.**

The **sys.cfg** file is created and maintained by the OSCONFIG command. For example

osconfig 3:/config/sys.cfg

would create a configuration file for a single non-networked machine. It is active for network booting but does not become active for disk booting until you use the BOOT command to set it.

NOTE: When booting from hard disk you should not remove the config file name after you have selected it with the BOOT command. You may copy a new file on top of it but NEVER remove it (FREL, RM, ...) and then create a new one even if it has the same name. The BOOT command saves away the absolute starting block numbers of each of these files which will change once the file is removed. When booting over the network, the NETBOOT command opens the file through regular means and this is not a concern.

See Also:

BOOT
NETBOOT

P - Print file contents on terminal

Syntax:

p [*file*] [+raw]

Options:

file - Name of file to print (default: keyboard)
+raw - Don't expand control characters

Examples:

p joe
p picture +r
p file1 >$lpt

Description:

The P command prints the contents of a file. Output is sent to the standard output. If no input file is specified, it will default to the standard input. Any unprintable characters will be expanded into *hh* where *hh* is the hex value of that character. This expansion can be turned off by specifying the **+r** option.

See Also:

COPY
DUMP

PACK

PACK - Pack a file to reduce its size

Syntax:

pack *file** [**-remove**] [**-verbose**] [**-time**]
or
pack *<infile >outfile* [**-verbose**]

Options:

-remove - Suppress the removal of input files.
-verbose - Suppress status messages.
-time - Retain original file time. Set to current time by default.

Examples:

pack *.c	- Pack all C files in current directory
unpack *.z	- Unpack all packed files in current directory
pack <text >packed	- Pack file "text" and put packed file in "packed"

Description:

PACK compresses files into a more compact form. Any file can be packed, but program source files and text files benefit the most, typically shrinking by 35%. Files containing only a limited character set, such as dictionary files, may shrink as much as 48%. Packed files look like gibberish and must be unpacked before they can be used.

The unpacker, UNPACK, expands packed files into exact duplicates of the original.

As each file is packed, a of message of the form:

Analyzing "filename" ... packing... size = 70% of original

will be displayed. This can be suppressed by specifying the **-verbose** option.

PACK

When the first form of PACK is used each file will be replaced by a file with the same name and a **.z** appended. A filename may not exceed 16 characters so you should limit the name length of input files to 14 characters. The input file will by default be removed. You can suppress the removal by specifying the **-remove** option.

When the second form of PACK is used it will read from the standard input and write to the standard output. In this case the input file will never be removed and you have control over the name of the output file.

The data in the file is treated at the byte level rather than the word level, and can contain absolutely anything. The compression is in two stages: first repeated byte values are compressed and then a Huffman code is dynamically generated to match the properties of each particular file. This requires two passes over the source data.

The decoding table is included in the packed file, so packing short files can actually lengthen them. Fixed decoding tables are not used because English and various computer languages vary greatly as to upper and lower case proportions and use of special characters. Much of the savings comes from not assigning codes to unused byte values.

See Also:

UNPACK

PARK

PARK - Park the heads of a hard disk

Syntax:

park *drive*

Options:

drive - QNX drive number of disk to park.

Examples:

park 3 - Park heads of dive 3.

Description:

PARK is called to park the heads of a hard disk prior to turning power off the machine. A disk with parked heads will have a much smaller chance of being damaged should power fail, or if the unit is being moved.

The PARK utility will determine the actual size of the hard disk and attempt to move the heads to the inner-most track of the disk.

IMPORTANT: The PARK utility remounts the hard disk to position the heads, so should only be called when the system is idle. No other users or programs should have any open files when PARK is issued.

PASSON - Turn on password protection

Syntax:

passon

• *This is a local shell command* •

Description:

The PASSON command will cause the LOGIN command to look for the file "/config/pass" and enforce password protection on login. If this file does not exist you will be unable to login. Systems that have booted from a QNX network will have passwords on by default.

Once passwords are enabled, they cannot be turned off.

PATCH

PATCH - Patch files

Syntax:

patch *file* [*options*]*

Options:

o=*offset*	- Specify offset in file to patch (hex).
b=*byte*	- Specify new byte at this offset (hex).
s=*stack_size*	- Set the stack size for an executable program.
+change_user	- Make command run with effective user of user which owns the file.
-change_user	- Remove *change_owner* flag.
+high_load	- Force command to be loaded into high memory.
-high_load	- Remove *high_load* flag.
+privileged	- Make command privileged.
-privileged	- Remove *privileged* flag.
+remote_ok	- Allow command to be executed on a remote node.
-remote_ok	- Don't allow command to be executed on a remote node.
+shared	- Make command sharable.
-shared	- Remove *shared* flag.
+8087	- Indicate that command needs an 8087.
-8087	- Pretend that the command doesn't need an 8087.

Examples:

patch program
- Display the current patch status of the file.

patch core +p
- Make the file "core" privileged.

patch my_prog +s
- Make "my_prog" sharable.

patch data o=2 b=2a
- Patch the 3rd byte of the file "data" replacing it with the byte 2a hex.

Description:

PATCH can be used to modify the attributes of an executable file or it can be used to patch any byte in the file.

Setting the **change_user** flag will cause the command to assume an effective user number which is the same as the file. If the file is owned by the super-user, then the command will run with super user privileges regardless of who executes it. For example,

patch /cmds/date +c

will allow anyone to set the date.

Setting the **high_load** flag causes the code segment to be allocated at the top of memory. Normally code and data are allocated at the low end of memory. This flag may be set on tasks which remain in memory for long periods of time (such as CLOCK). By keeping these long lived code segments high in memory, memory fragmentation problems are reduced.

Setting the **privileged** flag allows the privileged system functions to be used. Only files which are created by the super-user AND which have been made privileged will be allowed access to these functions. The absolute disk block I/O functions of DISK_READ_BLK and DISK_WRITE_BLK require that the task be privileged.

Setting the **remote_ok** flag allows a command to be executed on a remote node even if remote task creation has be disallowed on that node. This allows information programs to be run remotely by a non super-user.

Setting the **shared** flag allows multiple instances of this command to be created without going to the disk. Instead, the code is shared and any constant data is copied form the data segment of an existing task. Each task will still have its own data segment. For example, the SHELL and the EDITOR have been made sharable. The program must not have have any initialized global variables.

Setting the **+8087** flag will cause the system to refuse execution if there is NOT an 8087 installed in the system. Any programs which contain 8087 opcodes will automatically cause this flag to be set. However, in some instances, the majority of a program may never use the 8087 and can still be used on systems which do not have an 8087 so long as these opcodes are avoided. For these programs, the **-8087** option allows the USES_8087 flag in the load file to be turned off. It is the user's responsibility to assure that these opcodes are never used on machines which do not have an 8087, as the hardware will hang forever!

See Also:

DUMP
SPATCH

PATH - Change command search path

Syntax:

path *searchpath*

• *This is a local shell command* •

Description:

Each time a command which does not start with a slash is executed by the shell, a list of directories is searched for an executable file. The default is "/cmds/" followed by the current directory. This may be changed using the PATH command followed by a list of directories surrounded by exclamation marks. For example, to search the current directory followed by "/cmds" followed by "/user/cmds"

path !!/cmds/!/user/cmds/!

and to return to the default

path !/cmds/!! *OR* path

You may print your current search path with

path ?

POLL

POLL - Poll nodes (network version only)

Syntax:

poll [+verbose] [-verbose] [**l**=*log_file*]
[**p**=*poll_period*] [**r**=*retries*] [**s**=*slow_poll_period*]

Examples:

poll &	- Run the poller.
poll +v	- Run poller with full screen display.
poll l=poll.log &	- Create a log of status changes.
poll p=20 &	- Poll nodes every 20 ticks (1 second).

Description:

The POLLER is responsible for ensuring that resources are reclaimed when a node goes down. It should be run on a node which is always UP, probably the node which is used to download other nodes, or acts as a fileserver to other nodes.

The POLLER scans all nodes which are alive and ensures that they are capable of receiving communications. If a node fails to respond to the poller in a reasonable amount of time, it is marked as DOWN and all other nodes are informed. This has the effect of releasing any resources which this node may have been used anywhere on the network.

The POLLER will also poll one DOWN node for every complete cycle of polling UP nodes. This is called the slow poll and may be eliminated by specifying "s=0". The slow poll will discover nodes which have recently booted, but at a much slower rate than it finds nodes which have crashed. The **s=** option can also be used to reduce the rate of slow polls if discovering newly booted nodes is not critical.

The POLLER will poll a node every 5 ticks which amounts to 4 polls per second. The **p=** option can be used to alter this rate, but the default rate provides good results for most network configurations. The POLLER decides that a node has crashed when it has attempted to poll the same node 3 times without receiving a response. The number of retries can be changed with the **r=** option, but 3 retries should be sufficient since QNX will always reply to poll messages immediately if it is UP, and the network should never be so busy that a poll message can't be sent and a reply received within 1 second.

The operating system will not initiate any communications with another node unless it thinks that node is UP.

When a node changes from UP to DOWN, the operating system automatically releases any resources which were being used by tasks on that node (usually files), and unblocks any local tasks which were waiting for messages from tasks on that node. This cleanup does not take place on the transition from UP to BUSY, or BUSY to DOWN.

BUSY status is useful to temporarily remove a node from the network without releasing any resources on other nodes. This might be done on a node before using the debugger to prevent the POLLER from flagging the node as down.

alive +b

When the debugging session has ended; the alive command can again be issued to tell the poller that the node is now UP and ready to receive communications from other nodes.

alive +up

When a node is first booted, it is usually a good idea to inform the poller that the node is now up rather than waiting for the slow poll to discover the fact. The poller will then broadcast this information to all other nodes in the network. To achieve this, include the ALIVE command in the "/config/sys.init" file.

alive +n

NOTE: For proper operation, only ONE poller should be running on a QNX Network.

See Also:

ALIVE CLRHOUSE KILL_VCS NACC

PRI - Set priority

Syntax:

pri *number*
pri *+number*
pri *-number*

• *This is a local shell command* •

Description:

Set the priority of commands executed by the shell. A '+' increases it while a '-' decreases it. If no argument is given the priority is set to the default of eight. The priority must lie within 4 and 15 with 4 being the highest priority. Smaller values have higher priority. The command.

pri -1

will ADD one to your priority number which will cause the command to run at LOWER priority. The priority number may be seen by using the tsk command and noting it's priority. This command does not affect the priority of the shell itself, only the commands executed by the shell.

PROMPTT - Display tty number

Syntax:

promptt

• *This is a local shell command* •

Description:

The PROMPTT command will toggle the display of your tty number before the standard shell prompt. It is useful when you have several windows mounted on the console which may be difficult to tell apart.

3 $ will be displayed if you are logged into $tty3

PRTSC

PRTSC - Print Screen

Syntax:

prtsc

Options:

none

Description:

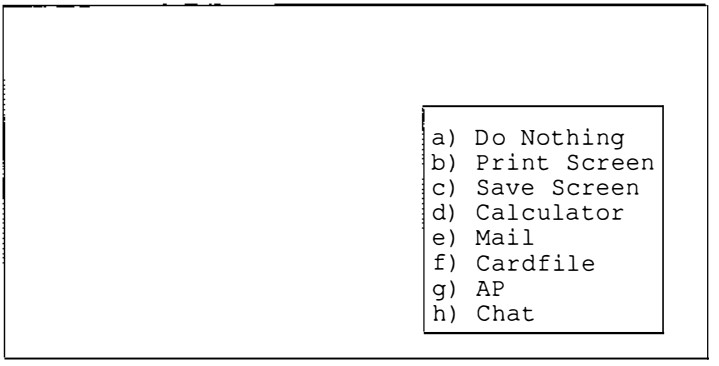
PRTSC saves the console screen, then pops-up a menu in a full screen window which allows you to select one of several options. One of these choices allows you to make a hardcopy of the saved screen. Another allows you to save the screen contents in a file.

This command is invoked automatically by QNX whenever the user types Ctrl-Alt-PrtSc on the console keyboard. The screen is restored automatically when PRTSC terminates. PRTSC can therefore be used as a means of executing commands without destroying the contents of the screen.

A file of additional menu items can be created by the user, which will be added to the menu already supported by PRTSC. This file is called **/config/prtsc.cfg** and consists of one or more text lines. The first 12 characters are used as the menu identifier. The command which starts in the 14th column will be executed when this menu item is selected. A sample file would look like:

```
Calculator   calc +f
Mail         mail
Cardfile     cardfile
AP           ap
Chat         chat
```

With the above data in the file **/config/prtsc.cfg**, PRTSC would display a menu like:



Typing the indicated letter will perform the associated function. Also, an inverse video cursor can be moved with the UP and DOWN arrow keys. Typing **Enter** will select the highlighted function.

See Also:

CALC

PWD

PWD - Print Working Directory

Syntax:

pwd

Options:

none

Description:

The PWD command will display the complete pathname of the current directory. Under the network version of QNX, the pathname will be preceded by your node number in square brackets.

See Also:

CD
LS

QCP - QNX Communications Protocol

Syntax:

qcp [*device*] **SEnd** [*options*]* *src_file*[,*dst_file*] [**x**=*index_file*]
qcp [*device*] **REceive** [*options*]* [**f**=*forced_filename* | **p**=*prefix*]

Options:

device	- Name of QNX device to use (default: current device).
-make_dir	- Suppress making directories for received files.
+newest	- Receive only files that are newer than existing files.
+relaxed_timing	- Double timeouts and quadruple retry counts.
+today's_date	- Place today's date on received files.
+verbose	- Display error status while transferring files.
-verbose	- Display nothing during the transfer.
f=*filename*	- Force received files to have this name.
p=*prefix*	- Place this prefix on the names of any received files.
s=*packet_size*	- Set size of transmitted data bursts (default: 2048).

