QNX®

Technical Notes

Technical Notes

1. Operating System Limits

This section describes limits of the QNX operating system at the time of printing. Use the TSK command to verify what your version supports. For more information call our Technical Support line.

1.1 File System Limits

These are the same for PCAT and ATP versions

Max number of extents per file:	65,535
Max bytes per extent:	2,147,483,647
Max files per directory:	32,767
Max blocks per disk:	2,147,483,647
Max disk volume size (bytes):	1,099,511,627,776
Max file size (bytes):	1,099,511,627,776
Max number open files:	4 to 1000 (configurable). Default ->64.
Max number QNX drives:	8
Max number mounted drivers:	8
Max number adopted drives:	16
Max drives/nodes in search:	8
Max characters in filename:	16
Max characters in a pathname:	256

1.2 Device Limits

	PCAT	ATP
PHYSICAL TTY's (console):	1 *	1 *
(serial ports):	10	16
(parallel ports):	2	2
Additional ADOPTABLE TTY's:	20	40

* Unused serial/parallel port device entries can be used to mount up to 7 extra consoles.

1.3 Other Limits

	PCAT	ATP
Tasks (combined virtual and local):	64	150
Ports:	28	40
Accounting Entries:	50	100
Registered Names (local):	30	50
Extra Segments (configurable):	160 to 200.	160 to 800
Memory:	640K	640K Base + up to 15Meg Extended

NOTE: Only one RAMDISK can be mounted.

2. Floppy and Tape Backup for QNX

Please read this entire document before deciding on

an archive philosophy.

All examples using FBACKUP also apply to TBACKUP

2.1 Introduction

When saving large amounts of data to floppy diskette the QNX BACKUP command poses several problems. It is relatively slow and no single file may be greater than the size of the floppy. The solution to this is the FBACKUP and TBACKUP commands which are very fast and allow files to span media boundaries.

FBACKUP	• **Archive to floppy diskette or Bernoullie removable hard disks.**
TBACKUP	• **Archive to 1/4 inch tape cartridge using Everex, Tecmar and Wangtek QIC60 internal/external tape units.**

The set of floppy diskettes or tapes produced is called an archive and may only be manipulated by the FBACKUP or TBACKUP commands. In other words, you may not use LS, COPY, ... on archive diskettes and tapes.

There are seven functions provided by the FBACKUP and TBACKUP commands.

config	• **Configure the tape hardware. Used by TBACKUP only.**
init	• **Initialize a floppy/tape for archiving.**
files	• **List files in an archive.**
name	• **Print the volume name of the archive.**
save	• **Disk to archive save.**
restore	• **Archive to disk restore.**
verify	• **Disk and archive compare.**

Maintaining an archive system involves several phases, as follows.

Phase 1

Before you can use a diskette/tape for archive backup it must be initialized. This is done using the the INIT function to the FBACKUP or TBACKUP command. Note that only the first diskette/tape of an archive should be initialized.

Phase 2
Files can now be saved on the floppy/tape using the SAVE function. This function supports a rich series of options similar to the QNX command called BACKUP.

Phase 3
After a SAVE you may wish to run a verify operation on the tape. This is optional. If performed, it should be run on a new tape.

Phase 4
In the event of a catastrophe, you may have to restore files saved in the archive back to a disk using the RESTORE function. This may be needed in the case of a hard disk malfunction, or in the case of a "*boo-boo*", such as accidentally releasing a file.

2.2 Archive Structure

The first block of each archive diskette/tape contains the name of the archive and the sequence number of this diskette/tape.

```
|volume name   | directory | file 1 | file 2 |     |volume name | continued data
Diskette/tape 1  floppy only                        Diskette/tape 2
```

In the case of FBACKUP the first floppy also contains a fixed length directory which is created when the archive is initialized. The data is stored after this directory. The floppy data portion of the diskettes are formatted without any sector interleave. FBACKUP uses multi-sector I/O to maximize the data rate to and from the diskettes.

2.3 Backup Philosophy

Using a file-by-file system, one backup technique available is called incremental backup. In this technique, a backup of all files is performed periodically, and a backup of the modified files is done more frequently. For example, a complete backup of all files may be performed Monday evening at closing time. Then, on Tuesday through Friday, only the modified files are saved into separate directories on the archive. In the next week, the process is repeated with a DIFFERENT set of archive diskettes. Depending on preferences, there may be four or five archives being cycled in this fashion. If a complete restore is needed, restore from the last complete backup, then restore from each of the partial backups taken since then. If only one file is needed, look in the most recent partial backups first, and if it is not found there, restore the file from the most recent complete backup.

Some computer centers also like to maintain archival copies of their files. In this case, every fourth incremental archive, for example, could be filed away instead of being cycled. Of course, it would have to be replaced with a new archive to enter the cycle of incremental archives. In this way, a record of all files would be kept for each month, approximately.

Although incremental backups are faster and smaller, they are more complex to restore from. You have to determine the day on which the file was last saved. This will be Monday and perhaps one or more weekdays. A full backup every day eliminates searching for files on the archive, at the expense of using diskettes/tapes and time.

Lastly, when backing up a large database, a single change to a record entails saving away the complete file, which may be many megabytes. In this case, an incremental backup may not save any time since the large database files may change (if even a little bit) every day.

2.4 Detailed Command Descriptions

Each of the five functions provided by the FBACKUP and TBACKUP commands will be described separately. Typing

fbackup ?
or
tbackup ?

will give you a usage message should you forget one of the option letters. Note that only the first two letters of a FBACKUP/TBACKUP function need be entered.

2.4.1 CONFIG - Configure the Tape Hardware

Syntax:

tbackup config **c=***dma_channel* **i=***io_port* **u=**a|e|i

where unit is:	u=a	• Archive Scorpion (internal or external).
	u=e	• Everex/Tecmar/Wangtek external.
	u=i	• Everex/Tecmar/Wangtek internal.

Description:

To use the TBACKUP command you must first configure the TBACKUP command to your hardware. This is done using the CONFIG function. This will create a file called **/config/tape.*n*** where *n* will be replaced by your node number. This will be zero in a non-networked system. For example, to configure an external tape unit which uses ioport 338 and dma channel 3 you would type.

```
tbackup config c=3 i=338 u=e            typical AT setup
```

On a PC with a hard disk, dma channel 3 is already in use. You should set your tape controller card to use use DMA channel 1.

```
tbackup config c=1 i=338 u=e            typical PC setup
```

2.4.2 INIT - Initalize NEW Archive

Syntax:

fbackup [*drive*] **init** *max-num-files* ["**v**=*volume-name*"] [**c**=*capacity*] [-format] [**f**=*format_cmd*]

tbackup **init** ["v=*volume-name*"]

Description:

The INit option will reformat the diskette/tape. In the case of a floppy archive (only) you must reserve room for a directory which can contain up to the number of files indicated. The directory may not span a floppy diskette and is limited to the following sizes.

