DANOSI

RC5000 File System

File System Nucleus

Programmer's Reference Manual

Keywords: Filesystem nucleus, Winchester disk, RC5000 Real-Time Pascal.

Abstract: This is the reference manual for the RC-FS (File System) nucleus as implemented on the RC5000 Network Switch.

Date: July, 1985

Author: Jens Kristian Kjærgård

PN: 99000761

TABLE OF CONTENTS PAGE

TABLE OF CONTENTS (continued) PAGE

1. INTRODUCTION

This manual describes the file system nucleus RC-FS implemented in Real-Time Pasçal to the RC3502 computer for backing storage support.

The backing storage may consist of several - possibly different - backing storage devices, e.g. hard disks and floppy disks.

The current realization is based upon 10 Mbytes winchester disk drives connected to the SCSI-bus via a DTC510 disk controller (ref. 3). This controller operates as a target on the SCSI-bus. The RC3502 is connected to this SCSI-bus either through the backplane adapter SAD201 or the I/O-channel adapter SAI201.

Chapter 2 of this manual describes the basic concepts used in the file system nucleus.

Chapter 3 describes the routine interface to the file system nucleus.

Chapter 4 describes how the RC-FS process hierarchy is configurated.

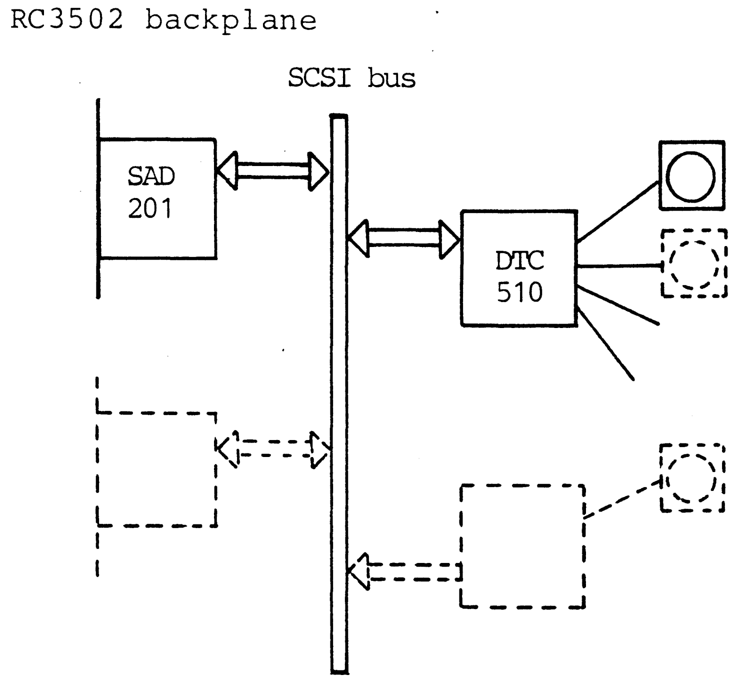

Fig. 1. Backplane connection

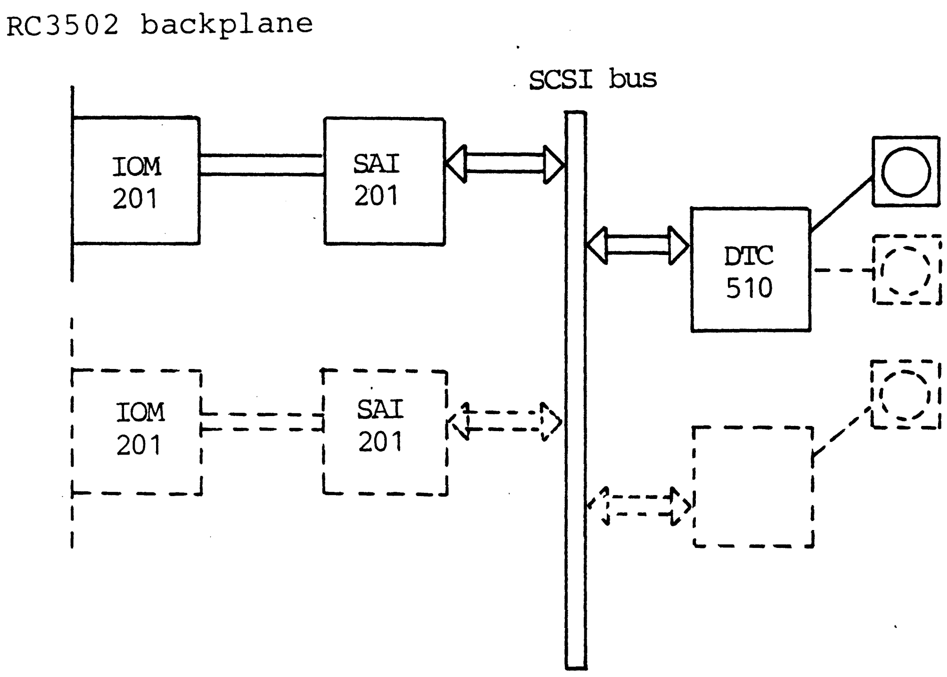

Fig. 2. I/O-adapter connection

2. COMPONENTS OF THE BACKING STORAGE 2.

2.1 Volumes 2.1

The backing storage of a computer system consists of one or more volumes. Each volume resides on a device. A given volume is not necessarily tied to one particular device: Volumes may be exchanged or replaced.