Description:

QCP provides the error checked file transfer protocol needed by Qtalk, Qterm or QCL (QNX Communications Language) to transmit or receive files. The QCP protocol achieves both high efficiency on packet switched networks and high reliability due to the use of 16 bit CRCs (Cyclic Redundancy Check). QCP is specific to the QNX environment and automatically sends files with their path name, attribute, permissions and date fields intact.

QCP is another one of those commands that needs the timer administrator task running in the background. This can be accomplished by executing the command:

timer &

or by placing it in the sys.init file so that it is always present.

When connected to another QNX system through a serial port and running Qtalk or Qterm, files can be transferred from the remote system to the local system by giving the remote system a command of the form:

qcp se file1 file2,file3 x=file4

This example would send file1, send file2 and cause it to be received with the filename "file3" and then send all the files named in the index file "file4". Index files are easily created with the LS or FILES command. The Qtalk and Qterm programs will automatically start QCP on the local side to receive the file. To explicitly invoke QCP to receive a file, a command of the form:

qcp $mdm re

should be given. This would start up the QCP task such that it would receive the file from the $mdm device.

To send files to the remote system, Qtalk or Qterm can be commanded to invoke QCP with the appropriate options to send the file. Before starting the QCP send, the remote side must have a QCP task in a receive state. This can be accomplished by giving the remote system the command:

qcp re

If the send side of the QCP file transfer is being explicitly started from another task, rather than automatically from within Qtalk or Qterm, the send command would look like:

qcp $mdm se filename

This would send the named file(s) through the $mdm device using the QCP protocol.

While a QCP file transfer is in progress, it can be aborted by pressing the space bar or the escape key. QCP will provide a prompt to which a 'y' can be supplied to abort QCP. If a remote QCP in a receive state must be shut down, the control character sequence ^V ^X ^X will abort it.

See Also:

FILES LS QTALK

QTALK - Talk over Communications Line

Syntax:

qtalk [*system*] [*options*]*

Options:

system	- The name of a system you wish QTALK to call.
+echo	- Local echo.
+iflow	- Support flow control on the incoming data.
-parity	- Ignore 8th bit of received characters.
c=*hh*	- Specify character which invokes QTALK commands. (Default is CTRL-A).
d=*delchar*	- Replace ASCII RUBOUT with *delchar*.
l=*logfile*	- Log session into "logfile".
m=*modem*	- Name of async device (Default: $mdm).
b=*buf_size*	- Size of receive buffer. (Default: 4000).
k=*hh*	- Kick character.
p=*hh*	- Pause or turnaround character.
o=*option*	- Option(s) to be used for file transfer protocol.

Examples:

qtalk home	- Phone home. (Pretend you are E.T.).
qtalk +e d=08	- Communicate with a machine which does not echo (half duplex) and expects an ASCII BACKSPACE to delete characters.
qtalk l=$lpt	- Communicate with another system with hardcopy record.
qtalk k=11 p=0d	- Prepare to send a file to a mainframe which sends Ctl-Q when ready for input.
qtalk m=[1]$tty6	- Use the $tty6 serial port on node-1.

Description:

QTALK allows QNX users to communicate with other computers via a modem. The destination may be another host computer (mainframe) in which case QTALK allows your computer to be used as a terminal. QTALK also allows two QNX users to communicate and transfer files.

QTALK will send any characters typed on the keyboard to the other system over 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.

A recording of a QTALK session can be filed using the **l=** option.

l=$lpt
or
l=/tmp/logfile

A special COMMAND character allows special modes and options to be set while within the QTALK environment. This special character defaults to control A (Ctrl-A), unless changed with the **c=** option before entering QTALK. The character following the COMMAND character is interpreted by QTALK to perform some special function. If the next key is not a valid QTALK command, then it will be echoed to the modem. Hence Ctrl-A Ctrl-A will send a single Ctrl-A to the modem.

The QTALK commands can be one of the following:

b - BREAK
Send a break over the modem. Breaks may also be sent by pressing the BREAK key (Ctrl-BREAK on the console keyboard). NOTE: if you press the break key four times in a row, without typing any intervening characters, QTALK will terminate, as if you had invoked the **x** (exit) command.

c - CHANGE DIRECTORY
QTALK will prompt you for a new directory name, and will attempt to change directory to it. If successful, the specified directory will become the current "working directory" (see the command **PWD**), for the duration of QTALK, or until you change directory again. When QTALK terminates, your current working directory will revert to the directory you were in when QTALK was invoked.

d - DIAL SYSTEM

QTALK will prompt you to enter a system name. If that name is found in the dial.dir file, the associated string (usually a modem dialing command) will be sent to the modem. If you enter a question mark (?) for the system name, the contents of the dialing directory will be displayed, and you will re-prompted for the system name. Pressing return, without entering a system name will abort the dialing command, and return you to normal communications mode.

Dialing is implemented by looking up the system name (which can also be specified on the command line when QTALK is invoked) in a file */config/dial.dir*. This file is of the form:

system-name *dialing-command*

The system name may be any string of characters (except for spaces and tabs) of any length. The dialing command is any characters (including spaces and tabs) which are sent to the modem. Typically this would be the dialing command for your modem, and the phone number to dial. The system name and the dialing command are separated by one or more spaces and/or tabs. An example dialing directory entry that instructs a Hayes-compatible modem to call the QUICS update system would be:

quics ATD16135910934

e - ECHO

The local echo feature is toggled. Some mainframes expect the "terminal" to perform local echoing.

h - HANGUP

The CTS/RTS lines will be lowered for approximately 1/2 a second. This permits modems which support hardware hangup to do so.

l - LOG

Begin or end logging of this session. If no log file is open, then QTALK will ask for the name of a file to log into. If logging is already in progress, then it will be terminated and the log file closed. LOG will record every character which is sent OR received in the log file.

o - TRANSFER OPTIONS

QTALK prompts for the options to use when invoking a file transfer protocol. Often, the QCP "+newer" option is used to condition the transfer to ignore files which are older than the copy you already have on disk. This can be a real time-saver when downloading an update from QUICS (the QNX Software Update system).

For example:

Transfer options: **+n**

p - PARITY

Ignore parity (top bit) of received characters. If this option is already set, then tum it off.

q - QUIT and HANGUP

This does a hangup command (see above) before exiting from QTALK.

s - SEND A FILE

Send a file using the QCP file transfer protocol. This sends a file to another system running the same protocol, which is far more secure than simply writing the file to the modem.

QCP is a very secure, fast protocol specifically designed for QNX. It permits information about the file (the file date, attributes and permissions) to be transferred, as well as the file contents. QCP is ideally suited for use over public packet switch (X.25) networks, as well as direct modem-to-modem connections. If communication errors are encountered, portions of the file will automatically be resent until the far end acknowledges correct reception of the file.

More than one file can be sent by specifying **x=***filename* 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.

QTALK allows you to follow the name of the file to send with the name of the destination file separated by a comma. This is also true for file names within an index files (x=). For example:

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

will send the file "file1" as "file1" and the file "main.c" with the name "new_main.c". If no new name is given, then QTALK will create a file with the same name as the file which is sent.

Files received by QTALK using the QCP protocol will have the same attributes and date as the file on the sending machine.

Protocol transfers require that the modem port be configured for 8 bit data (see the STTY command).

u - UPLOAD TEXT

Upload text to another system. This permits text files to be sent to another computer using a simple handshaking protocol based on the *pause* and *kick* characters, specified on the command line. If local-echo is not enabled (see the **e** command, above), QTALK will wait for the character to be echoed by the remote system, before sending the next character. If local-echo is enabled, the text will be written to the modem with pauses only when the *pause* character is encountered.

A text upload may be aborted at any time by pressing the Escape key (ESC), or by pressing BREAK (Ctrl-BREAK on the console keyboard).

NOTE: you **must** specify both the *pause* and *kick* characters to use the upload handshaking protocol. If either (or both) characters are not defined, then it is the same as issuing a **w** (write) command (see below).

w - WRITE

Write a file to the modem. The file will be transmitted over the modem, and also echoed on the display if local echo is enabled. WRITE also allows files to be transferred to mainframes or other users with handshaking, but no error checking (described below).

Writing to the modem may be aborted at any time by pressing the Escape key (ESC), or by pressing BREAK (Ctrl-BREAK on the console keyboard).

The primary difference between **upload** and **write** is whether or not to wait for the character to be echoed by the remote system, before sending the next character.

x - EXIT

Exit from QTALK without performing a hangup. It is probably a good idea to get in the habit of using the **q** (quit and hangup) command for leaving QTALK, unless you really mean to not perform a hangup.

! - Execute a shell command

This permits you to execute any command from within QTALK. Typical usage of this would be to execute the **ls** command to discover what files are in your current directory prior to uploading or sending some of them, or to execute non-native file transfer protocols (such as Xmodem or Kermit) from within QTALK.

When communicating at high baud rates, the logging facilities may interfere with apparent response. The QNX operating system internally buffers up to 256 characters while data is being written to a file or printer. QTALK adds another

level of input buffering which can hold a larger number of characters. This buffer size can be changed with the **b=** option on the command line. The default buffer size is 4000 characters. Very high speed modems, or logging on slow printers or floppy disks may cause this buffer to be overrun, in which case some characters may be lost. Input flow control can be enabled (see STTY) prior to invoking QTALK, or at the time QTALK is invoked using the "+i" option, to prevent characters from being lost in these cases, provided that the machine which is sending the data supports flow control of its output.

Two methods are supported by the QTALK utility which allow users to transfer files. The simplest method is invoked using the **upload**, **write** and **log** commands to send and receive text files to/from another system. No error checking is performed so this method should only be used when communication lines are good (ie. direct connect lines or local modem connections). The **send** and **receive** commands use a more sophisticated protocol which includes error checking and re-transmission when transferring files to other systems.

Two parameters can be specified on the command line to select the protocol which is used by the **upload** command. These parameters are:

p=*hh* - **PAUSE** or turnaround character (hex). If **upload** detects this character in the file it is sending, it will be sent and QTALK will pause waiting for a KICK character to be received. Setting PAUSE to a CR (0D hex) will cause **upload** to send one line at a time.

k=*hh* - **KICK** character (hex). This character is used in conjunction with the PAUSE character by **upload** to handshake with other computers when transferring files.

These parameters allow the protocol to be adjusted according to the characteristics of the hardware at each end.

The **write** command can be used to transfer files to other systems which can accept input without needing to pause for any form of handshaking. For slower systems (especially systems running full-duplex), the **upload** command is more appropriate. The **log** command can be used to take "snapshots" of data from a host mainframe at slow speed.

To send a file to a mainframe, the user can use QTALK to enter into input mode of the mainframe's editor. The **upload** or **write** command can then be used to send a QNX file to the mainframe. QTALK should be invoked with **p=0d** to cause one line at a time to be sent. KICK should be the prompt character which is printed by the mainframe editor whenever it is ready to receive a new line. On some mainframes this may be a Ctl-Q or it may be a prompt character such as a dot (.).

The **d=** option is very useful when communicating with mainframes which have a different RUBOUT character than you are used to. Many mainframes use the backspace key (hex 08) to erase a character. QNX systems default to the ASCII RUBOUT character (hex 7F). Typing:

qtalk d=08

will cause QTALK to translate your RUBOUT key into backspace automatically.

See Also:

CHATTR
QCP
STTY

QUERY

QUERY - Query the utilization of a disk

Syntax:

query [[*node*]*drive*] [**-**address] [**+d**isplay] [**+h**ex] [**+v**isual]

Options:

-address - Don't print addresses at the side of the bitmap displayed with the **+d**isplay option.
+display - Display bitmap using character graphics and decimal addresses.
+hex - Use hexadecimal notation for the **+d**isplay option.
+visual - Maintain a real time bitmap display.

Examples:

query
query 3
query 3 +d
query [2]1 +v

Description:

QUERY allows the user to determine how much room is free on a disk. QUERY will examine the bitmap of the disk on the indicated drive and report how many blocks are used and what percentage of the total available blocks this corresponds to.

The +visual option is useful for monitoring how much space is left on a ramdisk or floppy disk while files are being copied to it. If the bitmap is too large to display on a single screen, the visual update is terminated. To exit from the continuous display mode, press any key.

See Also:

CHKFSYS

QUEUE - Implement queued message passing in QNX

Syntax:

queue [+**g**lobal] [**p**=*signal_port*] &

Options:

+**g**lobal - Globally register the queue manager.
p=*signal_port* - Specify which port to use. If unspecified, the first free port will be used.

Description:

QUEUE implements queued message passing in QNX. It should only be started if needed by applications. Although no QNX Software Systems applications are using QUEUE (as of January, 1988), this may not always be true. A number of 3rd party application developers are using QUEUE. If you have any doubt as to whether or not you should run QUEUE, check the documentation for the applications you wish to run, or contact the vendor or QNX Software Systems technical support.

If you ever need to terminate QUEUE, use the SLAY command:

slay queue

See Also:

Queue management routines in the QNX C compiler manual

RM

RM - Remove files

Syntax:

rm *file** [**+i**nteractive] [**+r**ecursive]

Options:

+interactive	- Display each file and ask whether to remove.
+recursive	- If a passed file is a directory the entire contents of the directory (including subdirectories) will be removed.

Examples:

rm *.o	- Remove all object files.
rm dir1 dir2 +r +i	- Remove dir1 and dir2. If they are directories remove all files and directories under them. Query for each file.
rm * +i	- Query whether to remove each file in the current directory.
rm *test*	- Remove all files containing the word 'test' in their name.

Description:

RM removes one or more files from a directory. Removal of a file requires write permission on the file itself.

If a designated file is a directory, an error is printed unless the +recursive option is specified. In that case, RM recursively deletes the contents of the specified directory and then the directory itself. Note that RM invokes the WS command for recursive removal. Placing the RM command in ramdisk can speed this process.

The +interactive option will cause RM to print the name of each file it is about to remove. A single character response of the letter 'y' (for yes) will remove the file. Any other response will leave the file untouched. This option may be combined with the +recursive option.

