# FileCore - Phase 2
# Functional Specification

**Confidential Information**

| | |
|---|---|
| Document Ref: | 1309,208/FS |
| Project: | Ursula |
| Revision: | 0.05 |
| Date: | 06-01-1998 |
| Author(s): | Simon Proven |
| Change: | None |

## Contents:

# 1.0 Overview

The Ursula FileCore Phase I specification deals with the changes required to FileCore to reduce the size of disc objects, by increasing the allowed value of idlen (previously, idlen was limited to 15 bits). Due to the limitations of the directory format, it is not possible to increase idlen beyond 16 bits, without altering the directory format. In order to gain more than a factor of 2 benifit in minimum object sizes, a new directory format must be introduced.

As the directory format must be changed, we will take this opportunity to enhance FileCore directories in other ways. The goals for this change are:

- Allow idlen to be larger than 16. The new maximum permitted value of idlen will be 19 bits, allowing free space maps of upto 1280K bytes.
- The new directory format will store the file name length as a word value. However, the source will be built to constrain the file name length to 255 characters. This value can be altered at compilation time.
- Variable length directories to cope with variable length of file names, and variable number of entries per directory.
- Backwards compatability will be supported with E and F format floppies, and existing E format hard discs. D format discs should work, but are deprecated; E format or later should be used in preference. L format discs are also deprecated, and are not supported, although the code to support them will remain in FileCore. L format discs will not be tested; they will be removed from the formatting menus in ADFSFiler and from *format in ADFS. S and M format discs are not supported.
- DOSFS and other image filing system support will continue as at present.

# 2.0 Outstanding Issues

None.

# 3.0 Technical Background

Directories on FileCore E-format discs are currently of a fixed size (2048 bytes). Directory entries are also a fixed size. This results in a maximum 77 files per directory, and a maximum file name length of 10 characters.

This specification proposes a new directory format. When a disc is formatted, the user will be able to specify that the new directory format is to be used. When this new directory format is used, the disc will not be readable on earlier versions of FileCore. The error "broken directory" will occur when attempting to access any directory on the disc. If, for a given disc, idlen is greater than 15, then we will also get "Disc not understood - has it been formatted" on versions of FileCore that are not aware of the changes.

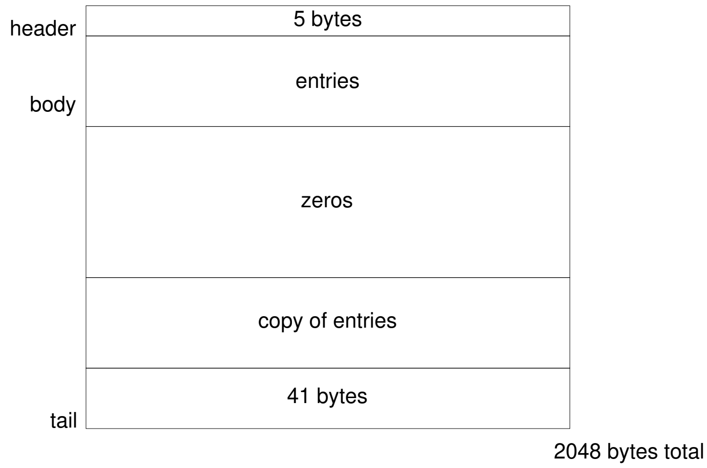
## 3.1 New Directory Format Features

The new directory format will have the following features:

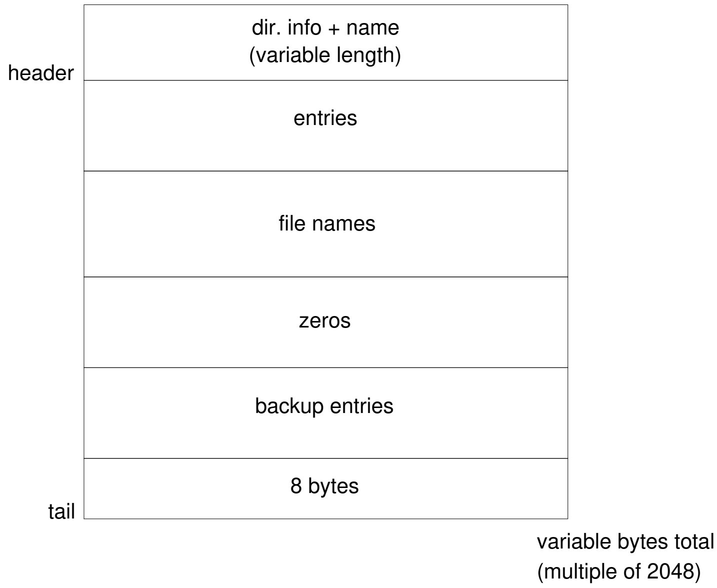
- Object names are separated from other information, to allow variable length names.
- Directories will be of variable length; they will be able to grow to accomodate new or extended entries.
- Directories can shrink when entries are removed, or are reduced in size.

## 3.2 Layout of a Directory

The current directory format has the following structure:




The new directory format looks like this:




## 3.3 Directory Entries

In the new directory format, the directory entries contain the information about file attributes (length, load address, etc), but do not contain the directory name. Instead, the directory names are held in the middle of the directory, immediately after the directory entries.

The directory names are held in the same order as the entries. Note that as directory entries are held in the directory in alphabetical order in the old directory format, this constraint will be maintained in the new directory format.

The name pointer is the offset from the start of the name space to the start of the corresponding name heap entry. All the name heap entries are held at word aligned offsets into the name heap.

## 3.4 The Name Heap

The name heap holds the names for all the entries in the directory. Each name is simply stored as a CR (&0d) terminated string, followed by 0 or more padding bytes, to pad the next name in the heap to a word boundary. The name heap entries are held in the order in which they occur in the directory. Note also that directory entries are always held in alphabetical order.

## 3.5 Backup Directory Entries

The backup directory entries are held immediately before the directory tail. They are present to make it easier to scavenge broken directories to reconstruct any damaged entries. Because it is only necessary to store the indirect disc address of a file to allow its recovery from the free space map, the backup directory entries only hold this information, avoiding wasteful duplication of data.

## 3.6 Size of a Directory

Variable length directories will always be a multiple of 2048 bytes in size, with a minimum size of 2048 bytes; a directory which is not a multiple of 2048 bytes in size is considered to be broken.

### 3.6.1 Maximum Directory Size

The maximum size of a directory will be 4,096K bytes. The formula below applies:

$$\frac{(max\_dir\_size) - (tailsize + headersize + round\_up(dir\_name\_size + 1))}{(entry\_size) + (round\_up(average\_name\_size + 1))}$$

For names of upto 19 characters, we can have upto 87380 directory entries.

### 3.6.2 Unused Space in a Directory

In the existing directory format, the unused space is filled with zero bytes. Similarly, in the variable length format, any unused space in the directory is filled with zero bytes.

## 3.7 The Dir Buffer

The directory buffer is the place where FileCore keeps the directory currently being worked on when modifying it. The directory buffer on earlier versions of FileCore is a fixed-length buffer (because directories are at most 2048 bytes). Since we are allowing directories to grow, it is no longer practical to make the directory buffer a fixed-length buffer. Thus, the directory buffer will be held in a dynamic area, which can grow to the size required to manipulate a given directory.

The dir buffer will be initially allocated a size of 8K bytes. This will allow most directories on a given disc to be read without need to grow the buffer. To allow the buffer to grow, the maximum size of the dynamic area will be specified as the maximum directory size (ie 4096K bytes).

When FileCore requires to read a directory, it will first attempt to grow its directory buffer to be sufficiently large that it can hold the directory in its directory buffer. If the directory buffer cannot be grown to the size required, then an error is produced. The error must be handled gracefully, and must not cause data loss.