Each volume has a volumename, given to it by the initialization program formatting the volume. It is the responsibility of the user to ensure that volumenames are unique.

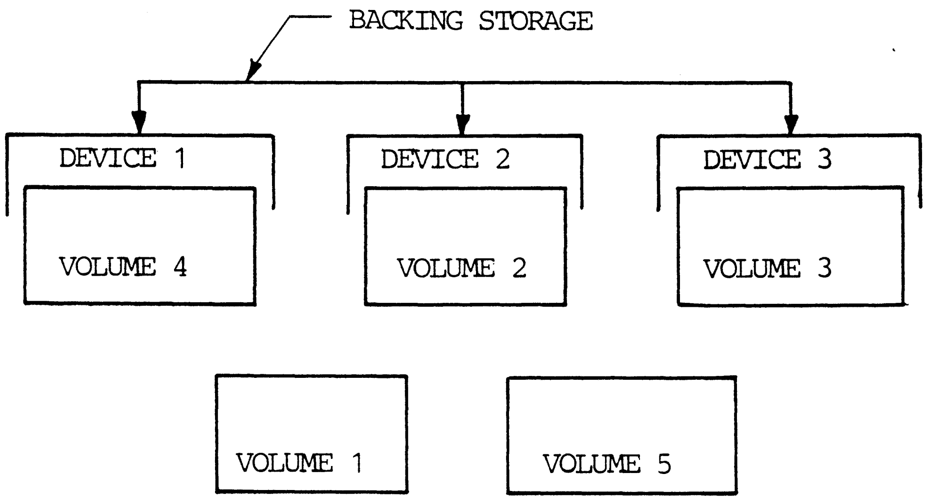

Fig. 3. Example

Volume 2, 3, and 4 are mounted on the devices 2, 3, and 1 respectively. The volumes 1 and 5 are not mounted (dismounted).

2.2 Pages 2.2

A volume consists of a number of pages. One page contains 512 bytes (octets). Pages may be allocated or free (i.e. not-in-use).

Free pages are not allocated individually: The smallest unit of allocation is a slice (consisting of a number of pages). The slice-size may vary from volume to volume.

The free slices of a volume are administered by the Bit Table, in which every slice of the volume is represented by one bit (a slice is free if the corresponding bit is set).

2.3 Files 2.3

Each volume contains a number of files.

A file is the basic facility for permanent storage of data.

A file consists of a (logical) sequence of pages. The individual pages (or rather: slices) of a file need not necessarily be placed contiguously on the volume; they may be scattered in chunks of slices.

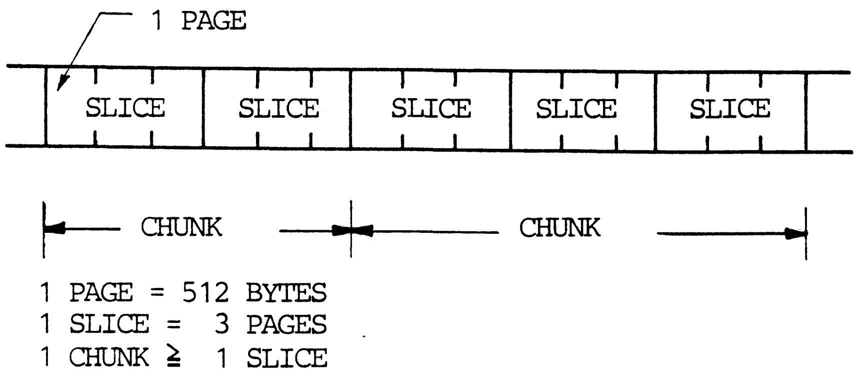

Fig. 4. Example

The administrative data concerning the whereabouts of the chunks of a file are placed on the first page of the file. A file may comprise a maximum of 25 chunks of arbitrary size. A file may hold all the pages of a volume.