Floppy Size	**Max Number of Files**
360K	**2876 (*c=360k*)**
720K	**5756 (*c=720k*)**
1.2Meg	**9596 (*c=1.2m*)**

The size of the archive may be selected to be one of three standards using the options provided. The 720K size requires an 80 track disk drive while the 1.2Meg size requires a high capacity disk drive on an AT. You may of course create an archive with a much smaller number of files in the directory. This will leave more room for data on the first diskette.

The **-format** option will suppress the message that this will destroy an existing archive. It will also suppress the reformatting of the diskette. The **f**=*format_cmd* lets you select an alternative command to format the media. This is needed to format an IOMEGA cartridge.

In the case of a tape archive you do not need to specify the directory size, tape size or format options.

You may also give the archive a volume name. The name will be displayed by the FILES option. The name is optional, but is recommended so you can tell one archive from another. The SAVE and RESTORE functions have an option (v=*volume_name*) which will verify that the volume name of the archive is what you are expecting.

You must run INIT on the first floppy/tape before you can use the archive. You can SAVE files any number of times to an existing archive. Until you re-create it, the new data is simply appended to the end of any existing data.

2.4.3 FILES - List Saved Files In the Archive

Syntax:

fbackup [*drive*] files [***directory***] [options]*

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

Options:

directory	- Show only files in this directory.
pf=[^]*pattern*	- Show only files whose names match this pattern.
pp=[^]*pattern*	- Show only files whose full path match this pattern.
-verbose	- Show only the file names.
+verbose	- Show complete information.
+summary	- Show only a summary of the directory.

Description:

The FILES option prints a list of the names of the files that have been saved in the archive. In fact, the FILES command is almost identical to the QNX FILES command.

The ***directory*** option allows you to look at just those files contained within a particular directory or subdirectory. If you do not specify, the FILES command will show files starting at the root directory.

The **pf**=*pattern* option allows you to specify part or all of the name(s) you want to see. For example, to see information about the file called "**test.c**", you could specify the option as "**pf=test.c**". Only files with a name matching that pattern, in this case "test.c", will be listed. To get information about all files whose names end with "**.c**", you could specify "**pf=*.c**". See the QNX FILES command for more information on patterns. If you do not specify a pattern, any file name will be displayed. You may specify more than one pattern including patterns which do not match a file. For example:

fbackup files pf=^*.o • **match all files that do not end in .o**

You may abbreviate **pf**= to a simple **p**=.

The **pp**=*pattern* option operates the same as the **pf**= option except the entire pathname is matched against the pattern, not just the trailing filename. For example.

fbackup files pp=*/dir/* • **match all files under a directory.**

The +verbose and -verbose options determine how much information is shown for each file. If neither is specified, the FILES option will display each file's name, date, time, and size. If the -verbose option is used, only the file's name is shown. If the +verbose option is used, each line of output will show the file's name, date, time, size, group and owner numbers, attributes, permissions, and the location of the file in the archive.

If the +summary option is used, no file names will be listed. Only statistics on the number of files and size of the archive will be displayed.

2.4.4 SAVE - Save Files to the Archive

Syntax:

fbackup [*drive*] **save** *save_spec* ... [*options*]*

tbackup **save** *save_spec* ... [*options*]*

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

Options:

disk-dir	- Directory to save files from disk.
,arch-dir	- Directory to save files to arch.
+**all**	- Save all files, modified or not.
-clear	- Do not clear the file's modified bit.
+list-only	- Show what will be done, but don't do it.
+**pause**	- Prompt for diskette/tape before starting.
-verbose	- Do not print filenames while operating.
pf=*pattern*	- Save only files whose names match this pattern.
pd=*pattern*	- Save only directories whose names match this pattern.
pp=*pattern*	- Save only files whose full path match this pattern.
g=*group*	- Save only files belonging to this group.
m=*member*	- Save only files belonging to this member.
d=*date*	- Save only files created/modified since this date.
t=*time*	- Save only files created/modified since this time.
+**bef**ore	- Change sense of d= and t=.
l=*levels*	- Only descend this far in subdirectories.
e=*err_file*	- Name of files to place errors in.

Tape only options:

-**hog**	• Never hog more than 256K of memory.
+**hog**	• Hog has much memory as possible.
+tension	• Tension the tape before saving. Use if tape has been banged, subjected to cold/heat or unused for some time.

Description:

The SAVE function is the one which does the real work. It copies files from a QNX disk to the floppy/tape archive. If you perform a save with the intention of repartitioning or reformatting your hard disk please follow the SAVE function with the VERIFY function before you release the file(s). Play it safe.

You may specify the files to be saved in three ways. You may mix these types on the same save command.

1. A directory name in which case all files under the directory will be saved. You may restrict the files saved using the other options. For example, **pf=** and **g=**.

   ```
   fbackup save 3:/ pf=*.[ch]
   fbackup save /projects/alpha g=12
   ```

2. An index file in which each line of the file contains the pathname of a file to be saved. The names files will always be saved regardless of any selection options set.

   ```
   fbackup save x=index_file
   ```

3. A simple filename. The file will always be saved regardless of any selection options set.

   ```
   fbackup save /data/dbase.dat /data/dbase.idx
   ```

In all cases, the full pathname from the root of the file is saved in the archive. The node and drive number are not saved. The ***,arch-dir*** option right after the source disk directory name (no space), allows you to prepend additional directory information to the file name which is saved to the archive. For example, if you simply type

```
fbackup save /user/george
```

all of george's files will be saved under **/user/george** in the archive. If, however, you type

```
fbackup save /user/george,/wednesday
```

then george's files will be saved under **/wednesday/user/george**. You can use this to prepend a drive and node number as well so that

```
fbackup save 3:/,3:/
fbackup save [4]3:/,[4]3:/
```

will save your files under **[4]3:/**, not just /. This should be used if your disk supports more than one QNX hard disk partition or you will NOT be able to differentiate which partition a file came from.