Once the directory buffer has been grown to manipulate a given directory, it will not be reduced in size.

## 3.8 The Dir Cache

FileCore caches recently accessed directories in RAM, to optimise performance when accessing multiple directories or when accessing deep directory structures. When directories are held in the directory cache, they are held in a more compact form than on the disc. The backup entries are removed and the unused space is removed.

When the directories are in this compacted form in the cache, they cannot be modified. For modification, they are copied into the directory buffer, changing to the expanded form. Note that the move from the dir cache to the dir buffer doesn't re-generate the backup entries or zero the unused space; this operation is performed immediately before writing the directory back to the disc after modification, as with the current implementation.

## 3.9 Reading a Directory

Currently, FileCore always knows how much data must be read when reading a directory into the directory buffer; the size is constant for a given disc. However, when we have variable length directories, we cannot assume a constant size to read. As FileCore doesn't pass around the length of a directory internally (as there has been no need to in the past), we can't rely on knowing the size of the directory before we read it.

To read a variable length directory, when the length isn't available, FileCore will read the first 2048 bytes (which must always be present) and will check the length field in the header to determine if it must read more data to read the entire directory. It will sanity check the value of the length field before attempting the read operation, and will also check that the header looks valid; in particular, it will check that the StartName is correctly formed and that all the header fields have sensible values (to be defined).

Thus, if a directory is more than 2,048 bytes in length, two read operations are performed to access it. As directories are not buffered by the random access file cache (for very good reasons), there will be two physical operations on the disc for each directory read. For floppies, this will mean that there will be an extra delay when reading large directories, due to rotational delay. However, on floppies, it is likely that there will be very few large directories.

For hard discs, the cache on the disc will probably contain the remainder of the directory, due to the read ahead nature of the caches on such devices.

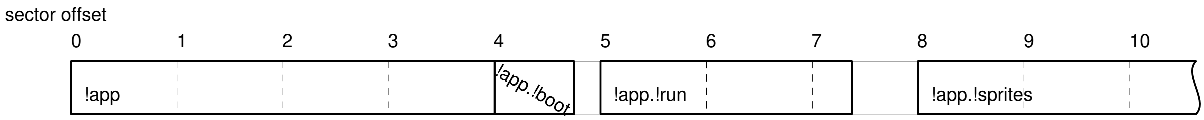
## 3.10 Writing a Directory

When a directory is modified, it must be written back to the disc. Currently, the directory modification is performed, then the entries are copied, and the unused bytes are zeroed. Then, the directory is checked to ensure that it is well formed (to avoid accidentally writing back corrupt directories). The Start and End master sequence numbers are then written into the directory; finally, the directory's check byte is calculated and inserted, and the directory is written to the disc.

The operation of writing a directory remains the same when dealing with variable length directories, except that only the indirect disc adresses are copied for the backup directory entry information.

## 3.11 Extending a Directory

With variable length directories, we must support the operation of extending a directory. Because directories are by default created as shared objects, we have the potential problem that the directory cannot be extended unless it is moved or other shared objects are moved to make space for it. Consider the situation where we have a directory which has two shared files in the same disc object:




In order to extend the directory !app, we need to move files !app.!boot and !app.!run to new locations (!app is being extended from 2048 bytes, or 4 sectors, to 4096 bytes, which is 8 sectors). Suitable locations would be:

- Later in the same shared object, if space is available.
- In another shared object; files shared with must be in the same directory as the file being moved.
- In a new disc object; this would probably be created as a shared object, with only one item held within it.

The operation to move each file to another place can be done relatively easily using existing code in FileCore, with some minor alterations. The only difficulty will be that files which move may be open files; we have to ensure that any files which are open are moved safely.