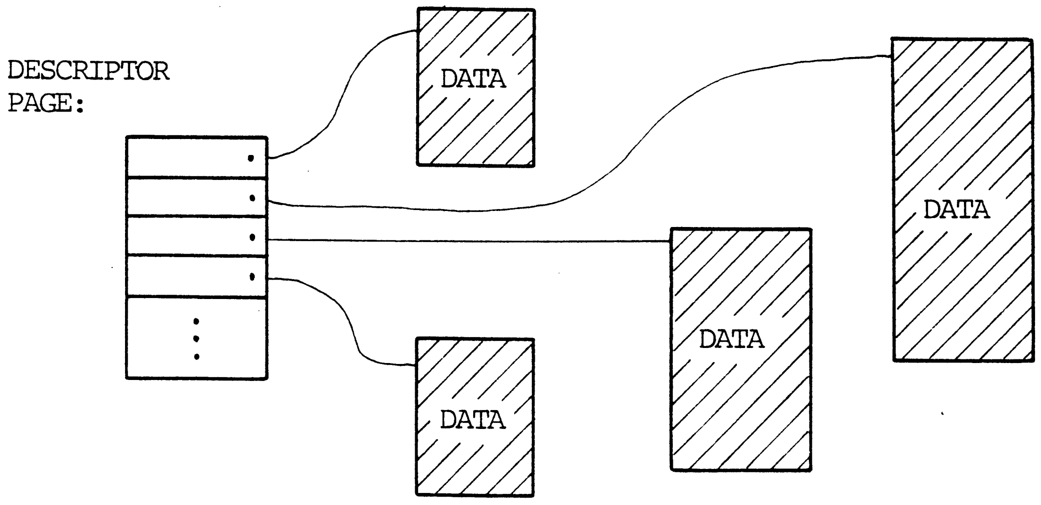

Fig. 5. File organization

The first page moreover contains the <u>descriptor</u> of the file. The descriptor describes the status of the file in terms of length, reservation locks, etc.

The first page of a file is called the <u>descriptor page</u> of the file.

The locations of files are administered by the <u>File Location Table</u> (FLT). This table contains the address of every file (i.e. the address of its first page). Not two table entries point to the same file; that is: The table index of a given file is unambiguous within the volume. Therefore the FLT-index of a file together with the corresponding volumename is used as a unique internal name for that file.

The pair of data (volumename, FLT-index) is called a <u>file identifier.</u>

2.4 Catalogs

Users must be able to associate symbolic names with files.

Symbolic names - together with the corresponding file identifiers - are stored as entries in special files: catalogs.

There may be several catalogs on a single volume.

An entry may reference a catalog, since entries contain file identifiers, and catalogs are files themselves. This facilitates hierarchical (or rather graph-represented) naming systems.

A single file identifier value may be contained in different entries. An entry may contain a file identifier of a file on another volume.

A special catalog exists on each volume: the Rootcatalog. This catalog is the "root" of the hierarchy of catalogs on that volume (i.e. every catalog may be reached, starting at the Rootcatalog).

Whenever a new volume is mounted, the File System delivers a reference to the corresponding Rootcatalog.