The **-c**lear option pertains to the modified bit of a file. This is a bit contained in the directory entry of a file on a QNX disk that indicates whether or not the file has been modified. Any time the file is written to, the bit is set. Usually the SAVE function will clear the bit after saving the file to the archive. Using the **-c**lear option indicates that the SAVE option is to leave the bit alone.

The +**a**ll option tells SAVE to save all files, even if they have not been modified. If this is not specified, only files that have been changed since the last backup will be saved.

The +**p**ause option tells the SAVE option to prompt for a diskette/tape before starting. If it is not specified, the command will start immediately.

The **-v**erbose option inhibits the command's display of information messages while it is operating. These messages indicate the name of the file currently being saved. The program will, however, still print messages about any errors it encounters while operating.

The +**l**ist-only option can be used to verify that things will happen the way you want them to. The file names will be processed, as usual. However, no data will be transferred. The file names will be listed on the terminal.

The **pf**=*pattern* option allows the user to specify which files are to be saved. Only files whose names match the specified pattern will be saved to the archive. For example, to save only files whose names end with the characters "**.c**", you could specify the option "**pf=*.c**". You may specify more than one **pf**= option. You can also specify a pattern which excludes files. For example.

```
fbackup save 3:/ pf=*.c pf=*.h        • Save only C and header files
fbackup save 3:/ pf=^*.o              • Don't save object files
```

For more information, see the QNX BACKUP command. If this option is not specified, the file's name is not considered while it is being saved (unless pp= is specified).

The **pd**=*pattern* option allows the user to specify which directories are to be saved. It is identical to the **pf**= option above, except that it operates only on directory names.

The **pp**=*pattern* option allows the user to specify which full pathnames are to be saved. It is identical to the **pf**= option above, except that it operates on the full pathname, not just the filename (or directory name in the case of **pd**=). For example.

fbackup files pp=*/dir/* • **match all files under a directory.**

The **g=***group* and **m=***member* options refer to the group number and member number assigned to each user of a QNX system. When you login with your name, your group and member numbers are read from the system password file. Thereafter, any files that you create contain your numbers. By specifying these options, SAVE will save only files created by a particular user. For example, if user george is assigned group number 2, member number 5, you could save only george's files by specifying "**g=2 m=5**". You can save files created by any member of group 2 by specifying just "**g=2**", or you can save files created by user 5 of any group by specifying "**m=5**" (although this would seem to be of limited usefulness). If neither of these options is specified, the group and member numbers are not considered when files are being saved.

QNX updates the date and time of each disk file any time the file is changed. The **d=***date* and **t=***time* options allow you to specify which files to save based on the last time they were modified. If you specify the date or time, only files changed since that time will be saved. For example, to save any files changed or created since 8 a.m. on October 28, 1985, you would specify

d=28-10-85 t=8:00

The date is specified as *yy-mm-dd*, and the time is specified as *hh:mm:ss*, in 24-hour format. If not specified, the date and time are not considered when files are being saved. Specifying a particular date and time is usually used because the modified bit cannot be used. This option is therefore typically used with the **+all** option.

The +**be**fore option changes the sense of the **d=***date* and **t=***time* options. Instead of saving files changed SINCE the indicated date, it will save files changed BEFORE the given date.

The **l=***levels* option allows you to control how far the SAVE option descends into subdirectories looking for files. For example, if you specify "**l=1**", only files in the top directory will be saved. If you specify "**l=2**", only files in the top directory and in the next level of subdirectories will be saved. If not specified, SAVE will descend to all subdirectory levels.

Here are some general notes about the SAVE option.

When SAVE locates a file that is to be saved, it determines the complete pathname of that file, even though you may not have given the complete path when you typed the command. For example, if your current directory is **/user/george**, and you have a subdirectory called **stuff**, you may save the files in **stuff** by typing

```
fbackup save stuff
```

However, the files will appear on the archive as **/user/george/stuff/file1**, **/user/george/stuff/file2**, and so on. If you specify an "***,arch-dir***" on the archive, the complete path will be appended to the destination archive directory name. For example, if you type

```
fbackup save stuff,/foo
```

the files will be saved as **/foo/user/george/stuff/file1**, **/foo/user/george/stuff/file2**, and so on.

2.4.5 RESTORE - Restore Files from an Archive

Syntax:

fbackup [*drive*] restore ***disk-dir***[***,arch-dir***] [*options*]*

tbackup restore ***disk-dir***[***,arch-dir***] [*options*]*

Options:

disk-dir	- Restore files to this directory.
,arch-dir	- Restore files from this archive directory.
+pause	- Prompt for diskette before starting.
-verbose	- Do not print files while operating.
-create	- Do not create files or directories on the disk.
pf=*pattern*	- Restore only files whose names match this pattern.
pp=*pattern*	- Restore only files whose full path match this pattern.
g=*group*	- Restore only files belonging to this group.
m=*member*	- Restore only files belonging to this member.
d=*date*	- Restore only files created/modified since this date.
t=*time*	- Restore only files created/modified since this time.
+before	- Change sense of d= and t=.
+list-only	- Show what will be done, but don't do it.

Description:

The RESTORE option is the inverse of the SAVE option. With a little luck, you will never need to use it. The RESTORE option copies files from the archive to a QNX disk. This is presumably used to restore files which have been corrupted or accidentally released. You should be sure to correct any problems that caused the corrupted file (such as bad disk blocks) before restoring the file.

The RESTORE option has one mandatory parameter. You must specify the destination directory where the files are to be written. For example, if you want to restore all the files on the archive to drive 3, you would type

```
fbackup restore 3:/
```

You can also restore files from the archive into a subdirectory. For example, typing

```
fbackup restore /old_files
```

will read files from the archive and write them into the directory **/old_files**. If there are subdirectories on the archive, they will be created as subdirectories of **/old_files**.

If an archive directory is specified on the command line, it is presumed to be a source directory in the archive. This is the directory in the archive from which files are to be read. For example, typing

```
fbackup  restore  /user/fred,/user/george
```

will restore all of george's files from the archive into fred's directory. Note that the path of the archive directory is stripped from the name of the file and that only files within this directory will be restored.