Note that once a directory is more than a given size, placing shared objects after the directory becomes less useful. Thus, if a directory is more than the minimum object size in length, or if sharing is not possible because the sharing offset would be too large (the maximum sharing offset of a file is 254 units) then files will not be created as shared objects after the directory.

## 3.12 Directory Check Byte Calculation

The directory check byte is calculated from an algorithm operating on all the bytes in the directory, excluding the space between the last directory entry and the directory tail. This algorithm continues to operate as at present, with one change. Currently, the last word of the directory is not included in the check, as it contains the EndName, which checked separately, and the directory check byte, which is not included in the calculation. For new format directories, the last word will contain useful information, so the first 3 bytes will be included in the check byte calculation.  As this is not a whole word, each byte will be accumulated separately, as the current algorithm does for parts of words.

## 3.13 Root Directory Disc Address

[3] specifies that when idlen is more than 15, the root directory is no longer held in the same object as the free space map.  With variable length directories, it is necessary to allow for the root directory to grow.  Therefore, when there are variable length directories on a given disc, the root directory is held in a separate object, as specified in [3].

## 3.14 Backwards Compatability

There are a number of backwards compatability issues, which are summarised here.

### 3.14.1 Compatability With Existing Applications Software

Where existing applications only use legal FileSwitch and FileCore interfaces to access the disc, and where the software makes no attempt to access or modify the low-level structures on the disc, there will be no problem with running existing software, except that:

- Long filenames may give some applications difficulties; many assume that the maximum leafname of a file is 10 characters.  However, NFS and DOSFS have been supplying longer filenames for some time now, so this is not a new problem.
- Long directories may also cause software some problems; if software assumes a maximum of 77 entries in a directory, when examining directories.  Again, both DOSFS and NFS result in directories with more than 77 files, so this is not a new problem.

Effectively, only poorly written or very old software is likely to suffer from major problems of this nature.  However, in order to ensure that when the product is launched, as large a number of applications work successfuly as possible, application developers should be informed of the need to support long file names and large directories.

### 3.14.2 Backwards Compatability With Disc Recovery Utilities and Similar Software

Many disc utilities (especially ones which attempt to fix corrupted discs), require an understanding of the low-level disc structures.  Where idlen is more than 15, or where the new directory format is in use, or both, it is highly unlikely that such disc tools will operate correctly, as they require an understanding of the structures on a disc.  If they attempt to modify the structures on the disc, they may cause serious damage to the disc structures.

It is highly important that:

- Users are made well aware of the problems with using such tools.  The welcome and user guides should make this very clear.
- Developers are made aware of the impact of the changes as early as possible.
- A list of tools known to be a problem should be provided with the machine.  This list should document where upgrades can be obtained, and should be created in coordination with developers.

### 3.14.3 Disc Format Compatability

Old disc format support continues as with RISC OS 3.71, so all old discs which RISC OS 3.71 works correctly with, the new FileCore will work with correctly.

To summarise, all ADFS/FileCore disc formats from D format onwards will work; earlier formats are not supported. D format is, however, deprecated, so testing on this format will be less extensive than on other disc formats.

### 3.14.4 *Configure Truncate Behaviour

When Truncate is configured to On, currently, names are truncated to 10 characters if they are longer than this. Otherwise, names are not truncated, and attempts to create an object name of more than 10 characters result in an error. On discs with new format directories, the behaviour is similar, except that the maximum file name length will be 255 characters.

### 3.14.5 FileCore Disc Version Number

In order to allow future applications to determine if they can understand the format of a given discs, a new disc record field, the FormatVersion field is introduced. This is held at offset 44 into the disc record, and is a word value. On all existing discs this will be zero; on discs with big directories, this will be 1.

## 3.15 Boot Block Disc Record

The boot block disc record must contain the following fields to allow the disc to be mounted correctly by FileCore. All other fields should be set to zero.