RM is a more advanced version of the command FREL.

See Also:

DREL FREL
RMDIR WS

RMDIR

RMDIR - Remove directories

Syntax:

rmdir *directory** [+interactive]

Options:

+interactive - Display each directory and ask whether to remove.

Examples:

rmdir test
rmdir test 3:/tmp

Description:

RMDIR will remove a directory from the disk. Only empty directories can be released. If the RMDIR command verifies that the directory is empty, then the space used by the directory will be reclaimed.

If a directory is to be removed which is at the top level (root), it is usually necessary to specify the correct drive prefix as in the last example.

The +interactive option will cause RMDIR to print the name of each directory it is about to remove. A single character response of the letter 'y' (for yes) will remove the directory. Any other response will leave the directory untouched.

See Also:

DREL
FREL
RM

RTC

RTC - Get Date from Real-time Clock

Syntax:

rtc *type* [+set] [+localtime]

Options:

type - Type of clock/calendar card.
+set - Set date/time of card.
+localtime - Use localtime. The default is GMT time.

Examples:

rtc at - Get QNX date from AT clock.
rtc at +set - Set AT clock with QNX date.
rtc ? - Display supported cards.

Description:

This command will set the QNX date and time from a battery backed-up clock/calendar card.

PLEASE NOTE: The types of cards supported by this program were verified at the time they were created. We do not warrant in any manner that they will necessarily work on newer versions of these cards or cards similar in design.

This command should be included in your "/config/sys.init" file if you have one of these clock/calendar cards, or if your machine has a built-in clock/calendar (such as the IBM AT):

rtc at

If the time in your clock/calendar card is incorrect (perhaps the battery has been replaced), it can be set using the **+set** option of RTC. First, set the QNX time with the DATE command, then issue the RTC command:

date 10 Mar 86 12 38 pm
rtc ast +set

The RTC command supports many different types of clock/calendar cards. To find out which cards are supported by your version of RTC, type:

rtc ?

In some cases a hardware vendor has sold many versions of the same card, such as the Quadram and AST cards. The RTC command will identify the versions with a number after the name. In general, the smaller the number, the older the card.

quad1 Oldest style quad card
quad2 More recent quad card
and so on...

See Also:

CLOCK
DATE

SAC - Display system activity at each priority level

Syntax:

sac [**-**repeat] [+scale] [+vertical] [**r**=*sec*] [**i**=*inertia*]

Options:

-repeat - Suppress continuious update of display.
+scale - Scale largest bar to screen size.
+verticle - Run bars verticaly (this is slower).
r=*sec* - Repeat rate (screen update period). Default 1.
i=*inertia* - Sensitivity to changes. Default 5.

Examples:

mount lib /config/sac.slib *Shared lib must be mounted.*

sac
sac i=8

Description:

SAC will display a bar graph showing the amount of processor activity at each priority level. In an idle system you should see one large bar at priority 15. The sum of all the bars should add up to 100% unless the **-**scale option is specified. This option expands the largest bar to fill the screen, scaling all other bars resulting in a finer resolution.

Unless you request the **-**repeat option, SAC will continuiously update the screen at the requested poll period. SAC itself will cause some processor activity. This activity will be greater if the +verticle option is selected. On the console, your TCAP entry should be set to qnx to minimize SAC's cpu useage. In general, your console should always be type qnx.

tcap set qnx

To use SAC you must mount a shared library which collects statistics on processor activity at each priortity level. This shared library integrates these numbers over time. What is displayed is the average over the integration time. Smaller values of

inertia average over shorter periods resulting in a faster response to changes in system activity. Larger values allow you to monitor average activity over a longer period of time. The averaging periods are as follows:

i=	Time		i=	Time	
5	1.5	sec	11	102	sec
6	3	sec	12	3	min
7	6	sec	13	6	min
8	13	sec	14	13	min
9	25	sec	15	27	min
10	51	sec			

The value of i must lie between 5 and 15 inclusive.

See Also:

SEARCH - Define or Query the Disk Search Order

Syntax:

search *drive** [+remote]

Options:

+remote - Define search order for requests from other nodes on the network.

Examples:

search - Query the current disk search order
search 5 3
- Define the disk search order such that drive 5 is searched first, followed by drive 3.
search 3 +r - Set remote search order to be drive 3.
search 3 [5]
- Search drive 3 then use node 5's remote search order.

Description:

SEARCH defines the "order" in which the operating system scans disk drives while opening files.

A default disk search order is setup at boot time by the operating system. If you boot from floppy diskette the disk search order is linear over your floppy drives.

search 1
or
search 1 2

if you booted from hard disk your search order will be disk 3 which is mapped to the the disk partition you booted from. Please read the section on hard disk booting for more details.

search 3

If you boot over the network your search order will the remote node you booted from.

search [*node_booted_from_*]

It is usually desirable to change the search order to include mounted ramdisks and hard disks. Most users do not search the floppy drives at all.

Adding new drives to the system with the MOUNT command will not automatically include the drive into the system search order. SEARCH must be used before these drives will be automatically scanned when looking for files. Files can still be accessed on drives which are not included in the search order by preceding the filename with a drive prefix.

If a number in square brackets is included, then the remote search order on that node is used.

Requests for files on a local hard disk can come from two sources. They can originate locally from tasks on the same machine or remotely from tasks on other machines. The +remote option determines the search order which will be used for remote requests. A local ramdisk is usually excluded from the remote search order.

See Also:

MOUNT
NACC
PATH

SH

SH - Execute Shell Commands

Syntax:

sh [*options*]* *filename* [*arguments*]*

Options:

+back - Suppress the display of background task ids.
+menu - Internal use only.
+qdb - Internal use only.
+initial - Execute the next argument as a command using the EC command, then accept input from the keyboard.
+restrict - Restrict the commands and command line syntax allowed. The commands **cd, path** and the characters >, / and ^ become invalid.
+string - Execute the next argument as a command then terminate.
+transform - Cause the shell to transform itself into the first command it executes.
+verbose - Print each command line before executing.

Examples:

sh commands
sh /cmds/mycc test.o c=test l=/lib

Description:

SH is the command interpreter (referred to as the "shell") and acts as the interface between the user and QNX. SH can take commands interactively from the keyboard or from a batch file. The following documentation concerns itself mainly with batch file operation. Interactive operation is described in a chapter in the QNX manual. This chapter also contains background information on the shell which is recommended reading.

SH allows the user to execute the commands found in a file as if they were typed on the keyboard by the user. Each line in the file will be executed as a command by the shell. The shell's arguments can be included into these commands by referencing them as #1 #2 ... #9. Any instance of #1 will be replaced by the 1st argument on the command line. For example, in the second example above, #1 is replaced with "test.o", #2 will be replaced with "c=test" and #3 will be replaced

with "l=/lib" before the commands are executed. #4 through #9 in this example will be replaced with the null string. The complete list of # macros are as follows:

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

The **#p** macro contents is defined as follows :

PC	**1**
AT real mode	**2**
PS2	**4**
old HP Vectra	**5**
PS2 model 30	**6**
AT Protected mode	**42h**
PS2 protected mode	**44h**
old HP Vectra Protected mode	**45h**

SH allows a user to define his own commands. Commonly repeated sequences of commands can be grouped into one command file thereby allowing the user to execute this group of commands with a single command.

An example of a command file would be a file called "rename" which contains the following line:

chattr #1 n=#2

The user would then type

sh rename bill joe

to change the name of file "bill" to "joe". If the command is used quite often, the user can remove the need for typing "sh" every time by giving the command file execute permission with the CHATTR command (use: "chattr rename a=+e"). The user can now type:

rename bill joe

Each command will return a value when it terminates. This value will usually be zero if the command terminates normally.

The **+i** option is used by the LOGIN command to pass the initial command to execute which was read from the password file.

If you type a Ctrl-d to the shell it will terminate. This is useful when invoking the shell from within the editor (or any command) by typing a Ctrl-z. Typing a Ctrl-d will return you to your application.

The complete list of shell batch commands follows:

back - suppress background tid

When you run a program in background using the **&** symbol the shell will normally display the taskid of the background task created. In shell files this may not be desirable. The BACK command suppresses this display.

base *number* - base for number expansions

When an integer shell variable is displayed it will be displayed as either a 4 digit hexadecimal number or a 5 digit unsigned number. The argument for base should be either **10** or **16**.

break *task_id* - break a task

The indicated task will be killed by setting a break exception on it. You may obtain the tasks identity with the TSK command.

cd [*directory*] - change directory

Since each task has its own directory this command must be handled by the shell. Placing it in a command which loaded off the disk would change the directory of the new task loaded but NOT the directory of the shell. This should be kept in mind when escaping from a command via Ctrl-z. You can NOT change your directory then return back with the new directory in effect. This is documented in detail in the utilities section of your binder.

debug [*text*] - debug a command

If text is missing then the debugger is invoked with the data and code segments pointing to 4K of free memory. If text is present then the debugger is invoked after loading the command indicated.

```
debug ls
```

The debugger must be mounted or the request for debug is ignored. Please read the debugger documentation first! If you don't, simply type 'g' followed by a carriage return to return from the debugger.

defpipe *path* - pipe temp files

The DEFPIPE command allows you to change the pathname where temporary pipe files are placed.

```
defpipe my_pipe
```

The default is

```
defpipe /tmp/..#n#$
```

ec *shell_file* - execute sh file

The EC command causes the shell to temporarily take its input from the indicated file. Since the execution of commands in the file is performed by the current shell, it is possible to execute commands like CD, PATH, PRI, PROMPTT etc..., which affect the environment of the shell which executes them.

This is especially useful in the password file where you will often see a line such as

```
ec user.init
```

EC commands cannot be nested.

else - conditional

The ELSE command precedes a group of one or more commands terminated by an ENDIF which are to be executed if the preceding IF condition was false. An ELSE must be within the scope of a preceding block IF command. For example

```
if +f #1.c then
    cc #1 m=map
    sort map f=1,3 +r
else
    type File #1.c does not exist
endif
```

endif - end a block if

The ENDIF statement ends a list of statements following a block IF or an ELSE.

exit [*number*] - exit shell with status

The EXIT command will exit the shell and return back either the indicated status or the status of the last executed command if status is omitted. The status argument is assumed to be in the current base set by the BASE command.

goto *label* - transfer control

The GOTO command transfers control to the line following the one containing the indicated label. A label is entered as a colon (:) followed by label name of up to 8 characters. For example

```
:loop
type This is a loop
goto loop
```

If the label does not exist then the shell file will terminate.

if *test cmd* - conditional

The IF command allows conditional execution of QNX commands. There are two forms of this command. If the argument *command* is the keyword THEN, then commands on the following lines until an ENDIF are considered to be within the scope of the IF. This is called a block IF. Otherwise, only the indicated command is conditionally executed. There can be no ELSE or ENDIF. If the TEST is true then the *command* is executed.

if +f ***pathname***	***command***	**- true if** ***pathname*** **is a file and exists**
if +d ***pathname***	***command***	**- true if** ***pathname*** **is a directory and exists**
if +m ***pathname***	***command***	**- true if** ***pathname*** **has the modified bit set**
if +a ***nodeid***	***command***	**- true if** ***node*** **is alive**
if eq ***arg1 arg2***	***command***	**- true if** ***arg1*** **is identical to** ***arg2***
if ne ***arg1 arg2***	***command***	**- true if** ***arg1*** **is different from** ***arg2***
if lt ***arg1 arg2***	***command***	**- true if** ***arg1*** **is less than** ***arg2***
if ge ***arg1 arg2***	***command***	**- true if** ***arg1*** **is greater than or equal to** ***arg2***

The arguments in the last two forms may NOT contain an embedded blank unless the argument is enclosed in double quotes.

```
if eq "#1" "Clarke Kent" frel
```

You may match a null (or missing) argument using the following trick.

```
if eq abc #1abc type Argument missing
```

As a final warning, remember that the # shell variables which result in a number are always expanded to 4 places if hexadecimal and 5 places if decimal.

```
if eq #& 0000 exit
```

It is possible to nest block IF's.

```
if eq #1 abc then
    if eq #2 def then
        type abc def
    else
        type abc NOT def
    endif
else
    if eq #2 def then
        type NOT abc def
    else
        type NOT abc NOT def
    endif
endif
```

kill *task_id* ... - kill a task

The indicated tasks will be killed. You may obtain the tasks identity with the TASK command. This is included in the shell to allow you to kill a task even when there are no free task descriptors in the system to create a new task. The escape sequence #& refers to the last background task you created.

```
kill #&
```

If the first argument starts with a plus sign (+) it is taken as a system exception to set on the remaining taskid's specified. For example

```
kill +0001 050c      -set exception hangup on task 050c
```

Refer the to chapter on MULTI-TASKING for more information on exceptions.

You may wish to refer to the SLAY command to remove a task by it's name rather than it's task id.

ontty *tty cmd* - create task on another tty

The ONTTY command executes the indicated command on the indicated tty. The command is detached from your terminal and inherits your accounting information and current directory. After mounting a console you may wish to create a SHELL

or LOGIN on that console.

```
mount console $con2
ontty $con2 sh
    OR
ontty $con2 login
```

If this command is combined with the syntax of remote task creation, then the remote node should precede the ONTTY. You may also specify the node number as part of the device name.

```
[4] ontty $con2 ed file
ontty [4]$con2 ed file
```

These will create the editor on $con2 of node 4. The editor will attempt to read the indicated file using the search order on node 4.

passon - password protection

The PASSON command will cause the LOGIN command to look for the file "/config/pass" and enforce password protection on login. If this file does not exist you will be unable to login.