The following examples may clarify things.

```
fbackup  restore  /user/george
      • Restore ALL files from the archive.
        Take the names and save them under /user/george.
        Arch File         ->         Disk File
        /user/george/file            /user/george/user/george/file.
        /cmds/ls                     /user/george/cmds/ls


fbackup  restore  /user/george,/user/george
      • Restore only files under /user/george from the archive.
        Strip the /user/george from each archive name.
        Take these names and save them under /user/george.
        Arch File         ->         Disk File
        /user/george/file            /user/george/file.


fbackup  restore  /user/fred,/user/george
      • Restore only fi es under /user/george from the archive.
        Strip the /user/george from each archive name.
        Take these names and save them under /user/fred.
        Arch File         ->         Disk File
        /user/george/file            /user/fred/file.
```

```
fbackup  restore 3:/
      • Restore ALL files from the archive.
        Take these names and save them under 3:/
        Arch File      ->       Disk File
        /dir/file               3:/dir/file


fbackup  restore 3:/,4:/
      • Restore only files under 4:/ from the archive.
        This assumes an archive dir of ,4:/ when SAVE was run.
        Strip the 4:/ from each archive name.
        Take these names and save them under 3:/.
        Arch File      ->       Disk File
        4:/dir/file             3:/dir/file


fbackup  restore [3]3:/,[2]3:/
      • Restore only files under [2]3:/ from the archive.
        This assumes an archive dir of ,[2]3:/ when SAVE was run.
        Strip the [2]3:/ from each archive name.
        Take these names and save them under [3]3:/.
        Arch File      ->       Disk File
        [2]3:/dir/file          [3]3:/dir/file
```

The **-create** option prevents the RESTORE option from creating new files or directories on the disk. If not specified, new directories will be created if they do not exist. For example, if george's files are all saved in the archive, and then george's directory is released, typing

fbackup restore /user/george,/user/george

will restore all the files, and RESTORE will create directory **/user/george** on the disk so it can write the files into it. If the **-create** option is specified and a necessary directory cannot be found, the RESTORE option will print an error message for the file it was trying to restore and will then proceed to the next file.

The remaining options have the same meaning as they do in the SAVE option. They apply to files in the archive to be restored, instead of files on the disk to be saved.

+list-only
-pause
-verbose
pf=*pattern*
pp=*pattern*
g=*group* **m=*member***
d=*date* **t=*time***
+before

2.4.6 VERIFY - Verify Data Saved to the Tape/Diskette

Syntax:

fbackup [*drive*] verify ***disk-dir***[,***arch-dir***] [*options*]*

tbackup verify ***disk-dir***[,***arch-dir***] [*options*]*

Description:

The VERIFY function can be used to verify that data was saved correctly in the archive. It should follow the SAVE option before any disk files are modified. When this function is specified, the data from the archive IS NOT written to the disk files. Instead the disk files are read and compared to the files in the archive. This function supports the same options as the RESTORE function.

```
fbackup save 3:/
fbackup verify  3:/
```

2.4.7 NAME - Print Volumn Name of an Archive

Syntax:

```
fbackup [drive] name

tbackup name
```

Description:

The NAME function will print the volume name on an diskette or tape. It is useful if you forget to label your archive.

2.5 Examples

The following is an example of the use of INIT option.

```
$ fbackup init 200 "v=test-disk"
$ tbackup init "v=test-tape"
```

In the following examples, we assume that the user's hard disk is mounted as drive 3. We also assume a 1.2meg floppy drive. If this is not the case in your system, you must specify the correct parameters.

Here we describe the commands to be used to implement the incremental backup system outlined in section 1.3.

On Monday, type the command

```
$ fbackup init 200 v=week35 +h
$ fbackup save 3:/,/monday +all
```

The first command creates a brand new archive. You may use any volume name you like. In this example, the volume is named for the week of the year.

The second command will save all files from the hard disk to the archive. This is the complete backup, which happens each week.

On Tuesday, type the command

```
$ fbackup save 4:/,/tuesday
```

On Wednesday, type the command

```
        $ fbackup  save  4:/,/wednesday
```

On Thursday and Friday, follow a similar scheme. These commands save only the files which have been changed that day into a directory with the name of the day.

By the end of the week, you will have made a complete backup of your files, and all of the changes that have been made through the week.

Using the example above, you could restore all files that changed on Thursday by typing

```
    $ fbackup  save  3:/,/thursday
```

If your disk is destroyed for any reason, the contents of the disk can be completely restored from the archive by issuing in order a command for each day.

```
    $ fbackup  restore 3:/,/monday
    $ fbackup  restore 3:/,/tuesday
        .
        .
```

2.6 Saving Multiple Disk Partitions Per Archive

If your hard disk contains more than one QNX partition, say disk 3 and disk 4, and you wish to save them both on one archive you SHOULD use the *,arch-dir* option. This allows you to differentiate there files once they are placed in the archive. For example.

```
    fbackup  save  3:/,3:
    fbackup  save  4:/,4:
```

Or using just one command

```
    fbackup  save  3:/,3:  4:/,4:
```

This can of course be extended to include node numbers in a networked system.

```
    fbackup  save  [2]3:/,[2]3:
```

3. The QNX File Structure

QNX Software Systems Ltd. has chosen a rather unique method of storing files on disks which provides

- **the flexibility of a true hierarchical file structure**
- **the security required in a multi-user environment**
- **the fast access using direct seeking in huge data bases**
- **the space efficiency of a single 512 byte unit of allocation**

while at the same time minimizing the potential damage which could be caused to files when the system crashes (such as during a power failure).

The QNX File System Administrator views a physical drive as a consecutive array of 512 byte BLOCKS (usually disk sectors). The file system assumes a particular format of data which is stored in these blocks. Up to 8 drives are supported by the File system. These drives may be physical drives, or logical "pieces" of a physical drive referred to as partitions.

1.1 BLOCK 1 - The ROOT Block

The first block of any disk is the ROOT of the files system and is usually called "/". The ROOT block contains exactly one directory entry (described later) which is named "/" and which points to the disk block which contains the directory information about the files under "/". The rest of the ROOT block is typically not used and often contains information required for BOOTing the system.

```
Block 0001
000:  00 00 00 00 00 00 00 00 64 00 00 00 00 00 00 00 <- extent header
010:  01 00 01 00 31 5A 2F 1E 00 00 00 00 00 00 00 00
020:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
030:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
040:  00 00 00 00 50 07 00 00 00 6F 2F 00 00 04 00 00
050:  00 05 00 FF FF 00 00 00 00 00 00 00 21 21 35 0F
060:  62 BA 37 00 00 00 00 00 00 00 00 00 00 00 00 00

Offset  010: 01 00 01 00  <- Father dir extent and index.
                             This does not change.

Offset  014: 31 5A 2F 1E  <- Date disk was created.

Offset  044: 50 07 ...    <- Directory entry which points to the
                             root directory just past the bitmap.
                             This entry is 48 bytes.
```

3.2 The BITMAP

Immediately following the ROOT block (block 2) is a file of contiguous blocks called the BITMAP.