*log2secsize, secspertrack, heads, density, idlen, log2bpmb, skew, lowsector, nzones, zone_spare, root, disc_size, disc_size_2, share_size, big_flag, nzones_2, format_version*

## 3.16 Free Space Map Memory Allocation

Currently, FileCore allocates the free space map and the associated control structure (one byte per map zone) from the RMA. When FileCore is initially started, the free space maps are allocated sizes from suggested values passed by the parent module (eg ADFS) when FileCore_Create is called. This helps to prevent RMA fragmentation, since ADFS stores the last known size of each free space map in CMOS RAM.

This becomes problematic now, since the largest free space map size that can be passed is 65280 bytes (the value passed is the size of the map in 256 byte units, and each disc has one byte, hence this value).

FileCore's new behaviour will be to create dynamic areas for all configured drives, when initialising. Each dynamic area will initially have a size of zero bytes. The dynamic area's maximum size will be 4096K bytes. This is thus defined as the maximum permitted size of a FileCore free space map.

When FileCore needs to mount a disc, it will alter the size of the relevant dynamic area to correspond to the size of the free space map for that disc. Note that when FileCore first attempts to mount a disc, if there is no free memory, then it may not be able to allocate the dynamic area, since the memory in the machine may be claimed by another process.

FileCore will also free the space used in a dynamic are when the disc is dismounted.  This will prevent permament loss of memory when briefly accessing a removable media with a particularly large free space map, for instance.

The dynamic areas will be named "<fsname> Map <n>" where <n> is the drive number of the disc to which the map belongs, and <fsname> is the name of filing system.  So, for example, ADFS would have names such as "ADFS Map 0".

The dynamic areas will be protected from USR mode write access, to prevent corruption by errant USR mode code.

# 4.0 User Interface

## 4.1 *map

The *map command is altered to show "big directories" instead of "new directories" when large directories are in use.  Currently, *map on E format discs shows:

```
(   start,  length) new map, new directories
```

On discs with big directories, it will show:

```
(   start,  length) new map, big directories
```

When idlen is more than 15, and with new directories, it will show:

```
(   start,  length) big map, big directories
```

## 4.2 Other User Interface Changes

ADFS must be modified to support formatting floppies with large directories.  The options to *format are specified in the Ursula ADFS Functional Specification, which is yet to be written.  L format will also be removed from the options.

Similarly, the new formats will be added to the ADFSFiler format menu, and L format will be removed from that menu.

# 5.0 Programming Interface

No new FileCore SWIs are introduced.  Note that some FileCore SWIs take a disc record as an argument, and some return a disc record; the disc record structure passed or returned is now as defined in section 7.4 of this document.

# 6.0 Data Interchange

No new data flows are introduced.

# 7.0 Data Formats

## 7.1 Directory Entry Format

Currently, directory entries are fixed length.  Under the new directory format, directory entries will be variable length. Directory entries will consist of a fixed-length part giving the length of the directory entry, and a variable-length part, giving the name of the object.

The current directory entry is fixed length (26 bytes):

| Name | Bytes | Offset | Purpose |
|---|---|---|---|
| DirObName | 10 | 0 | Name of object |
| DirLoad | 4 | 10 | Load address of object |
| DirExec | 4 | 14 | Exec address of object |
| DirLen | 4 | 18 | Length of object |
| DirIndDiscAdd | 3 | 22 | Indirect disc address of object |
| NewDirAtts | 1 | 25 | Attributes of object |

Note that, since the directory entry is 26 bytes long, directory entries are not always word aligned, and word-length field within the directory entry are also not word aligned. This reduces efficiency since we must perform non word aligned accesses to read or write the fields. Thus, the new directory entry format will always result in a word-multiple length for the directory entry, and the fields will all be word-aligned.

Unlike the current directory format, the list of entries has no terminating 0 byte. To determine where the list ends, the field BigDirDataSize in the directory header is used.