Once passwords are enabled, they cannot be turned off.

path *searchpath* - change search path

Each time a command which does not start with a slash is executed by the shell a list of directories is searched for an executable file. The default is "/cmds/" followed by the current directory. This may be changed using the PATH command followed by a list of directories surrounded by exclamation marks. For example, to search the current directory followed by "/cmds" followed by "/user/cmds"

```
path !!/cmds/!/user/cmds/!
```

and to return to the default

```
path !/cmds/!! OR path
```

You may print your current search path with

```
path ?
```

pause - pause for carriage return

The PAUSE command will pause and wait for you to type a carriage return. When used with the TYPE command it gives you the opportunity to change disks or perform some other action before executing the next command.

pri [+|-]*number* - set priority

Set the priority of commands executed by the shell. A '+' increases it while a '-' decreases it. If no argument is given the priority is set to the default of eight. The priority must lie within 4 and 15 with 4 being the highest priority. Smaller values have higher priority. The command.

pri -1

will ADD one to your priority number which will cause the command to run at LOWER priority. The priority number may be seen by using the task command and noting it's priority. This command does not affect the priority of the shell itself, only the commands executed by the shell.

promptt - display tty number

The PROMPT command will toggle the display of your tty number before the standard shell prompt. It is useful when you have several windows mounted on the console which may be difficult to tell apart.

3 $ will be displayed if you are logged into $tty3

setvar *op var val* - set variable

The user variable is set to a new value based upon the *op* type.

String Variables

setvar	**=**	**s***n* ***value***	**- assign string**
setvar	**|**	**s***n* ***value***	**- concatenate string**

```
setvar = oranges
setvar | ", apples"
```

Integer Variables

setvar	**=**	**i***n* ***number***	**- assign number**
setvar	**+**	**i***n* ***number***	**- add by number**
setvar	**-**	**i***n* ***number***	**- subtract by number**
setvar	**x**	**i***n* ***number***	**- multiply by number**
setvar	**/**	**i***n* ***number***	**- divide by number**
setvar	**%**	**i***n* ***number***	**- modulus by number**

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

setvar = s0 a#k          - get line from keyboard, the 'a'is
                           to ensure some data is assigned in
```

case a blank line is entered.

sharoff - don't share code segments

This option will stop the sharing of code. See SHARON for details.

sharon - share code segments

This option will cause the task admin to scan the code table and check if a task is already loaded into memory. If so, it will link to the existing code and only load the data from the file. Note that this code sharing only occurs for commands under '/cmds' or those specified with a leading '/'. This the default when the system boots.

shift - shift # args down one

The SHIFT command allows shell commands access to arguments beyond #9. Each argument is shifted one position to the left. #0 becomes #1, #1 becomes #2, ... and #9 becomes what would have been #10. The following shell command will print any number of arguments passed to it.

```
:loop
    if eq ## 0000 exit 0
    type #1
    shift
    goto loop
```

stype [*text*]* - type arguments

Identical to TYPE, however, no carriage/linefeed is printed.

sysup - set system up flag

The SYSUP command will set the system up flag. Until this flag is set, Ctrl-z and Breaks will be ignored on terminals, preventing anyone from logging in, or terminating execution of the sys.init file. This flag is automatically set *after* the '/config/sys.init' file has run to completion. In some cases a user may wish to invoke a command in the 'sys.init' file (like a menu shell) which prevents it from finishing. In this case you may force the system up placing the SYSUP command in your 'sys.init' file before this command is invoked.

then - conditional

The THEN command can only be used on a line containing an IF command. It signals the start of a block IF.

trap [*label*] - trap keyboard breaks

The TRAP command allows a shell file to trap keyboard breaks. After executing each statement the shell checks to see if a keyboard break has been typed. If so, execution is transferred to the indicated label as though a GOTO statement had

been executed. If the label argument is missing then the break is ignored. If the indicated label does not exist then the shell command will terminate.

```
trap          - ignore keyboard breaks
trap +        - since label + will never occur
                this will cause the shell command
                to terminate on a break
trap loop     - transfer control to the label :loop
```

The following will print out a message until a break is typed.

```
trap stop
:loop
type Type break to stop me
goto loop
:stop
type Ok, you stopped me!
```

type [*text*]* - type arguments

The arguments following the command are displayed on the terminal, followed by a carriage return. This is often used within command files for informational purposes.

verbose - toggle verbose mode

The VERBOSE command toggles verbose mode of the shell. In verbose mode the shell will echo each line it reads to your screen before it executes it.

zeros on/off - toggle verbose mode

The ZEROS command turns on or off the printing of leading zeros when expanding numbers from shell variables. By default they are on to allow correct string comparisons in the IF command. They are usually turned off to generate text or pathnames which is more readable. For example.

```
base 10
zeros off
type "You are on tty #t"
```

See Also:

LOGIN
LOGOFF
QNX Manual

SHARON/SHAROFF - Do/don't share code segments

Syntax:

sharon
sharoff

• *This is a local shell command* •

Description:

When SHARON has been enabled, the task admin will scan the code table to check if a task is already loaded into memory. If so, it will link to the existing code and only load the data from the file. Note that this code sharing only occurs for commands under '/cmds' or those specified with a leading '/'. This is the default when the system boots. SHAROFF disables code sharing.

SIZE

SIZE - Display the size of a file

Syntax:

size [*file* | **x=***file*]*

Options:

file - Name of a file to size
x=*file* - Index file containing a list of files to size

Examples:

size test.c
size inventory mailing_list x=misc_files

Description:

SIZE will calculate the total number of characters in each of the given files.

If more than one filename is given, then SIZE will also report the total number of characters and lines in all the files.

The **x=** option allows a file containing a list of filenames to specify the files to count. This "index" file may be the result of redirecting the output of the FILES command (with -v option) or the LS command (with the +f option).

See Also:

FILES
WC

SLAY - Kill a task by name

Syntax:

slay [*task_name*] [*options*]*

Options:

+break_only	- Set only a break exception on the named task.
+dump	- Force task to perform a dump.
+hold	- Hold the named task.
-hold	- Unhold the named task.
+query_always	- Always query before killing a task.
-query	- Don't query before killing tasks.
+remote	- Also kill tasks started by remote nodes.
-son_kill	- Don't kill tasks which have son tasks running.
+tid	- Don't kill the task, return it's task id to the shell.
-verbose	- Suppress messages about missing tasks.
i=task_id	- Select task based on *task_id.*
n=*node_num*	- Search for named task on this node.
t=*tty_num*	- Limit task search to this tty.
s=*system_exc*	- System exception bits value (hex)
u=*user_exc*	- User exception bits value (hex)
p=*priority*	- Assign new priority to selected task(s)

Examples:

slay spooler n=2	- Kill spooler task on node 2.
slay dragon t=3 -q	- Kill dragon task on tty3 without querying first.
slay test +d	- Force memory dump on a test program.

Description:

SLAY is used to kill a task by name rather than by the task identification number (tid). This saves the user from having to first run the TSK utility before issuing a KILL command. Task names are specified without the path. An example would be a task called "/cmds/list" that the user wished to kill. Entering "list" as the task name would be sufficient to allow SLAY to find and kill it.

There are many forms of this command. The simplest (and probably most often used) form of the command is of the form:

slay task_name

This command will locate the task bearing the specified name and if only one is found both the KILL and BREAK exceptions will be set on it. If more than one task bears the specified name, the user will be prompted for a yes / no response for each task. When each task is listed in this form, the task name, tid, node number, tty and user group / member numbers will also be displayed to help the user make a selection. If no task name is specified, all tasks in the system will be matched and the user will be prompted for each of them. Answering the prompt with any character other than a 'y' or 'n' (with 'q', for example) will cause an exit from SLAY.

By default, tasks that are running remotely from other nodes in a network will not be included in the scan for matching tasks. If these tasks must also be killed, use the +remote option.

In more detail, the options supported by SLAY are:

+break_only - Normally, SLAY will place both a break and a kill exception on the named task. This option will cause only the break exception to be set on the named task.

+dump - This option sets a quit exception on the named task. If the dumper task has been installed a "snapshot" of the segments that make up the named task will be written to disk. QDB, the QNX Source level debugger, can then be used to browse through the execution environment at the moment of the dump. Please consult the QDB documentation for details on dumper.

+hold -hold - The hold option allows this utility to set or remove a hold status on the named task. One particularly useful example of this option is for a program which wishes to temporarily not have the CLOCK program display the time in the upper right corner of the screen. Issuing the command:

slay clock +h

would stop the time display. Once the application was finished, it could resume the time display with the command:

slay clock -h

+query - This option forces SLAY to query before killing the task even if only one task is found with a matching name. This option is useful for viewing the other task information that SLAY will present before killing it.

-query - Force SLAY to not query if multiple occurrences of the same task name exist. This makes SLAY useful in shell commands if the environment is known to be appropriate for running SLAY in this manner.

+remote - This option will allow killing local tasks that were started from another node. This option is needed to allow a user on his workstation to kill tasks that other network users have started on his node.

-son_kill - This option will suppress killing tasks that have son tasks. A typical usage of this command would be within a shell command that shuts down shells on other tty devices. Setting this option would prevent SLAY from killing those shells if they had other tasks (such as editors) running. Running SLAY with the **+query** option also set, will cause SLAY to prompt for a forced kill even if the named task has sons.

+tid - This option will cause SLAY to return the task id of the first task which bears the requested name. If the named task does not exist, a task id of zero will be returned. This option then makes SLAY useful for shell scripts and C programs which wish to determine the task id of a task without having to resort to elaborate programming. Within a shell program, the usage would be:

```
slay task_name +t -v
type #?
```

Within a C program, the SHELL library routine could be used to invoke SLAY to obtain the task id of a task as follows:

```
tid = shell( "slay task_name +t -v" );
```

-verbose - Suppress messages about a task not being found or having sons preventing the removal of the task. If the **+query** option is set, the prompts will not be suppressed. This option is useful within shell commands to suppress unwanted output.

n=*node_num* - This option specifies the network node from which tasks are to be killed. This option allows convenient removal of tasks from a remote node without having to have a shell running on that node.

t=*tty_number* - This option specifies the tty from which tasks are to be killed. Tasks running from other ttys will not be affected.

s=*system_exc* - Set the specified SYSTEM exception bits on the selected tasks. See the '/lib/exc.h' file for exception bit definitions.

u=*user_exc* - Set the specified USER exception bits on the selected tasks. See the '/lib/exc.h' file for exception bit definitions.

p=*priority* - Change the priority of the the selected tasks to *priority.*

Using the SLAY command, a very useful shell program can be created to allow a user at the console to position the first virtual console ($con0) to a particular directory, execute the shell command and cause all the other virtual consoles to also be positioned to the same directory. This is an ideal way to begin working within a particular directory.

```
" This shell script removes shells from other consoles and starts up shells in
" the same directory as the current shell. This command can only be run from
" $con0, the first virtual console. It assumes that the virtual consoles
" represent tty devices 0, 7 and 8.
if eq #t 0000 then
    slay sh t=7 -q -s
    if eq #? 0000 then
        ontty 7 sh
    endif
    slay sh t=8 -q -s
    if eq #? 0000 then
        ontty 8 sh
    endif
endif
```

See Also:

TSK

SLEEP - Sleep for a number of seconds

Syntax:

sleep *time*

Options:

time - Number of seconds to sleep.

Examples:

sleep 60

Description:

SLEEP suspends execution for *time* seconds. It is useful in shell files where a delay is needed. For example

```
"Take a snap shot of all users signed on every 10 minutes.
:loop
    sleep 600
    who net >>users
    goto loop
```

The SLEEP command requires that the timer administrator be running.

timer &

See Also:

TIMER

SLICE

SLICE - Set the timeslice rate

Syntax:

slice [*num_ticks* [*tick_size*]]

Options:

num_ticks - Number of clock ticks to allow a task to run before timeslicing (1..255). The default is 1.

tick_size - Time duration represented by each clock tick. Valid numbers are 1, 2, 5, 10, 25 or 50 milliseconds. The default is 50.

Examples:

slice 20 50 - Timeslice every 1 second

slice 50 1 - Timeslice every 1/20 second, but provide a fast timer tick rate of 1 millisecond.

Description:

SLICE allows the rate at which QNX will timeslice tasks to be altered. A large timeslice count will allow each task to have a large amount of processor time before being interrupted by the operating system to run a new task. Large timeslice counts favor tasks which require a lot of processor time and which perform little I/O.

Small timeslice values improve interactive response.

The default value of 1 results in task switching at least as fast as every 50 msec (1/20 sec), which usually provides pleasing interactive response with commands such as the editor, while still allowing background tasks a fair percentage of the processor.

User written tasks are able to take advantage of fast timer ticks by setting *tick_size* to a value representing the desired *fast_tick* duration. By requesting timeouts in units of *tick_size*, where *tick_size* has been appropriately adjusted, short duration timeouts can be easily implemented.

The current timeslice is displayed by the command:

tsk info

See Also:

TIMER
TSK

SORT

SORT - Sort files

Syntax:

sort [*file*] [*options*]*

Options:

f=*field*[,*field*]*	- Define sort fields (max 20).
l=*nn*	- Max lines to sort (default:1000).
t=*termination_characters*	- Characters which isolate fields.
+**b**ubble_sort	- Use bubble sort.
+**d**escending	- Sort in descending order.
+**f**loat	- Fields are floating point numbers.
+**h**ex	- Fields are hexadecimal numbers.
+**i**nteger	- Fields are integer numbers.
+**r**eplace	- Replace file with sorted output.
-**s**kip_leading_delimiters	- Don't ignore leading spaces.
+**u**nique	- Remove duplicate lines.

Where:

field is *field_number* [. *offset* [. *width*]]

Examples:

```
sort mailing_list
sort inventory f=1,3.2.2,2,7.1 >$lpt
sort data "t= .-," +r +d -s +u
sort keyword +b +r
sort file_list t=/ f=-1
```

Description:

SORT will sort text files based on fields. Lines can be of varying lengths but cannot exceed 300 characters. SORT uses an in-memory sort and is limited to files of about 55K characters. For larger files the utility MSORT (merge sort) should be considered.

A field is by default assumed to be a string of characters which are delimited by spaces or tab characters. The delimiter characters can be changed with the **t=** option. The fields can be specified with the **f=** option, otherwise sorting will be based on field 1 ONLY. The sorting is based on the ASCII values of the characters in the file unless the **+f, +i** or **+h** options are used. If a one of these numeric options is specified all fields are considered numeric. To sort on a mix of fields you should run several sorts. For example:

sort file f=1,2 +r
sort file f=3 +f +r

The order and definition of the fields is specified with the **f=** option. If the field number is negative it is taken as the nth field from the right (not the left). A value of -1 with a terminator of '/' allow you to sort a list of pathnames (output of FILES command) by their filename only. An offset *nn* (if given) allows the first nn characters of the field to be ignored in the comparison. If a width is given as well, then only a maximum of that many characters will be compared. Offset and width allow sorting of files which have well defined formats.

If no input file is specified, the SORT utility will take its input from the standard input. The default output of the SORT utility is the standard output. This can be redirected to a file or device such as the line printer if desired. The **+r** option causes the output of the sort to replace the contents of the original file instead of sending the output to the standard output.

The sorted output will normally be in ascending order. This can be reversed to descending order by specifying **+d**.

The **-s** option allows the user to suspend the automatic skipping of leading fill characters. This is particularly useful if the user wishes to distinguish between lines which start with fill characters (eg. spaces) and those which do not.

The **+b** option will specify that a bubble sort rather than the much faster shell sort be used. Bubble sorts have the advantage that order is preserved for identical items. This may be important for sorting keyword or "index" files.

See Also:

MSORT

SPATCH

SPATCH - Full screen patch utility

Syntax:

spatch file *filename*
spatch disk *drive block*
spatch mem *segment offset*

Examples:

spatch file /cmds/ls
spatch disk 1 1
spatch mem b000 0

Description:

SPATCH is a utility that allows full screen editing of **files, disk blocks** and **memory.** The screen will display a 16 by 16 (256) byte image of the data being examined, similar to that shown below:

```
Edit Next Prev Home Goto Find Continue Save Addr Quit      Nov 28  9:37:02 am

000000000:  2E 28 6E 65 77 29 20 53 50 41 54 43 48 20 22 46   .(new) SPATCH "F
000000010:  75 6C 6C 20 73 63 72 65 65 6E 20 70 61 74 63 68   ull screen patch
000000020:  20 75 74 69 6C 69 74 79 22 1E 2E 28 73 79 6E 74    utility"..(synt
000000030:  61 78 29 1E 09 11 73 70 61 74 63 68 10 20 20 11   ax)...spatch.  .
000000040:  66 69 6C 65 10 20 20 AE 66 69 6C 65 6E 61 6D 65   file.  .filename
000000050:  AF 1E 09 11 73 70 61 74 63 68 10 20 20 11 64 69   ....spatch.  .di
000000060:  73 6B 10 20 20 AE 64 72 69 76 65 AF 20 20 AE 62   sk.  .drive.  .b
000000070:  6C 6F 63 6B AF 1E 09 11 73 70 61 74 63 68 10 20   lock....spatch.
000000080:  20 11 6D 65 6D 10 20 20 AE 73 65 67 6D 65 6E 74    .mem.  .segment
000000090:  AF 20 20 AE 6F 66 66 73 65 74 AF 1E 2E 28 65 78   .  .offset...(ex
0000000a0:  61 6D 70 6C 65 73 29 1E 09 11 73 70 61 74 63 68   amples)...spatch
0000000b0:  20 20 66 69 6C 65 20 20 2F 63 6D 64 73 2F 6C 73     file  /cmds/ls
0000000c0:  1E 09 73 70 61 74 63 68 20 20 64 69 73 6B 20 20   ..spatch  disk
0000000d0:  31 20 20 31 1E 09 73 70 61 74 63 68 20 20 6D 65   1  1..spatch  me
0000000e0:  6D 20 20 62 30 30 30 20 20 30 10 1E 2E 28 73 74   m  b000  0...(st
0000000f0:  61 72 74 29 1E 53 50 41 54 43 48 20 69 73 20 61   art).SPATCH is a
```

The menu at the top of the screen can be used to select one of the options listed below. You can either position the inverse cursor to the desired menu item with the arrow keys and type Enter, or simply type the first letter of the command.

EDIT
Enter the data area. The TAB key will switch between hex and character data entry. The SELECT (large +) key will return you to the menu. The changed data is **NOT** updated to the disk or memory.

NEXT
Move forward 256 bytes. The PgDn key may also be used.

PREV
Move backward 256 bytes. The PgUp key may also be used.

HOME
Go to the the start of the file, disk or memory. The Home key may also be used.

GOTO
You will be prompted for an address. The type of address will depend on the the source of the data (file, disk, mem) and the address type. Moving past end-of-file in a FILE may produce unexpected results.

FIND
You will be prompted for a pattern to search for. The pattern may consist of single characters or two hex digits separated by a space.

61 62 63 d e	**- match the 5 characters 'abcde'**
a b c d e	**- match the 5 characters 'abcde'**

Typing any key during the search will stop it. **CONTINUE**
Find the next occurrence of the last pattern found. Usually used after a FIND when searching.

SAVE
Save the current screen back to the source. Without issuing this command all changes made using EDIT will be lost as soon as you leave the current screen of data.

ADDR
Toggle through three address types.

Absolute	**- Default for FILE**
Disk Block:Offset	**- Default for DISK**
Segment:Offset	**- Default for MEM**

QUIT
Leave spatch.

See Also:

DUMP
DDUMP
PATCH

SPLIT - Split a file into one or more files

Syntax:

split [*output_file*]* [**s**=*size*[,*size*]*[.]] [<*input_file*]

Options:

output_file - One or more output files
s=*size*[...] - Define number of lines to put into each output file (default: 200)
input_file - Source file which is being split

Examples:

split small1 small2 s=1000 <large
split file1 file2 file3 file4 excess <massive s=100,200,100
split first100 next100 <big s=100,100.

Description:

SPLIT allows files to be split into several smaller files. Normally, the file will be split every *size* lines, with the excess going into the last output file. The input file is the standard input which should normally be redirected from the file which is to be split.

In the first example above, the first 1000 lines of the file "large" will be placed into the file "small1", and the remaining lines will all be placed into "small2".

In the second example, the first 100 lines of "massive" will be directed into "file1", the next 200 will go to "file2", the next 100 into "file3", and the remainder will end up in "file4".

The 3rd example will place the first 100 lines of "big" into "first100" and the next 100 lines into "next100". Since the size list ended in a period (.), the remaining characters are discarded.

SPLIT will only split text files with lines no longer than 1000 characters, and has a limit of 40 output files.

SPLIT

See Also:

CAT

SPOOL - Spool files to a printer

Syntax:

spooler d=*device* [**p=***name*] [**t=***tmp_dir/*] [+clrhouse] **&**

OR

spooler "c=*command***"** [**p=***name*] [**t=***tmp_dir/*] [+clrhouse] **&**

spool [*name*] **su**bmit [*filename*|**x=***filename*] [*options*]
spool [*name*] **ca**ncel *spool_id*
spool [*name*] **ho**ld [**next**]
spool [*name*] **co**ntinue
spool [*name*] **mo**ve *spool_id position*
spool [*name*] **qu**ery
spool [*name*] **st**op
spool [*name*] **ab**ort
spool [*name*] **ki**ck

Options:

+**c***lrhouse* - If a clearing house is installed on the network, this option will cause the spooler to use it.
c=*command*- Name the command used to process and output the file. (default: spooler prints file).
d=*device* - Name the device to which the spooler is to output.
p=*name* - Spooler task number/name to use (default 0). May be a digit (0-9) or an 8 character name.
t=*tmp_dir* - Automatically delete any files which are in this directory after printing. This should be a fully specified pathname, including node number if necessary.
x=*filename* - Index file containing a list of files to spool.
filename - The name of the file to spool.
options - Arguments to pass to *command*.
spool_id - A spool-id displayed by the **QUERY** option.
position - The position in the spool queue.

Examples:

spooler c=list & - Create spooler using LIST.
spooler p=nec d=$lpt & - Create spooler with name "nec".
spool su test.c - List file 'test.c'.

spool su x=cfiles	- List files named in 'cfiles'.
spool su sales +e s=6 w=80	- List file 'sales' with options to list.
spool qu	- Query spooled files.
spool ca 24	- Cancel spool file #24.
spool mo 23 1	- Move spool file #23 to the first position.
spool 1 qu	- Query files spooled to spooler 1.
spool nec qu	- Query files spooled to spooler "nec".
spool hold next	- Finish current list then hold.
spool co	- Continue printing.

Description:

The SPOOL command is the user interface to the SPOOLER task in the system. To start the SPOOLER task you must run it in background. If you wish to spool to more than one printer you must run one SPOOLER for each printer. If you do not specify the spooler name/number it will default to 0.

The SPOOLER may be invoked with the name of a command to run (**c=**) for each file printed. This will typically be the LIST command. If you do not specify a command the files will simply be printed to the standard output (or a named device using the **d=** option) with no attempt at formatting or pagination of the data. The SPOOLER will typically be run on the local machine which contains the printer. It will register a network wide, global name which will be queried by the SPOOL command to locate the SPOOLER. The TSK command may be used to print all global names in the system.

tsk names

You can use the SPOOLER and SPOOL command for purposes other than printing. For example, you might wish to serialize compiles. This would provide a batch processing mode under QNX.

The subcommands to SPOOL are listed below. Any subcommand may contain a spooler number **n**, to indicate one of several SPOOLER tasks in the system. By default SPOOLER 0 will be assumed. Only the first two letters of a subcommand need be specified.

ABORT
Abort the SPOOLER task immediately. The current file being printed will stop and all spooled requests which have not yet been printed will be lost.

CANCEL
Cancel the request to print the file with the indicated *spool_id*. The *spool-id* is listed by the **QUERY** subcommand.

CONTINUE
Continue printing after a **HOLD**.

HOLD
Hold the printing of files. If the optional **next** argument is specified the current file will print to completion before the hold takes affect. Otherwise, the hold will be immediate. Once held, printing can be restarted by using the **CONTINUE** subcommand.

KICK
Kick the spooler onto the next print request. Any request currently being printed will be aborted.

QUERY
Query all files awaiting print.

MOVE
Move the file with the indicated *spool-id* to a new position in the queue. The *spool-id* is listed by the **QUERY** subcommand.

STOP
Stop the spooler **after** the current file is printed. All queued requests will be lost.

SUBMIT
Submit a file to the spooler for printing. The command arguments will depend on the print command invoked by that SPOOLER command. This will typically be the LIST command. Only the name of the file is submitted, **not** its contents. You should wait until the file is actually printed before deleting it.

See Also:

COPY
LIST
SPOOLDEV
CLRHOUSE

SPOOLDEV

SPOOLDEV - Create a Spooling pseudo-Device

Syntax:

spooldev [**p**=*port*] [**f**=*config_file*] [**P**=priority] **&**

Options:

p=*port* - Port to attach to for timeouts. If not specified, the first available port will be used.
f=*config_file* - File to read configuration from.
P=*priority* - Priority to run the specified command at. This option is particularly useful for use with QDOS.

Examples:

```
spooldev &
spooldev p=22 &
spooldev f=/config/spool.config.3 &
```

Description:

SPOOLDEV is an optional system administrator which implements spooling devices in QNX. Whereas *physical* devices have hardware associated with them (such as serial ports), a spooling device is a *logical* device which becomes the interface between a program and the print spool queue(s). Anything written to a spool device ends up in a temporary file, which will be processed by a user specified command when the file is closed. This command will typically submit the file to the spool queue using the SPOOL command.

SPOOLDEV is typically invoked with the **f=** option, naming a file which contains the specifications for the pseudo-devices which will be created. The syntax of each line of the file is:

dev_name [command [arguments [directory [timeout]]]]

The **dev_name** is the name by which that spooling device will be known in QNX. It is limited to 5 characters drawn from the set of **(a-z,A-Z)**. Numbers may **not** be used. The name may optionally be preceded by a dollar sign **($)** in the file. If not present, a dollar sign is assumed.

The **command** is used to process the spool file after the spooling device is closed. Typically, it looks like:

"spool submit"

with the command surrounded by double-quotes ("). This is to "protect" the embedded space between the words of the command. The default, if the user enters a null command (""), is **"spool submit"**.

The **arguments** are used as options to the **command** to control the appearance of the printed output. An example of this is shown later.

The **directory** is where SPOOLDEV writes the temporary files prior to actually submitting them for printing. The default, if the user enters a null directory (""), is **"/spool"**.

The **timeout** field is optional. If you do not wish to support timeouts, specify **0**, or omit the field. This field specifies the amount of time in seconds, that output must stop before SPOOLDEV will force a close of a spool file and process it. This option is designed for applications which open a device for a long period of time and submit several "print" jobs without opening and closing the device around each job. A classic example of this is the optional QDOS package. QDOS opens the printer and leaves it open, although multiple print "jobs" may come from the DOS programs run under it.

A reasonable value for the timeout would be 30 seconds or more, on the theory that if any program pauses for that length of time while printing, it has probably finished.

Each field may be separated by one or more spaces or tab characters. If there are embedded spaces in a field (such as the command or argument string), the field must be enclosed in double quotes (").

If SPOOLDEV is invoked without a configuration file specified, it is the same as invoking it with a configuration file containing the following entry:

$spool "spool submit" "" /spool 0

The following example will help to illustrate the use of configuration files. It implements a logical device which connects to a Postscript laser printer. These printers do not except simple text but Postscript programs as input. It is therefore necessary to take text output and turn it into a program which will then drive the laser printer. QNX provides a command called LPS which does just that and we will use it in this example.

SPOOLDEV

Line 1 interfaces to a simple dot matrix printer. Line 2 interfaces to a postscript printer and uses the Courier font. Line 3 interfaces to the same postscript printer except it prints two pages side by side in landscape mode using the Times-Roman font. It is useful for producing program listings.