2...n   Bitmap (n = 2 + NBLKS/512/8)

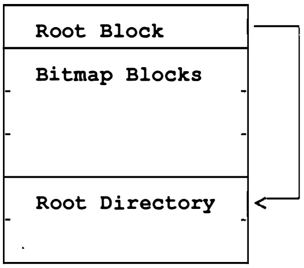
This file contains a map of all the blocks on the disk indicating whether or not a block is used. Each block is represented by a bit in the bitmap with block 1 being the least significant bit of byte 0. If the bit is 1, then the block is in use. The DCHECK command with the **+mark** option will also set bad blocks as used so that no files will attempt to use them.

```
Block 0002
000:  00 00 00 00 00 00 00 00 4C 09 00 00 04 00 00 00  <- extent header
010:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  <- file starts on
020:  FF FF FF FF FF FF FF FF FF FB FF FF FF FF FF FF     this line.

First byte at location  010: FF        (hex)
                             11111111  (binary)
                             |      |
                             |      Block 1 of disk
                             Block 8 of disk
```

The QNX file system uses the bitmap for allocating blocks for new files or files which must grow. The bitmap is always kept up to date on the disk (not kept in memory) such that in the event of a system failure, the bitmap and directory entries will never be inconsistent.

The ROOT block points to the first block past the bitmap which will contain the root directory, as shown:



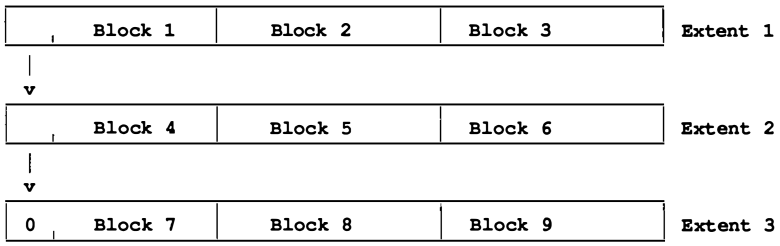
3.3 Files

A QNX file consists of a linked list of EXTENTS which are contiguous arrays of blocks. Wherever possible, the QNX file system will try to keep a file contiguous. If a file must grow, but the next contiguous disk block is used, then a new "extent" is created and linked into the file.

As a result of the extent philosophy, QNX files may grow to any size (limited only by the available space on the disk). Keeping files as contiguous as possible keeps disk I/O fairly efficient.

The extent information is kept within the file itself avoiding the need to seek to a known region of the disk to find the next block of a file while still allowing the information to be maintained on the disk rather than in memory. This ensures consistency of the file system after system crashes. In particular, finding any extent of a file will allow complete recovery of the file.

A typical QNX file has the following structure:



Note: For efficiency, QNX caches the 16 byte extent headers.

3.4 Extents

Every QNX file is a linked list of "extents". An extent is a contiguous array of disk blocks. The first 16 bytes of an extent is called an "extent header" which contains the following information:

```
struct xtnt_hdr {
     long prev_xtnt;         /* block address of previous extent */
     long next_xtnt;         /* block address of next extent */
     long size_xtnt;         /* size of this extent in characters */
     long bound_xtnt;        /* offset of last block in this extent */
     };
```

prev_xtnt is zero in the first extent of a file. Similarly next_xtnt is zero in the last extent of a file.

For example

```
Block 0ED1  Status: 00
000:   00 00 00 00 F8 07 00 00 F0 05 00 00 02 00 00 00  ...
        | prev_xtnt  | next_xtnt  | size_xtnt  | bound_xtnt|
```

indicates that the previous extent is zero and the next extent is at block 000007f8. The size is 000005f0 characters and the number of blocks in the extent is 3 (add one to bound_xtnt).

After the 16 byte extent header, all remaining bytes are data.

The fields **size_xtnt** and **bound_xtnt** are related. One states the number of characters in the extent and the other states the number of blocks in the extent. For a closed file or a file open for read the following relationship will always be true.

size_xtnt = (bound_xtnt + 1) * 512 - 16

If the file is open for write then this relationship may not be true for the last extent only. When a file is grown, it is grown by more than one block at a time. As a result the **bound_xtnt** keeps track of the number of blocks allocated to the extent while **size_xtnt** keeps track of the number of characters written to the extent (user data). The amount to grow the file depends on how big the file currently is. When the file is closed the extra blocks will be given back to the bitmap and **bound_xtnt** shrunk to the proper size.

When growing a file, if free blocks exist immediately after the last extent, then the blocks will be allocated by setting their bits in the bitmap (indivisible operation) and then increasing "bound_xtnt" in the file system's local copy of the extent header. Subsequent writes will fill up these blocks.

When the extent cannot be grown further, the file system will then link in a new extent into the file. First a block of memory is allocated from the bitmap. The first block of the new extent is initialized to have a size of zero, a next pointer of zero, a bound equal to the possible number of blocks in the extent, and a previous pointer to the first block of the (now) previous extent. The first block of the previous extent is then updated with new size/bound/next information.

The process continues until the file is closed, or no more room exists on the disk.

When the file is closed, the extent header of the last extent is updated with the correct size/bound information and then the bitmap is updated if any blocks allocated to the file were not actually used.

Finally, the directory entry is updated (indivisible) with the correct size, first and last extent information.

Notice that at no time will the file ever be inconsistent, although the bitmap may indicate more than the necessary number of blocks have been allocated on the disk and the last extent and size information in the directory may not be correct.

Therefore, in the event of system crashes, it is usually possible to read the data in a file (which will contain everything flushed to the disk up until the time the system went down), but since the last extent information may not be correct, it is wise to use the CHKFSYS command to fix the directory entry. If a file can not be corrected using CHKFSYS then use the ZAP command to erase the directory entry of that file without attempting to give back the blocks to the system.

In addition, if a file is open for read/write (eg. a database file) and if the file is not grown, then all of the extent and directory information will not have been changed so it is possible to recover the file to the point of its last disk flush by removing the BUSY bit in the file's directory.

```
chattr s=-b file
```

The BUSY bit in a file's directory is set whenever a file is open for write or read/write and grows. It is cleared when the file is closed. The reason that this bit is maintained in the directory (therefore on the disk) and NOT internal to the file system is to ensure that potentially damaged files will be detected after system failures. If you do internal writes in a file open for read/write and the file does not grow then the BUSY bit will not be set.