The new format is as follows:

| Name | Bytes | Offset | Purpose |
|---|---|---|---|
| BigDirLoad | 4 | 0 | Load address of object |
| BigDirExec | 4 | 4 | Exec address of object |
| BigDirLen | 4 | 8 | Length of object |
| BigDirIndDiscAdd | 4 | 12 | Indirect disc address of object |
| BigDirAtts | 4 | 16 | Attributes of object |
| BigDirObNameLen | 4 | 20 | Length of object name |
| BigDirObNamePtr | 4 | 24 | Offset into name heap for name of this object |

The backup directory entries are identical to this.

## 7.2 Name Heap Entries

Name heap entries have the following format:

| Bytes | Offset | Purpose |
|---|---|---|
| BigDirObNameLen+1 | 0 | Name of file plus CR |
| 0-3 | BigDirObNameLen+1 | Padding (0 bytes) |

## 7.3 Directory Header

The current directory header is as follows:

| Name | Bytes | Offset | Purpose |
|---|---|---|---|
| StartMasSeq | 1 | 0 | Sequence number. |
| StartName | 4 | 1 | 'Hugo' or 'Nick' |

The old header is not a word multiple length. For efficiency, the new header will be of word multiple length. Note that the directory header is now variable length to allow for variation in the length of the directory's name. The new header is 24 bytes, plus the number of bytes required for the directory name, as follows:

| Name | Bytes | Offset | Purpose |
|---|---|---|---|
| StartMasSeq | 1 | 0 | Sequence number. |
| BigDirVersion | 3 | 1 | Version number (reserved, must be 0) |
| BigDirStartName | 4 | 4 | 4 characters "SBPr" |
| BigDirNameLen | 4 | 8 | Length of directory name |
| BigDirSize | 4 | 12 | Length of directory, in bytes |
| BigDirEntries | 4 | 16 | Number of entries in directory |
| BigDirNamesSize | 4 | 20 | Number of bytes allocated for names |
| BigDirParent | 4 | 24 | Indirect disc address of parent dir |
| BigDirName | BigDirNameLen+1 | 28 | Name of directory, plus CR terminator. Null padded to word boundary. |

## 7.4 Directory Tail

The current directory tail is as follows:

| Name | Bytes | Offset | Meaning |
|---|---|---|---|
| NewDirLastMark | 1 | 0 | 0 to indicate end of entries |
| Reserved | 2 | 1 | Reserved - must be zero |
| NewDirParent | 3 | 3 | Indirect disc address of parent directory |
| NewDirTitle | 19 | 6 | Directory title |
| NewDirName | 10 | 25 | Directory name |
| EndMasSeq | 1 | 35 | To match with StartMasSeq |
| EndName | 4 | 36 | 4 characters, 'Hugo' or 'Nick' to match StartName |
| DirCheckByte | 1 | 40 | Check byte on directory |

The new directory tail is 8 bytes, as follows:

| Name | Bytes | Offset | Meaning |
|---|---|---|---|
| BigDirEndName | 4 | 0 | 4 characters, 'oven' |
| BigDirEndMasSeq | 1 | 4 | To match with StartMasSeq |
| Reserved | 2 | 5 | Must be zero |
| BigDirCheckByte | 1 | 7 | Check byte on directory |

## 7.5 Disc Record

The disc record format is changed so that it indicates whether large directories are used on this disc.  The change is to the meaning of bit 1 of byte 41 of the disc record.  The change is underlined for clarity.