```
$lpt      "spool su"
$laser    spool_list   "f=0 p=8 +p"          /tmp  30
$laserl   spool_list   "f=4 c=2 p=6 +l +p"   /tmp  30
```

In the above example, **spool_list** is a shell file which interfaces with the Postscript laser printer, as shown below:

```
setvar = s1 [1]5:/tmp/ps.tmp.#n#c
lps #* +pc >#s1
spool nec su #s1 >$null
rm #1
```

The keyword "nec" indicates that the spooler associated with the NEC laser printer is to be used (see the SPOOL command). See the LPS utility documentation for details about PostScript Laser printers.

If a user then executes the following command:

```
cp report $laser
    or
cp program.c $laserl
```

the following command will be executed in the background, after the text is fully written and the device closed, or no data is written for more than 30 seconds:

```
spool_list /tmp/file f=0 p=8 +p
    or
spool_list /tmp/file f=4 c=2 p=6 +l +p
```

where *file* is replaced with the name of the temporary file created by SPOOLDEV.

SPOOLDEV does **not** erase the temporary file. It doesn't know when the command which it invokes is finished with the file. It is up to the invoked command to erase it when done.

In the above example, the shell file will erase the temporary file created by SPOOLDEV. The SPOOLER, if started with proper options, such as:

```
spooler "c=cp >$mdm" t=[1]5:/tmp/ &
```

will automatically erase the second temporary file after printing it.

The user may specify over-riding timeouts and/or arguments when he names the spool-device. These are given as extensions to the device name as shown below:

dev_name[.timeout][,arguments]

For example:

copy newfile $laser.5,+d

specifies a 5 second timeout and an *additional* "+d" argument when the temporary file is sent to the spool queue.

One final example may prove useful. In a multi-user system, it is sometimes necessary to set up the parallel printer as a spooled device, but allow existing applications that write to **$lpt** to continue to work. If you place the following lines in your sys.init.*nn* file:

```
stty n=$prt >$lpt
spooler d=$prt t=/spool &
spooldev f=/config/spool.init
```

and the file **/config/spool.init** looks like this:

$lpt

then anything written to the device **$lpt** (formerly the parallel printer) will be placed on the spool queue, eliminating multi-user printer contention. The printer port (**$lpt**) was renamed to **$prt** so that the system may differentiate between the *physical* printer (which SPOOLER will write to) and the *logical* printer which applications will write to.

To run SPOOLDEV, the Timer Administrator must first be started.

timer &

See Also:

LIST SPOOL SPOOLER TIMER

STTY

STTY - Set TTY options

Syntax:

stty [*options*]* [>*device*] [<*device*]

Options:

+echo - Turn on echoing of input (default).
-echo - Turn off echoing.
+efunc - Precede each function and cursor key code with a hex FF.
-efunc - Do not prefix function and cursor keys (default).
+edit - Perform line editing (default).
-edit - Unbuffered, non-edited input mode.
+etab - Expand TAB into spaces on output (default).
-etab - TABS are not expanded.
+ers - Expand RS (newline) into CR/LF on output (default).
-ers - RS not expanded.
+edel - Expand DEL into BS,SPACE,BS on output (default).
-edel - DEL not expanded.
+fix - Restore all options to their default values.
+FIX - Restore all options and control characters to default values.
+igate - Turn on input gate (default).
-igate - Turn off input gate.
+mapcr - Map CR into RS (newline) on input (default).
-mapcr - CR not mapped.
+hangup - Allow hangup exceptions to be generated when carrier detect drops.
-hangup - Disallow hangup exceptions (default).
+hflow - Support flow control with hardware.
-hflow - Support flow control with XON/XOFF (CTL-Q/CTL-S) protocol (default).
+hold - Freeze tasks when switching to another virtual console.
-hold - Don't freeze tasks when switching to another console (default).
+iflow - Enable flow control of input.
-iflow - Disable input flow control (default).
+oflow - Enable flow control of output (default).
-oflow - Disable output flow control.
+lock - Allow only one task at a time to open for write.
-lock - Allow multiple tasks to open for write (default).
+monocurs - Perform cursor handling for the console as if the display adaptor

was a monochrome adaptor. Video cards which allow the use of color software on monochrome monitors may require this option to be set.

-monocurs - Perform cursor handling appropriate to the type of display adaptor installed. This mode is the default.

+noboot - Forbid users from rebooting from keyboard.

-noboot - Allow users to reboot from keyboard (default).

+nocolour - Suppress colour output for black/white monitors which are connected to colour cards.

-nocolour - Allow colour output (default).

+noswitch - Suppress console switching with the control - Alt key combinations. Programs can still output <Escape> and a console number from *1* to *n* to switch consoles under program control.

-noswitch - Allow console switching with the control - Alt key combinations.

+paged - Tum on paged flag.

-paged - Tum off paged flag.

+poll - Poll line printer (default).

-poll - Printer uses interrupts. See

+split - Cause the serial port handling within QNX to use only the RTS and CTS lines to implement hardware flow control. This frees the DTR and DSR signals for other uses.

-split - Cause the serial port handling within QNX to use all four hardware flow control signals (RTS, CTS, DTR, DSR). This is the default.

ptime=*n* - Set parallel printer poll period. The default value for this setting is 1 millisecond (expressed as 1000 microseconds). Different printers will require that this value be adjusted so that a better output rate can be achieved.

page=*n* - Define page size (default: 0 - no pagination).

rows=*n* - Set number of rows displayed on the console. On an EGA display adaptor with the EGA graphics library mounted, a **rows=***43* option is supported, allowing a 80 column by 43 line display.

cols=*n* - Set number of columns displayed on the console. If the display adaptor and mounted graphics library allow, a 132 column mode could be selected with a **cols=***132* option.

break=*hh* - Define input character which causes break (default: Ctrl-C).

esc=*hh* - Define input char which escapes to new shell (default: Ctrl-Z).

rub=*hh* - Define RUBOUT character (default: 7F hex).

can=*hh* - Define CANCEL (line erase) character (default: Ctrl-X).

eot=*hh* - Define character which causes EOF (default: Ctrl-D).

ins=*hh* - Define INSERT character (default: *Ins* on console, Ctrl-N on terminal).

del=*hh* - Define DELETE character (default: *Del* on console, Ctrl-K on terminal).

up=*hh* - Define character which recalls previous line (default: *up-arrow* on console, Ctrl-U on terminal).

down=*hh* - Define character which recalls next line (default: *down-arrow* on console, LF on terminal).
left=*hh* - Define character which moves cursor left (default: *left-arrow* on console, BS on terminal).
right=*hh* - Define character which moves cursor right (default: *right-arrow* on console, none on terminal).
type=*n* - Set console type.
1 - fast colour (don't wait for retrace)
2 - colour
3 - monochrome
ioport=*hh* - Define the I/O port of device controller or memory segment for console display.
baud=*rate* - Define baud rate (default: 1200).
par=*parity* - Define parity (**odd**, **even**, **none**, **space**, **mark**) (default: **none**).
stop=*n* - Define number of stop bits (default: 1).
bits=*n* - Number of data bits (default: 8).
inton=*n* - Enable interrupt *n* (2..15).
intoff=*n* - Disable interrupt *n*.
intpri=*n* - Define which interrupt has the highest priority.
intcp=*src*,*dst* - Copy interrupt vector *src* to vector *dst*. This option is used to allow previously unused interrupt vectors to be set to point to existing interrupt handlers. For example, to allow interrupt 5 on an AT to be used for a serial port (as are interrupts 3 and 4) the following would be used:

stty intcp=4,5

n=*name* - Change name of device ("$" optional).

Examples:

```
stty baud=9600 par=even stop=1 bits=8 >$mdm
stty +oflow +ers -edel -igate +poll >[7]$lpt
stty page=24 >$con
stty <$mdm
```

Description:

STTY allows the characteristics of a device to be modified. The features which are supported by the QNX device drivers can be enabled or disabled to better support a particular device or terminal. Device dependent parameters such as baud rate and transmit parity can also be altered with STTY. If no arguments are given, then STTY will display the current options which apply to the specified input device as shown in the last example.

Line editing features can be changed to please the user. For example, the back-arrow key is used on the IBM keyboard as a RUBOUT character. If an ASCII terminal is connected to the system, the Backspace key or some other key may be more appropriate (note that any character but ASCII DEL will not "erase" input characters from the display). Similarly, the CANCEL key may be defined as a key which clears a line on a particular terminal.

The newline character which is used by the QNX operating system is an ASCII Record Separator (RS, hex 1E). When CR is typed on a terminal, it will normally be mapped into a RS (+mapcr). On output, RS is normally expanded into CR/LF (+ers). The RS character is used within files to separate lines (records) and is referenced in "C" programs as "\n".

The BREAK, ESCAPE, LEFT, RIGHT, UP, DOWN, INSERT, DELETE, RUBOUT, CANCEL, and EOT characters all have special meaning to the operating system. The features supported by these characters can be turned off by setting these fields to zero in the device table. These characters also lose their meaning if the EDIT option is turned off.

The BREAK key causes an interruption (break) of a running program which results in the program being terminated if it is not prepared to handle breaks. The ESCAPE key causes a program to be temporarily suspended, and allows the user to type in commands to a new SHELL before the suspended program is resumed. EOT will indicate end-of-file to a program reading from the terminal.

The UP and DOWN keys allow a user to "step" through the input buffer, one line at a time. UP will go the the previous line, DOWN to the next (in case you have "over shot" the line you are looking for). The LEFT and RIGHT keys allow you to move your cursor left and right over a line of text to perform line editing. RUBOUT will erase the last character on the line. CANCEL will erase the entire line. The INSERT character toggles insert mode for the line currently being entered. Further characters will be inserted before the character at the cursor until INSERT is again detected. The DELETE character will delete the character at the cursor.

When the EDIT option is turned off, the operating system stops supporting line editing and all typed characters are passed directly to user programs without delay. The following options and special characters are only valid when EDIT is on:

Options	Special Characters		
igate	**esc**	**eot**	**left**
	rubout	**ins**	**right**
	cancel	**del**	**up**
	break		**down**

Input flow control (iflow) is only supported when both EDIT and ECHO are disabled (such as in TALK).

IOPORT is the I/O address of the control port(s) which are used by the operating system to communicate with a device. Asynchronous terminals and parallel printers use this field. In the case of console displays, this field is the segment of the display memory.

The default type for a *colour* card is TYPE=2 which will cause QNX to wait for horizontal retrace before writing characters to the screen. This eliminates "flash" on the screen for some colour cards. Specifying TYPE=1 for colour consoles will stop QNX from waiting for horizontal retrace. This will result in much faster display updates, especially in the editor, for cards which do not have the flash problem. TYPE=3 is used for monochrome cards. TYPE=0 means that no monitor is installed.

INTON allows a hardware interrupt to be recognized by the operating system (an interrupt handler must be available to service the request). INTOFF disables the interrupt.

See Also:

MOUNT
NACC

STYPE

STYPE - Type arguments on the terminal (no CR/LF)

Syntax:

stype [*arguments*]*

Examples:

stype Compiling...

Description:

STYPE will echo its arguments to the terminal. STYPE may be used within command files to display progress information. No trailing CR/LF is added, so many consecutive STYPE commands will all print on the same line. Use TYPE to finish with a CR/LF.

STYPE is a *local* command which is implemented by the SHELL and therefore does not require a command to be loaded from disk. A side effect of having the SHELL implement the command is that output cannot be redirected.

See Also:

ECHO
TYPE

TBACKUP - Archive files to QIC tape(s) (QIC02 controllers)

Syntax:

tbackup **CO**nfig **c**=*dma_channel* **i**=*ioport* **u**=*a*\|*eli*

tbackup **IN**it ["**v**=*volume_name*"]

tbackup **FI**les [*arch_dir*] [+summary] [+|-verbose]

tbackup **NA**me

tbackup **SA**ve save_spec* [+**a**ll] [-**c**lr] [**l**=*levels*] [*options*]*

tbackup **RE**store [*disk_dir*[,*arch_dir*]] [-**c**reate] [*options*]* [+**N**ewer] [+**O**lder]

tbackup [*drive*] **VE**rify [*disk_dir*[,*arch_dir*]] [*options*]*

tbackup [*drive*] **TE**st (** some QIC02 controllers only **)

save_spec: *disk_dir*[,*arch_dir*] **x**=*index_file filename*[,*arch_dir*]

options: +**p**ause -**v**erbose +**t**ension +**l**ist-only +**6**40k +|-**h**og +**o**ld
pf=[]*file_pattern* **pd**=[]*dir_pattern* **pp**=[]*path_pattern*
+**b**efore **d**=*dd-mm-yy* **t**=*hh:mm:ss* (Use digits)
+**F**orce (allow use of non-floppy disks)
g=*group* **m**=*member*
h=*number_64k_segments_to_hog*
e=*error_file*
"**v**=*volume_name*" (Use quotes if name contains spaces)

Description:

The TBACKUP command is used to archive files to one or more QIC tapes. This command will allow a single backup to span multiple tapes, so that files larger than the tape media may also be saved to tape. Each time a file is saved it is appended to the end of the archive which may span many tapes. Earlier versions of the same file will NOT be overwritten. You may restore any version of a file on the archive.

When performing a backup that spans tapes, TBACKUP will prompt for new tapes as required. The CRON utility should also be investigated, as CRON can cause tape backups to automatically occur overnight.

There are many options for this command and to properly use it, the *Floppy and Tape Backup* technical note at the back of this manual should be read.

See Also:

BACKUP CRON FBACKUP
Floppy and Tape Backup Technical Note

TCAP

TCAP - Manage terminal capability database

Syntax:

tcap append *tcap_file* [*name*] [**f**=*database*]
tcap copy *name new_name* [**f**=*database*]
tcap create [*num_nodes*] [**f**=*database*]
tcap delete *name* [*occurrence*] [**f**=*database*]
tcap define *name* [**f**=*database*]
tcap keys [**f**=*database*]
tcap list [**f**=*database*]
tcap query [**f**=*database*]

tcap set *name* [**f**=*database*]
OR
tset *name*

Options:

tcap_file	- TCAP database to append from.
f=*database*	- TCAP database to use (default: /config/tcap.dbase).
name	- Entry within the database.
new_name	- Name of a new entry.
num_nodes	- Maximum number of nodes in network (omit for non-networked QNX).
occurrence	- Which of several entries with the same name to use.

Examples:

tcap set qnx
tset qnxs
tcap define vt100
tcap copy qnxs vt52
tcap append old.tcap.dbase vt100

Description:

TCAP is a data base describing the capabilities of terminals connected to QNX. It is used by commands such as ED and MAIL to determine the escape sequences required to move the cursor, turn on attributes, recognize function keys etc...

The TCAP database file contains information about every terminal on every node in the network. A terminal type need only be SET once into the database. Once set, all full-screen applications should work properly on that terminal.

The data base is kept in a file called 'tcap.dbase' under the directory 'config'.

/config/tcap.dbase

This file is maintained by the TCAP command. A different database may be selected by using the **f=** option. This is often useful when testing a new terminal entry before committing it to the system TCAP database.

Three of the standard entries in the tcap.dbase file begin with the letters "QNX". They should be used in the following circumstances:

qnx - For the consoles. If you are on the console and your tcap entry is not set, it will default to **qnx**. This setting lets the term() functions use high speed video calls. The QNX terminal characteristics are fully supported.

Remote execution when invoked from the console, will behave like a **qnxt** setting even if the console is set to **qnx**. This is because high speed video calls only work on a local machine.

qnxs - This entry is used when a user is communicating to a QNX system using TALK. Since TALK does not send an FF character to precede function key codes, it does not fully support the QNX terminal characteristics. With this tcap setting, the term_key() library will insert an FF when the +efuncs option is enabled. In this way, programs like ED which use the FF character to distinguish between data and commands will be able to work. Any data which has the value of a function key will be taken as a command, due to the automatic insertion of the FF. The major disadvantage to using **qnxs** is that there is no means to enter foreign language characters.

This setting will throw away FF's if they come in because term_key() is generating them on its own based on the value of the data received, so if you have a QNX compatible terminal you should avoid using this mode.

qnxt - Used when communicating with a QNX compatible terminal to another QNX system through a serial line. This setting assumes that the terminal fully supports the QNX terminal characteristics (including colour). To take full advantage of QTALK and QTERM, this setting should be used. There are other terminals on the market which also support this tcap setting.

TCAP

Setting your tcap entry to **qnxs** or **qnxt** will never go through high speed video calls when using the term() functions.

APPEND

This command will append the terminal entries in a specified data file to the data file in '/config/tcap.dbase'. For example, assume that you have just received a tcap data file from a friend which contains several terminal definitions which you wish to use. The APPEND command may be used to append the new definitions to the end of your tcap data file. This might result in multiple definitions for the same terminal. You can remove a multiple definition using the DELETE command. If a *name* is specified, then only that entry will be copied over.

The APPEND command can also be used to change the number of node-id's supported. For example, to change the number of node ids to 12 you might enter the following sequence of commands.