The BUSY bit also provides lockout of files preventing other tasks from opening a file which is already open for write or read/write, although this information is also maintained internal to the file system.

3.5 Directories

QNX directories are QNX files which just happen to contain a known pattern of data.

Each directory file consists of a 4 or 6 byte header followed by one or more directory entries which are described below. Since directories are really files, they can grow to any size.

The first physical block of a directory contains the standard 16 byte extent header. This is followed by data as follows.

```
struct directory_file {
    unsigned parent_xtnt;
    unsigned dir_index;          <- if top bit set, next word is an
                                    extension for parent_xtnt
    struct dir_entry directory[0...n];
    };
```

The 48 byte directory entry is structured as follows:

```
struct dir_entry {
    char fstat;
    long ffirst_xtnt;
    long flast_xtnt;
    long fnum_blks;
    unsigned fnum_xtnt;
    unsigned char fowner;
    unsigned char fgroup;
    unsigned fnum_chars_free; /* No. unused chars in last block */
    long fseconds;
    unsigned char ftype;
    unsigned char fgperms;
    unsigned char fperms;
    unsigned char fattr;
    unsigned fdate[2];
    char fname[17];
    };
```

The bits in fstat are defined as follows:

```
#define _FILE_MODIFIED  0x10
#define _USED           0x40
#define _FILE_BUSY      0x80
```

The bits in fattr and fperms are defined as follows:

```
#define _READ           0x01
#define _WRITE          0x02
#define _APPEND         0x04
#define _CREATE         0x04
#define _EXECUTE        0x08
#define _BLOCK          0x08
#define _MODIFY         0x10
#define _DIRECTORY      0x20
```

This structure is defined in the file **/lib/lfsys.h**.

Note that in the above structure, there are two fields which can hold the file date: **fdate** and **fseconds**. The field **fseconds** is new to QNX 2.10. It stores the date and time as the number of seconds offset from 0:00:00 GMT, January 1, 1980. The field **fdate** will be de-supported with the next release. In the meantime, if your application has opened a directory for read, you should look first at the field **fseconds**. Only if it is zero (**0**) should you examine the field **fdate** (in case the file was written under an older version of QNX). If you use the function *GET_DIR_ENTRY* to obtain the directory entry for a file, both date/time fields will be correctly filled in (even though both may not be stored on disk). If you use the function *SET_DIR_ENTRY* to modify a directory entry, both file date fields will be recomputed. Changing one field will automatically result in both changing (QNX is smart).

4. Recovering Files On a Disk

This technical note will describe several procedures for recovering files, and directories which have been lost through a software or hardware fault. If the fault appears to be software and is repeatable, please contact QNX Software Systems Ltd.

4.1 Utilities Provided

QNX Software Systems provides the following utilities to aid you.

chkfsys	**- Check file system for consistency and rebuild the bitmap. Can be used to recover zapped blocks (see ZAP command).**
ddump	**- Dump raw blocks on a disk.**
dinit	**- Re-initialize a disk.**
relink_file	**- Create a directory entry and point it to the start of a lost file or directory.**
scan_for_dir	**- Scan the disk, block by block looking for a directory.**
scan_for_file	**- Scan the disk, block by block looking for a file.**
spatch	**- Display and edit raw blocks on a disk.**

All references to a disk refer to a mounted QNX disk or partition which may be a subset of the physical disk. All blocks are relative to the start of the mounted volume. The examples assume that drive 1 is a floppy, drive 3 the disk which we are trying to restore and drive 5 some other mounted disk (ramdisk, floppy etc..). Typing any of the commands listed, followed by a question mark will cause it to print a very terse usage message.

4.2 Recovering a Single File on the Disk

The first step in recovering a file is determining where on the disk it starts. Given that, the RELINK_FILE command can be used to relink to the file.

You may find a file on the disk using the SCAN_FOR_FILE command if you know some of the data within the file. It will simplify your task considerably if the data is contained within the first block (512 bytes) of the file. Lets assume we are looking for a file which contains the data "#include manifests.h". Enter the command

```
scan_for_file 3 "#include manifests.h"
```

The command will read raw blocks from the disk looking for the text indicated. Text which spans a block boundary will NOT be matched so keep patterns as short as possible. If more than one text string is entered then a match will occur on the first block containing ANY of the text strings.

```
scan_for_file 3 "match this" "or this"
```

If you are lucky it will find the start block of the file and chain through the blocks making up the file. You may verify that the correct file was found by dumping the file using the DDUMP command or executing the command again specifying a starting block and redirecting your output. For example, if a match was found at block 6a7 then issue.

```
ddump 3 6a7
   or
scan_for_file 3 "#include manifests.h" b=6a7 >5:/tmp/data
```

If you match the text but it is not the correct file you may continue scanning past the matched block using the **b=** option specifying the next block.

```
scan_for_file 3 "#include manifests.h" b=6a8
```

If you match text which is not in the first block, then the command will issue an error. Use DDUMP to verify whether you are within the correct file. If you are then you will have to use DDUMP to move backward through the file until you locate the first block. This is described later on.

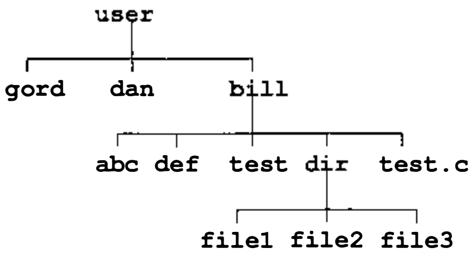
You may relink to the file or simply use the redirected output of the scan command. Always redirect the output to another disk to prevent overwriting of the file you are trying to recover. REMEMBER that to use relink your current directory MUST be on the same disk as the file. The directory entry will be created in your current directory.

4.3 Recovering a Directory on the Disk

The only thing worse than losing a file is losing a directory. Since it is the directories which contain the information on locating the files beneath it, the loss of a directory can imply the loss of all files in that directory. Losing the root of the file system means that you have lost all files on your disk! There are several possible situations.

4.3.1 A Single Entry in a Directory Was Corrupted

The command SCAN_FOR_DIR is analagous to SCAN_FOR_FILE except it matches the names of files in a directory. For example assume a directory structure as follows.