| offset | name | meaning | |
|---|---|---|---|
| 0 | *log2secsize* | $\log_2$ (sector size of disc in bytes) | |
| 1 | *secspertrack* | Number of sectors per track | |
| 2 | *heads* | Number of disc heads if sides interleaved | |
| 3 | *density* | **value** | **meaning** |
| | | 0 | hard disc |
| | | 1 | single density |
| | | 2 | double density |
| | | 3 | double+ density |
| | | 4 | quad density |
| | | 8 | octal density |
| 4 | *idlen* | Length of id field of a map fragment, in bits | |
| 5 | *log2bpmb* | $\mathrm{Log}_2$ (number of bytes per map bit) | |
| 6 | *skew* | Track to track sector skew for random access file allocation | |
| 7 | *bootoption* | Boot option (as *Opt 4,n) | |
| 8 | *lowsector* | **bit** | **meaning** |
| | | 0-5 | lowest numbered sector id on a track |
| | | 6 | if set, treat sides as sequenced (rather than interleaved) |
| | | 7 | if set, disc is 40 track |
| 9 | *nzones* | least sig. byte of number of zones in the map | |
| 10 | *zone_spare* | Numer of non-allocation bits between zones | |
| 12 | *root* | Disc address of root directory | |
| 16 | *disc_size* | least sig. 32 bits of disc size, in bytes | |
| 20 | *disc_id* | disc cycle id | |
| 22 | *disc_name* | Disc name | |
| 32 | *disctype* | File type given to disc | |
| 36 | *disc_size_2* | most sig. 32 bits of disc size, in bytes | |
| 40 | *share_size* | $\log_2$ (sharing granularity in sectors) | |
| 41 | *big_flag* | **bit** | **meaning** |
| | | 0 | if set, then RISC OS partition >512MB |
| | | 1-7 | reserved, must be 0 |
| 42 | *nzones_2* | most sig. byte of number of zones in the map | |
| 43 | | reserved, must be zero | |
| 44 | *format_version* | Version number of disc format | |
| 48 | *root_size* | Size of root directory | |
| 52-59 | | Reserved; must be 0 | |

# 8.0 Dependencies

Disc tools (such as !HForm) must be updated to cope with the changes specified here. These changes are specified in a separate disc tools functional specification.

ADFS requires modification to allow specification of variable length directories when formatting floppies; ADFSFiler requires similar modification.

# 14.0 Glossary

| | |
|---|---|
| **D-format** | Arthur introduced this disc format. The format has 77 entry directories (ie the same size as for E-Format) but still uses the old format free space map. |
| **E-format** | E-format discs have 77 entry directories, and a new format free space map. They can either be floppies which are 800K or hard discs which are variable size. |
| **F-format** | This is a 1600K floppy disc. Its structure is like that of an E-format hard disc. |
| **L-format** | 640K format, from the BBC Master series ADFS. 47 entry directories, and old format free space map. |
| **new map** | The free space map format on E and F format discs. This free space map allows fragmented files, and performs auto-defragmentation when needed. |
| **old map** | This format doesn't support fragmentation of files. Often, it is necessary to compact the disc (with *compact) before a file may be extended. |

# 15.0 References


1. *RISC OS 3 Programmer's Reference Manual, Vol. 5a*, pp 5a-165 to 5a-168, Acorn Computers Ltd, 1995
2. *RISC OS 3 Programmer's Reference Manual, Vol. 2*, pp 2-199 to 2-209, Acorn Computers Ltd, 1992
3. *FileCore - Phase 1 Functional Specification* (1309,207/FS) Acorn Computers Ltd, 1997


# 16.0 History

| Revision | Who | Date | Comment |
|---|---|---|---|
| 0.00 | SBP | 30-07-1997 | Main content complete. |
| 0.01 | SBP | 13-08-1997 | Added development test strategy, acceptance test, references, directory check byte calculation and root directory disc address information. |

| | | | |
|---|---|---|---|
| 0.02 | SBP | 22-08-1997 | Updated from review of specification version 0.01. |
| 0.03 | SBP | 29-08-1997 | Fixed error in directory header format. |
| 0.04 | SBP | 18-11-1997 | Changed dir format to make it more efficient. |
| 0.05 | SBP | 06-01-1998 | Reduced size of backup dir entries to 1 word. Added root_size field to disc record. Added details of memory allocation for free space maps. |