```
chattr n=oldtcap.dbase /config/tcap.dbase
tcap create 12
tcap append /config/oldtcap.dbase
```

COPY

This command will make a copy of an existing terminal entry. This is useful when defining a new terminal which is substantially the same as an existing terminal. After the copy you would use the DEFINE command to make and changes to the new entry.

CREATE

This command will create the file '/config/tcap.dbase'. The *num_nodes* parameter is used by the networking version of QNX. It should be set to the *node-id* of the highest node in the network. A value of zero may be used for non-networked versions of QNX, or the field can be omitted entirely.

DEFINE

This command is used to define new terminal definitions and modify existing definitions. The terminal capabilities are entered into several full screen menus. Each menu is responsible for one particular aspect of a terminal definition. The editor documentation contains a section on terminals which may give further insight into its operation.

When creating a new terminal definition, menu 1 should be completed on another terminal which has an existing TCAP definition (like the console). After saving it away you may wish to complete the definition on the terminal being defined. This will allow you to type in the special keys on the terminal rather than having to enter a series of escape sequences. The TCAP program uses the direct cursor information in menu 1 to position the menu on the screen. It makes use of NO other terminal capability.

Upon entering each menu you will be prompted for an action. Enter the first letter of a command to select it.

CHANGE - Enter the current menu for changes
NEXT - Goto the next menu
PREV - Goto the previous menu
ABORT - Exit TCAP without saving any changes
SAVE - Exit TCAP and save any changes

The CHANGE command will move you cursor to the first menu field. When positioned at the start of a menu field you may enter a

carriage return - goto next field
space - goto previous field
the letter **E** - erase current field
the letter **R** - return to the menu
the character \ - take the next character as data

Any other character will be taken as data. The current field will be erased and the typed character entered as data into the field. Each successive character will be entered as data until

1. **The maximum field length is exceeded.**
2. **A carriage return which was not preceded by a backslash (\) is entered.**

The cursor will then be positioned at the next field. If the field prompt ends with a colon (:) then ASCII input is expected. If the field prompt ends with an equal sign (=) then numeric input is expected. All numeric input is assumed to be decimal.

The menu for box characters and input keys will be initialized to default values when you create a new terminal entry. These may be changed to more closely accommodate your terminal. For the input key escape sequences you should use particular care to avoid identical escape sequences for more than one key. For example, the default value for the Page Up key is a Ctrl-b. If your terminal generated a Ctrl-b as a leadin for its functions keys you will have to change the default for Page Up to something else. As another example, some terminals return

a TAB character for their right arrow key. In this case you will have to redefine the TAB key to be something like an ESC t or you will be unable to enter tabs.

DEFAULT KEY DEFINITIONS

Alternate - ESC a

Up Arrow - Ctrl u
Down Arrow - Ctrl j or Linefeed
Left Arrow - Ctrl h or Backspace
Right Arrow - Ctrl r

Home - ESC h
End - ESC e
Page Up - Ctrl a
Page Down - Ctrl b

Insert - Ctrl n
Delete - Ctrl k
Rubout - Delete or Rubout
Erase line - Ctrl x
Select - ESC CR
Cancel - ESC -
Help - ESC ?
Show - ESC s
Tab - Ctrl i or Tab
Tab to begin - ESC TAB b
Tab to end - ESC TAB e

Alternate - ESC a
F1 to F10 - ESC 1 to ESC 0
F11 to F20 - ESC ! to ESC)

NOTE: The following are the translations which are done for the function keys:

TCAP entry		Actual term_key() value
F1 to F10	=	F1 to F10
F11 to F20	=	CNTL_F1 to CNTL_F10
Alternate F1 to F10	=	SHIFT_F1 to SHIFT_F10
Alternate F11 to F20	=	ALT_F1 to ALT_F10

DELETE

This command will delete the terminal entry specified. The terminal *name* specified must be in the data base. If the occurrence option is specified, then the n'th occurrence of a terminal entry will be deleted. This option is useful when multiple occurrences of a single terminal exist within the tcap data file.

KEYS

This command will display the escape sequence for the special keys. These may be generated automatically by keys on the keyboard or entered manually as multi-character sequences.

LIST

This command will list the *names* of all terminals in the data base.

QUERY

This command will list the *name* of your active terminal entry. It should match the physical terminal which you are connected to.

SET

This command will change your active terminal entry to the terminal specified. The terminal *name* specified must be in the data base.

The TSET command is provided to allow non super-users to set terminal types. It is functionally equivalent to the TCAP SET command, but does not require the user to be a super-user.

See Also:

TSET

TIMER

TIMER - Implement timing facility in QNX

Syntax:

timer [no_account_entries] &

Options:

no_account_entries- maximum number of queued timer requests. Default is 10.

Description:

TIMER implements a timing facility in QNX. It is required by many utilities and 3rd party applications. Unless you are absolutely certain that you don't need it, you should be running TIMER. TIMER is usually started from the sys.init at boot time, and always left running.

If you ever do need to terminate TIMER, use the SLAY command:

slay timer

See Also:

CLRHOUSE
CRON
LOCKER
QCP
SLEEP
SLICE
SPOOLDEV

TSET - Set terminal type

Syntax:

tset [terminal_type]

Description:

TSET allows the user to define query or define which type of terminal is in use for the current $tty device. For example, if a user was logged into a QNX machine from a vt100 terminal, the command:

tset vt100

would allow QNX to recognize the terminal type so that QNX full screen applications would be able to operate correctly. This example assumes that a vt100 terminal type is defined in the file */config/tcap.dbase.*

Typing the command:

tset

would cause the TSET command to display what the currently defined terminal type is.

Note that the TCAP command could also perform this operation, but the user of the TCAP command must usually be a super-user because the TCAP command also allows the contents of the */config/tcap.dbase* file to be modified. The TSET command is usable by a non-super user because it does not modify the */config/tcap.dbase* file.

See Also:

TCAP

TSK - Display task information

Syntax:

tsk [**f**={**cmoprst**}] [**t**=*tty*] [**u**=*userid*]
tsk **co**de [**p**=*program*]
tsk **in**fo
tsk **me**m *tid* ...
tsk **na**mes
tsk **si**ze [**t**=*tty*] [**u**=*userid*] [**p**=*program*]
tsk **po**rts
tsk **tr**ee [+tid]
tsk **us**ers [**t**=*tty*] [**u**=*userid*] [**p**=*program*]
tsk **vc**s
tsk **wh**o *tid* ...

options: +**q**nx -**h**eader +**p**hysical **n**=*node* **s**=*sortfield*

Options:

-**h**eader - Don't include a header.
+**p**hysical - Display memory as 24 bit addresses. Selectors in protected mode are mapped to real physical memory addresses.
+**q**nx - Use the QNX line drawing set when the **tree** option is used. This is the default on the console but must be requested on a terminal.
f={cmoprst} - Select a set of fields of information to be displayed. Each letter included represents a field.
n=*node* - Obtain information from the indicated node rather than the local node.
s=*sortfield* - Sort the lines based upon information in the indicated field. Each field is a column separated by blanks. The fields are numbered as s=1 (first), s=2 (second), ...
t=*tty* - Only display tasks associated with the indicated tty.
u=*userid* - Only display tasks associated with the indicated userid.
p=*program* - Only display information associated with the *program* task.
tid - Display information on the indicated task.

Examples:

tsk	- Display a brief summary of all the tasks in the system.
tsk t=3	- Display a listing of all tasks associated with tty 3.
tsk u=bill	- Display a listing of all tasks associated with userid bill.
tsk mem	- Display the free memory in the system.
tsk mem 30c	- Display any extra segments used by task (Tid) 30c.
tsk info	- Display system configuration information.
tsk tree n=7	- Display task tree on node 7.
tsk vcs	- Display virtual circuits to other nodes.

Description:

The TSK command allows the user to obtain a "snapshot" of the tasks which are currently in existence. If present, the keyword after the command allows you to select different types of information and different views of the information displayed. You need only type the first two characters of a keyword.

There are several options common to all forms of the command. The **n=***node* option obtains information on another node in the network. The default is to obtain information on the current node. The **s=***sortfield* option allows you to sort the displayed lines of information based upon a field. You could sort by tty, task id, state, code size ... The **+p**hysical option maps 286/386 protected mode selectors into real 24 bit memory addresses.

TSK

When invoked without a keyword TSK will list the name of all tasks which are currently active. The task-id (tid), state, priority, user-number, and associated tty are displayed for each task as well as the father-son relationship of the tasks. Priorities range from 1 to 15 with 1 being the highest. The task flags field contains letters to indicate the following:

A - Administrator.
C - Concurrent execution.
D - Doomed.
E - Escaped shell.
H- To be Held.
L - Locked in memory.
N - New task entry in the process of being created.
P - Privileged.

The **f=**{cmoprst} allows you to select which type of information you wish displayed. Each letter specified includes a field of information as follows.

c - Number of programs sharing this code segment and its Code flags.
m- Memory (code/data) used.
o - Ownership (group/member).
p - Program name and task id.
r - Relationships (father/brother/son).
s - State information (state/blocked_on/priority/flags).
t - Tty.
u - Userid.

eg: tsk f=pu (display program names and userids)

You may wish to create a shell command to invoke TSK with your favorite options.

TSK CODE

Display information on each code segment running in the system. The programs name and the number of links (programs sharing the code segment) to the program are displayed. The flags field contains letters to indicate the following:

F - Uses Floating point (8087).
H- Loaded in high memory.
L - Locked in memory.
O- Change Owner.
P - Privileged.
R - Remote creates ok.
S - Shareable.

TSK INFO

Displays information on QNX itself. This includes the version number and the date the operating system was created. Maximums for a number of items are displayed including number of tasks and files. The flags field contains letters to indicate the following:

D - Memory below 1 Meg reserved for DOS.
F - 8087 installed (float).
L - Large file system supported.
M- Memory manager task running.
P - Password protection enabled.
R - Remote creates allowed for non super-users.
S - Automatic sharing of code segments enabled.

TSK MEM

Display information on extra segment memory usage. If no arguments are

provided then the system memory free list is displayed. If you provide one or more task id's then the extra memory segments allocated by those tasks is displayed.

TSK NAMES

Display registered names in the system. Both names registered on the global name server and names registered on the local machine are listed.

TSK SIZE

Display information similar to that shown when TSK is invoked without a keyword, only replace the father/brother/son information with the tasks sizes.

TSK PORTS

Display which ports are in use and the tasks which have attached to them.

TSK TREE

Display a graph showing the relationship of tasks running in the system. The tty is displayed at the end of each line. The **+t**id option will include the task id with the name of each task in the displayed graph. You may select a subset of tasks to match using the **t=***tty* or **u=***userid* options. On the console the line drawing characters are used while on a terminal +'s, |'s and -'s are used. If your terminal supports the console line drawing character set you can force it to be used with the +qnx option.

TSK USERS

TSK will list the name of all tasks which are currently active. The task-id (tid), userid, state, priority, user-number, and associated tty are displayed for each task.

TSK VCS

Display virtual circuits existing between this node and other nodes in the network. The display shows each local task id and program which has a virtual circuit to another node and task id. The virtual id's and the size of the virtual circuit is also displayed. You can obtain more information on the task at the far end of the virtual circuit using the WHO subcommand and providing the local vid as an argument.

TSK WHO

Display the tty, program name and userid running the indicated task. If you type in a virtual id (vid) then information on the remote real task is displayed.

TYPE

TYPE - Type arguments

Syntax:

type
type *text*

• *This is a local shell command* •

Description:

The arguments following the command are displayed on the terminal, followed by a carriage return. This is often used within command files to display progress messages.

See Also:

ECHO

TZSET - Display or set timezone offset

Syntax:

tzset [[+-]*hour*[:*min*]]

Options:

hour - 0-12
min - 0-59

Examples:

tzset
tzset +5
tzset -3:30

Description:

If no arguments are given, the TZSET command allows the user to display the timezone offset on their terminal. The timezone offset is specified as a number of hours and minutes, plus or minus, from Greenwich Mean Time (GMT). This offset is used to adjust the internal time, which is maintained in GMT, to local time for display purposes.

When changing the timezone, the hour is entered first (optionally preceeded by a plus or minus; plus is assumed). The hour may be followed by a colon (:) and a number of minutes (which must be a multiple of 15) for the few places in the world which are in a timezone that is not on the hour (Newfoundland, Canada for example).

If you are in a location which periodically adjusts its clocks (for example, Daylight Savings Time), simply adjust the timezone offset given to TZSET to maintain the correct local time.

See Also:

CLOCK DATE RTC

UNPACK - Unpack a packed file

Syntax:

unpack *file** [*options*]*
OR
unpack *<infile >outfile* [*options*]*

Options:

+display - Force output to the screen.
-remove - Suppress the removal of input files.
+verbose - Display status messages.
-time - Retain original file time. Set to current time by default.

Examples:

unpack *.z - Unpack all packed files in current directory
unpack <packed_text >text

Description:

UNPACK expands packed files into exact duplicates of the original. UNPACK operates on files which have been created with the PACK utility.

When the first form of UNPACK is used each file will be replaced by a file with the same name and the **.z** removed. The packed file will by default be removed. You can suppress the removal by specifying the **-r**emove option.

When the second form of UNPACK is used it will read from the standard input and write to the standard output. In this case the input file will never be removed and you have control over the name of the output file. The **+d**isplay will force all output to the screen. The input packed files will not be removed. If more than one file is specified they will be appended together. If you were on the console the following two command would be the same:

unpack file >$con
unpack file +display

UNPACK

This option is typically used to quickly examine the contents of a packed file.

If the +verbose option is specified each file will be printed as it is unpacked.

See Also:

PACK

WC

WC - Word count

Syntax:

wc [*file*]* [**-chars**] [**-lines**] [**-words**] [**-verbose**]

Options:

file	-	Name of a file to count. If omitted the standard input is used.
-chars	-	Don't display the number of characters.
-lines	-	Don't display the number of lines.
-words	-	Don't display the number of words.
-verbose	-	Don't display the file name and omit descriptive text.

Examples:

wc test.c
wc *.c -w
wc inventory mailing_list -c -v

Description:

WC will count the total number of characters, lines and words in each of the given files. The number of lines and words is only meaningful for text files. A word is a maximal string of characters delimited by spaces, tabs or newlines.

If more than one filename is given, then WC will also report the total number of characters, lines and words in all the files.

See Also:

SIZE

WHO - Who is logged-in to the system

Syntax:

who [**am i** | **net** | *node**] [*options*]*

Options:

am i - Display who you are.
net - Display all users in the network.
node - Display all users on this node.
-header - Suppress column header.
+system - Display system owned tasks.
-verbose - Display only owner, tty and task name.

Examples:

who
who am i
who net +s
who 3
who 3 5 7

Description:

The WHO command will display the users which are logged on to the system. The output will be of the form.

Node	User	TTY	Idle-Time	Sign-On-Time	Flags	Command
4	opr	0	0:00:49	Nov-23 5:21 pm	----	sh
4	dtdodge	4	0:02:45	Nov-23 5:06 pm	----	sh

If invoked with no arguments, WHO will display all users logged-in to your machine. The **am i** option is useful to find out who is logged on to an unoccupied machine or a second window on the console. The **net** option will list all users logged on to all machines in the network.

The **NODE** field will only be displayed when running the network version of QNX.

A field of dashes (--- --) for the time indicates that someone forgot to set the date in the system.

The idle time field displays how long it has been since a character was received by that particular tty. This field is particularly useful for the detection of users who have been idle long enough that they should be logged off.

The flags field indicates that a message is waiting to be printed on that users console.

A - An appointment was changed
B - An APB (all points bulletin)
C - Chat request
M - A MAIL WAITING message

See Also:

LOGIN
NET

WS - Walk directory Structure executing a command

Syntax:

ws [*directory* ["*command ...*"]] [*options*]*

Options:

dc=*delete_char* - This character when present in *command* will delete the previous character of the expanded string. (default is a backquote)

pc=*path_char* - This character when present in *command* will be replaced by the current pathname. If the *path_char* is followed by a question mark, then the two character sequence will be replaced with input prompted for from the standard input. If the *path_char* is followed by another *path_char* then only the filename and not a complete pathname will be replaced. This is sometimes useful when +replace is NOT specified but you still want the relative name. (default is an at-sign @)

d=*date* - Match only those files which have changed since this *date.* Date is of the form dd-mm-yy. (default is 01-Jan-80)

t=*time* - Match only those files which have changed since this *time* on the given date. Time is in the 24 hour format hh:mm:ss. (default is 0:0:0)

l=*levels* - Specifies how many *levels* down from the indicated directory will be searched. The default is l=1 and will cause only files at the current level to be searched, not files under subdirectories.

g=*group* - Indicates that only files whose group number matches *group* will be selected.

m=*member* - Indicates that only files whose group member number matches *member* will be selected.

p=[^]*pattern* - Indicates that only files whose name match this *pattern* will be selected. Up to ten **p**= options are allowed, in which case files which match ANY of these *patterns* will be copied. Also, if the pattern is preceded by an up- arrow (^), then files which match this *pattern* will NOT be selected.

+abort - If a command returns a non-zero status then WS will abort.

+before - Changes the meaning of the time and date options to mean

before. The default is now or after.

+directory - Selects directories rather than files.

+error - If a command returns a non-zero status, a prompt to continue will be displayed. (default is to ignore returned status)

+locked - Match only files which are BUSY.

-locked - Match only files which are not BUSY.

+modified - Match only files which have been modified and not backed up using the BACKUP command.

-modified - Match only files which have been not been modified.

+query - Before executing each command a prompt will be issued requesting permission to execute. If option verbose is set (default) then the entire command line will be prompted, otherwise only the selected pathname is printed. A single 'y' will execute while anything else will skip to the next pathname.

+relative - The WS command will usually generate complete pathnames from the indicated directory. This option will cause WS to do a CD command when it moves to each new directory and the pathname will be the simple filename at this directory.

-verbose - Suppress the printing of each command before it is executed.

Examples:

ws "frel @" - Remove all files at current directory.

ws /user/bill "frel @" l=20
- Remove all files owned by bill.

ws /user/bill "drel @" l=20 +d
- Remove all directories owned by bill.

ws "chattr n=@? @" p=*data*
- Change the name of all files containing the string "data" to a new name which will be prompted for.

ws "ed @" p=*.c d=3/1/83
- Edit all C programs which were changed on Jan 3, 1983.

ws 2:/ "type @" -v p=xyz l=12
- Search all of disk 2 looking for the file xyz. If found, print its complete pathname.

Description:

WS is a utility which increases the power and scope of existing QNX commands. It is a very powerful command and should be carefully studied. You may rarely need it, but when you do, it will be of untold value!

The simplest form of the command is:

ws "*command arguments*"

which will execute the command in double quotes on each file in the current directory. The double quotes are necessary. Any occurrence of the AT-SIGN character (@) will be replaced by the current filename. For example:

ws "type @" - Type name of each file in current directory
ws "frel @" - Release each file in current directory

A number of options may follow the command. The two most common being **+query** and **-verbose**. The query option will prompt you for each possible execution of the command. A single "y" will execute the command, anything else will skip execution and proceed to the next. The **-verbose** option suppresses the printing of each command before execution.

WS will not normally descend to subdirectories in the current directory. This can be overridden by specifying the number of directories to descend using the **l=***levels* option. The following example will release all files under the directory "/user/bill" including all subdirectories. The directories are then removed. These two commands have effectively released all a users' files, perhaps in preparation of removing his user id.

ws /user/joe "frel @" l=100 - Release all files
ws /user/joe "drel @" +d l=100 - Release directories

Note that in this case a particular directory was specified and in the second example the **+directory** option matched directories, NOT files.

The time and date options allow the selection of files based upon their last change. The default will typically select all files. The following command would recompile all C source files which were changed today (read up on the ***MAKE*** command for a better method).

ws "cc -c @" p=*.c d= l=5

Note that a null string for the date defaults to today.

The **p=**[^]*pattern* option follows the same syntax as the ***BACKUP*** command and it should be referred to for wildcard character definitions.

The AT-SIGN prompt (@?) is provided to increase the flexibility of the WS command. There is no limit (within reason) to the number of **@?** prompts you may have on a command line. For example

ws "type 1-@?-2-@?-3-@?-4-@?-5-@?-"

will prompt for each of the five arguments. Note that the @ does not need to be delimited by spaces. This is a silly example in that it will select each file in the current directory but does not expand it on the command line. The more realistic example below allows the super user to interactively set the ownership on each user directory

ws /user "chattr @ o=@?" +q

The **+relative** option will cause WS to issue a CD command before entering each subdirectory. The pathnames expanded by the AT-sign (@) will contain the last component of the full pathname. This can best be understood with an example.

ws 1:/ "type @" l=10 - **Type complete pathnames.**
and
ws 1:/ "type @" +r l=10 - **Type simple file names.**

All examples above have contained a *command* and some have contained an overriding *directory* as their first argument.
If neither is specified then WS will operate on the current directory and prompt for command lines from the standard input. Each input line will be executed on all selected files and the user will then be prompted for another command. An end-of-file (**Ctrl-D**) will terminate the command.

If a *directory* is specified then the syntax requires you to enter a *command*. If the command is an empty string then you will be prompted as above.

ws - **Prompt for commands and operate on current directory.**
ws /user/bill "" - **Prompt and operate on /user/bill**

WS could be used as a filter in which it was piped a series of commands. The following simple example assumes that the file **my_cmds** contains a list of commands you wish executed on all files ending in ".c" at the current directory.

ws p=*.c <my_cmds

See Also:

BACKUP EO MAKE

XLAT - Translate Characters

Syntax:

xlat *xfile* [+raw] [+wildcard] [<*input_file*] [>*output_file*]

Options:

+raw - Turn off output character mappings (ERS, EDEL, ETAB). This only affects output to a device.
+wildcard - Allows the use of the '?' charcater to match any single char. This is useful for removing escape sequences from log files.

Examples:

xlat qume.xlat </expl/inform >$lpt

Description:

XLAT will translate single characters or multi-character sequences found in *input_file* based on the translations specified in **xfile.**

The translation file specified by *xfile* consists of one or more translate entries with optional comments. A translate entry has the form:

input_seq output_seq [alt_output_seq]

where each translate entry is on a separate line. *input_seq, output_seq* and *alt_output_seq* have the form:

char [,char]*

char may be specified as an actual character, two hex digits, or a carat (^) followed by an actual character. The last case will result in a control character if (and only if) the actual character is alphabetic (a-z or A-Z), else the actual character will be used (this can be useful for entering a comma (,) or carat (^), otherwise the hex value could be given). A comma (,) is used to separate characters and white space (spaces or tabs) is used to separate sequences.

If the *alt_output_seq* is given (it normally is not) then *input_seq* will toggle between the two possible output sequences.

The following xlat table will convert all record separators, tabs and runs of two spaces to a single space. Because only one pass is made through the translation table, this will not reduce all "white space" to a single space.