If the directory 'user' is consistent except for the entry for bill then you can locate the directory for bill as follows.

```
scan_for_dir 3 abc test.c
```

The names of files indicated MUST be in the first block of the directory. You may specify several names to increase your probability of a match. Redirecting the output of this command to capture an image of the directory is possible but is of little use. You must relink to the directory to regain access to it. For example if a match was made at a68.

```
relink_file bill1 3 a68 +directory
```

The +directory option is required. If you forget it then you will have to remove bill1 and try again.

```
zap bill1
relink_file bill1 3 6a8 +directory
```

You may specify a starting block to skip over an incorrect match.

```
scan_for_dir 3 abc test.c b=a69
```

4.3.2 An Entire Directory Was Corrupted

You must find the start blocks of all files and directories in the corrupted directory. It is NOT necessary to locate subdirectories. In the example above, if the entire directory for bill was lost then you would have to use SCAN_FOR_FILE and SCAN_FOR_DIR to locate

```
abc  def test  dir  test.c
```

on the disk. Record them on a sheet of paper. Now create a new directory under user called bill1 as follows.

```
mkdir  3:/user/bill1
```

Position yourself at that directory and issue multiple relink commands. For example:

```
cd 3:/user/bill1
relink_file abc      3  805
relink_file def      3  813
relink_file test     3  83a
relink_file dir      3  84c  +directory
relink_file test.c   3  902
```

Zap the corrupted directory out of existence and rename bill1.

```
chattr  a=+w  3:/user/bill
zap  3:/user/bill
chattr  n=bill  3:/user/bill1
```

You should now run CHKFSYS to ensure the integrity of the file system and to correct any errors in the bitmap.

```
chkfsys 3
```

4.3.3 The Root of the File System Was Corrupted

If you are unable to access any files on the disk you have probably destroyed some of the first blocks on the disk. The structure of the first n+1 blocks are as in detail in the technical note called **"The QNX File Structure"** and are summarized below.

BLOCK	**DESCRIPTION**
1	**Root descriptor block**
2...n	**Bitmap (n = 2 + NBLKS/512/8)**
n+1	**Root directory (ie: /)**

Remove your hard disk from your search order

```
1:/cmds/search 1   - assume drive 1 is a floppy
```

and use the DDUMP command to dump the first couple of blocks on your disk.

```
ddump 3 1
```

If the status returned is other than 00, then a hardware fault has occurred. In the case of bad status, try dumping other blocks to determine if the fault is local. If all blocks appear bad, your data may still be recoverable if the fault is in the controller and not on the physical media (head crash for example).

The first block of your disk should NOT look like a text file, nor should it consists of all zeros, ones etc... The bitmap blocks should consist mostly of FF,'s and 00's. The root directory should contain the names of your root directories (bitmap, cmds, lib, config ...). We recommend that you keep a dump of the root directory block's in a safe place.

```
ddump 3 1 >$lpt
```

It contains the mapping to ALL sub-directories and files on your disk.

A good alternative to listing the first blocks of the disk is to DCOPY the first few hundred blocks of your hard disk onto a FLOPPY disk which is then kept in a safe place.

Of course the best safeguard is to establish a good backup policy involving daily incremental backups and periodic full backups onto floppy disks, streamer tape or some other archive media. The BACKUP, FBACKUP or TBACKUP commands may be used for this.

The following is a partial dump of a 10 Meg hard disk partition.

ddump 3 1

```
Block 0001  Status: 00
000:  00 00 00 00 00 00 00 00 64 00 00 00 00 00 00 00 ........d.......
010:  01 00 01 00 11 3B A6 8A 00 00 00 00 00 00 00 00 .....;..........
020:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
030:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
040:  00 00 00 00 50 08 00 00 00 97 05 00 00 05 00 00 ....P...........
050:  00 04 00 FF 00 00 00 00 00 00 00 00 00 25 25 27 .............%%'
060:  49 81 93 00 00 00 00 00 00 00 00 00 00 00 00 00 I...............
070:  00 00 00 00 DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .....n.n.n.n.n.n
080:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
090:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
0A0:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
0B0:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
0C0:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
0D0:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
0E0:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
0F0:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
100:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
110:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
120:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
130:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
140:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
150:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
160:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
170:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
180:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
190:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
1A0:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
1B0:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
1C0:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
1D0:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
1E0:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
1F0:  DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E DB 6E .n.n.n.n.n.n.n.n
```

```
Block 0002  Status: 00
000:  00 00 00 00 00 00 00 00 29 0a 00 00 05 00 00 00 ........).......
010:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
020:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
030:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
040:  FF FF FF FF EF FF FF FF FF FF FF FF FF FF FF FF ................
050:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
060:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
070:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
080:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
090:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0A0:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0B0:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0C0:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0D0:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0E0:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0F0:  FF FF 31 72 8E F7 FF 8F CF FD F8 FF FF FF FF FF ..1r............
100:  EF F9 3F FF FF FF FF FF FF FF FF FF FF FF FF CF ..?.............
110:  7F FF FF FF DF EF FF FF FF 1F F5 23 FE 8F FF 1F ...........#....
120:  FD 3E 3E 26 1E FF FB 3F F9 FF 3F 7B BE FF FF FF .>>&...?..?{....
130:  7F FF FF FF C1 FF FF FF DF F8 FF FF FF FF FF FF ................
140:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
150:  FF FF FF FF FF FF FF FF FF FF FF 7F FC 9F 10 FC ................
160:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
170:  FF FF FF FF FF FF F9 03 30 06 FC F8 FF FF FF FF ........0.......
180:  FF FF FF FF FF FF FF FF FF 05 E0 03 E0 FF FF 47 ...............G
190:  FA FC FF FF FF FF FF FF FF F7 FF FF FF FF 9F FF ................
1A0:  2F FE FF FF FF FF FF FF FF FF FF FF FF FF FF FF /...............
1B0:  EF FF FF FF FF FF FF FB FF FF FF 7F EC F8 FF FF ................
1C0:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
1D0:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
1E0:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
1F0:  FF FF FF FF FF FF FF FF FF FF FF FF FF 7F FF 38 ...............8
```

```
Block 0008  Status: 00
000:   00 00 00 00 A8 06 00 00 F0 01 00 00 00 00 00 00 ................
010:   01 00 01 00 40 02 00 00 00 02 00 00 00 06 00 00 ....@...........
020:   00 01 00 FF 00 00 00 00 00 00 00 00 01 01 31 ...............1
030:   35 2E 56 62 69 74 6D 61 70 00 00 00 00 00 00 00 5.Vbitmap.......
040:   00 00 00 00 50 09 00 00 00 28 28 00 00 10 00 00 ....P....((.....
050:   00 03 00 FF 00 00 00 00 00 00 00 00 21 35 1A .............!5.
060:   42 3A 9C 63 6D 64 73 00 00 00 00 00 00 00 00 00 B:.cmds.........
070:   00 00 00 00 50 17 00 00 00 17 00 00 00 06 00 00 ....P...........
080:   00 01 00 FF 00 00 00 00 00 00 00 00 21 35 11 .............!5.
090:   42 B1 51 6C 69 62 00 00 00 00 00 00 00 00 00 00 B.Qlib..........
0A0:   00 00 00 00 50 1E 00 00 00 1E 00 00 00 06 00 00 ....P...........
0B0:   00 01 00 FF 00 00 00 00 00 00 00 00 21 35 3B .............!5;
0C0:   41 DB 59 62 6C 69 62 00 00 00 00 00 00 00 00 00 A.Yblib.........
0D0:   00 00 00 00 50 79 02 00 00 79 02 00 00 02 00 00 ....Py...y......
0E0:   00 01 00 FF 00 00 00 00 00 00 00 00 21 35 09 .............!5.
0F0:   41 D5 02 6D 61 74 68 6C 69 62 00 00 00 00 00 00 A..mathlib......
100:   00 00 00 00 50 89 02 00 00 89 02 00 00 02 00 00 ....P...........
110:   00 01 00 FF 00 00 00 00 00 00 00 00 21 35 09 .............!5.
120:   41 1D 35 63 6F 6E 66 69 67 00 00 00 00 00 00 00 A.5config.......
130:   00 00 00 00 50 9F 02 00 00 9F 02 00 00 09 00 00 ....P...........
140:   00 01 00 FF 00 00 00 00 00 00 00 00 21 35 23 .............!5#
150:   41 EF 1B 65 78 70 6C 00 00 00 00 00 00 00 00 00 A..expl.........
160:   00 00 00 00 50 3D 04 00 00 3D 04 00 00 02 00 00 ....P=...=......
170:   00 01 00 FF 00 00 00 00 00 00 00 00 21 35 09 .............!5.
180:   41 E4 02 65 74 63 00 00 00 00 00 00 00 00 00 00 A..etc..........
190:   00 00 00 00 50 AC 04 00 00 AC 04 00 00 02 00 00 ....P...........
1A0:   00 01 00 FF 00 00 00 00 00 00 00 00 21 35 09 .............!5.
1B0:   41 E6 02 71 73 70 65 6C 6C 5F 6C 69 62 00 00 00 A..qspell_lib...
1C0:   00 00 00 00 50 87 06 00 00 87 06 00 00 02 00 00 ....P...........
1D0:   00 01 00 FF 00 00 00 00 00 00 00 00 21 35 09 .............!5.
1E0:   41 EC 02 64 6F 63 73 61 6D 70 6C 65 73 00 00 00 A..docsamples...
1F0:   00 00 00 00 50 A9 06 00 00 A9 06 00 00 02 00 00 ....P...........
```

4.3.3a Root Directory Uncorrupted

If the root directory appears uncorrupted, you may recover by doing a DINIT on your disk with the +suppress option.

```
    dinit 3 +suppress
```

This will write out a new root descriptor block and a new bitmap. The new bitmap will indicate all blocks on the disk as free so DO NOT write any files to your disk. If there are any files which are critical, back them up NOW. You can rebuild the bitmap using the the CHKFSYS command with the **+rebuild** option. This option suppresses messages related to a corrupt bitmap. At this point you know your bitmap on the disk does not match your file structure.

```
    chkfsys 3 +rebuild
```

Answer yes to the prompt to write back the bitmap. Now run the command again without the rebuild option.

```
    chkfsys 3
```

If there are no errors, then your disk has been successfully recovered. If there are errors, you can try and fix them, otherwise you will have to DINIT without the **+suppress** option and backup the disk from scratch using your backup files.

4.3.3b Root Directory Destroyed

If the root directory has been destroyed, DINIT your disk.

```
    dinit 3
```

You will now have to use the SCAN_FOR_DIR and SCAN_FOR_FILE commands to locate the first block of each directory and file which was at the root of the file system. Once located, use the RELINK_FILE command to relink to the directories and files. REMEMBER to use the **+directory** option for directories and to position your current directory at the ROOT.

```
    cd 3:/
```

Most users do not keep files under the root, only sub-directories. This greatly reduces the number of relinks required. After relinking, run the CHKFSYS command twice as above.

```
    chkfsys 3 +rebuild
    chkfsys 3
```

4.4 The Structure Of A File

This is described in the technical note called **"The QNX File Structure"**.

5. Page Caching Disk Drivers

A page is a contiguous set of 512 byte disk blocks. Page caching drivers will do both read-ahead and write-behind to maximize performance. Each driver has a default cache as follows:

Driver	**Pages**	**Pagesize**
disk.ps2	**8**	**8 blocks**
disk.ps2esdi	**10**	**16 blocks**

You may override the cache size using the **a1=*numpages*** and **a2=*pagesize*** parameters on the MOUNT command. The maximum number of pages supported is 32 and the maximum pagesize is 16 blocks. This cache is intended to speed up sequential and record I/O as well as reducing head movement when multiple users access the disk at the same time.

The read-ahead means that requests for a single disk block may result in an entire page of disk data being read in a single transaction. When the task next requests a block of data, chances are very high that the data will already be in memory.

The write-behind means that when a task writes a block it may not go immediately to disk. The hope is that the task will provide another sequential block and eventually fill an entire page which can then be written with one disk transaction. To prevent volatile data from staying in the cache, the system ages each page, and after **one second** forces the page to disk. **This means you should always pause 1 second after a write before typing Ctrl-Alt-Shift-Del or hitting the big RED switch.** As an added precaution, QNX will always write directory and bitmap information through the cache onto disk immediately.

The combination of read-ahead and write-behind optimization means that delays due to rotational latency and head repositioning need occur for whole pages, not for each block.

If write-behind scares you, you can turn it on or off when you mount the driver using the **a3=0|1** parameter to MOUNT. You can use the MOUNT command to change the default at any time. For example, the following will mount a driver with write-behind disabled, enable it and then disable it.

```
mount disk 3 /drivers/disk.ps2esdi pa=qnx a3=0
mount disk 3 d=3 a3=1
mount disk 3 d=3 a3=0
```

QNX Version 2.2

Operating System

PRINTED IN CANADA