```
1e      20
09      20
20,20   20
```

This table will translate some control characters into their ASCII name with alternates for tab and rubout.

```
00  <,N,U,L,>
04  <,E,O,T,>
07  <,B,E,L,>
08  <,B,S,>
09  <,H,T,>     <,T,A,B,>
0a  <,L,F,>
0c  <,F,F,>
0d  <,C,R,>
11  <,D,C,1,>
13  <,D,C,3,>
1b  <,E,S,C,>
1e  <,R,S,>
7f  <,D,E,L>  <,R,U,B,>
```

You may wish to create a number of table to convert

```
upper case  -->  lower case
ascii       -->  ebcdic
QNX escape  -->  daisy wheel printer
sequences        escape sequences
```

ZAP - Zap damaged files out of existence

Syntax:

zap *file*

Examples:

zap junk
zap /user/bill/junk

Description:

ONLY USE THIS COMMAND TO RELEASE DAMAGED FILES.

ZAP should be used when a file is known to contain bad disk blocks, or if a file has been left in an inconsistent state (FILE BUSY) due to a system crash while a file was being written.

ZAP releases files by clearing the directory entry for that file. The disk blocks used by that file are NOT reclaimed. Repeated use of ZAP will therefore reduce the total number of disk blocks available to the user. These can be reclaimed by running the CHKFSYS utility when the system is idle.

Normally, a user should use the FREL, DREL, RM or RMDIR commands to release files or directories.

See Also:

CHATTR	DREL	RMDIR
CHKFSYS	FREL	RM