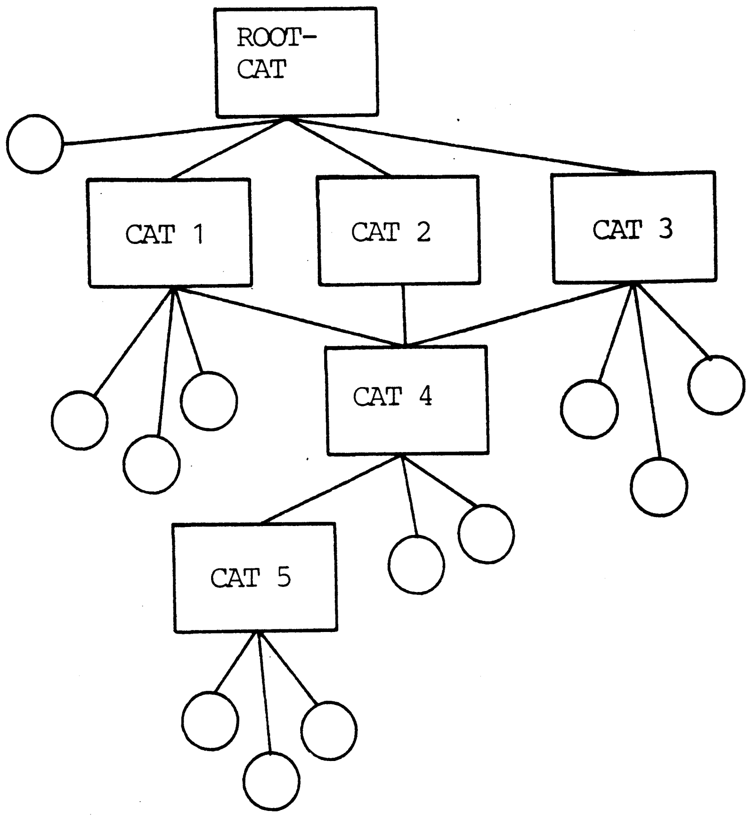

Fig. 6. Catalog organization

2.5 Volumedescriptions 2.5

Every volume contains a special system file: the Volumedescription. This file holds data concerning the status of the volume (slice, size, volumename) etc.

Furthermore it contains the Bit Table and the File Location Table.

To be able to interpret the data structures of an arbitrary volume, the following convention regarding the Volumedescription file is made: The descriptor page of the Volumedescription file is always placed as the first page of the volume.

This means that the Volumedescription file always can be read without knowledge of the exact locations of File Location Table, Rootcatalog, etc. It is the responsibility of the data in the volumedescription file to point out such data structures.

The volumedescription file contains an extra bit table - besides the Bit Table described in subsection 2.2. This bit table is called the Malfunctions Bit Table (MBT), and it specifies all slices in which persistent errors have been detected.

3. FILE SYSTEM INTERFACE 3.

This chapter describes the routine interface to the RC-FS nucleus.

3.1 Constant and Types 3.1

3.1.1 Buffer and Page Definition 3.1.1

```
TYPE
  fs_pageheader_type =
  RECORD
    ?, ?: integer;  (* page address *)
    ?   : integer;  (* page count   *)
    ?   : integer;  (* reserved     *)
  END;

CONST
  fs_pageheader_size =   8;  (* bytes *)

CONST
  fs_pagesize        = 512;  (* bytes *)

TYPE
  fs_pagedata_type = ARRAY (1..fs_pagesize) OF byte;

  fs_page_type =
  RECORD
    page_head: fs_pageheader_type;
    data:      fs_pagedata_type;
  END;

  fs_pagebuffer_type =
  RECORD
    first,last,next: integer;
    page_head:       fs_pageheader_type;
    data:            fs_pagedata_type;
  END;
```

3.1.2 Other Constants and Types 3.1.2

```
CONST
  fs_identifier_size = 16;

TYPE
  fs_identifier = ARRAY (1..fs_identifier_size) OF char;
```

```
CONST
  fs_noname = fs_identifier (fs_identifier_size *** " ");

TYPE
  fs_authorities = (fs_retrieve_authority,
                    fs_update_authority,
                    fs_adjust_authority,
                    fs_delete_authority,
                    fs_authorize_authority);
  fs_authority_type =
    RECORD
      authorizations,
      inheritable,
      protections: SET OF fs_authorities;
    END;

CONST
  fs_maxauthorities = (.fs_retrieve_authority,
                        fs_update_authority,
                        fs_adjust_authority,
                        fs_delete_authority,
                        fs_authorize_authority.);
  fs_maxauthority =
    fs_authority_type (fs_maxauthorities,
                       fs_maxauthorities,
                       fs_maxauthorities);

TYPE
  fs_accessmode = (fs_retrieve, fs_update, fs_exclusive);

TYPE
  fs_entrytype_type = (fs_datafile, fs_catalog);

CONST
  fs_arg_size = 37;
```

3.1.3 Tail Types 3.1.3

```
TYPE
  fs_time =
  RECORD
    date: coded_date;
    time: coded_time;
  END;

CONST
  fs_file_tail_size = 8;
TYPE
  fs_file_tail_type = ARRAY(1..fs_file_tail_size) OF byte;
  fs_file_info_type =
  RECORD
    tail:          fs_file_tail_type;
    written:      !integer;
    allocated:    !integer;
    creation:     !fs_time;
    last_update:  !fs_time;
  END;

CONST
  fs_entry_tail_size = 5;
TYPE
  fs_entry_tail_type = ARRAY(1..fs_entry_tail_size) OF byte;
  fs_entry_info_type =
  RECORD
    name:         !fs_identifier;
    authority:    !fs_authority_type;
    ?, ?:          integer;
    entrytype:    !fs_entrytype_type;
    tail:          fs_entry_tail_type;
  END;
```

3.1.4 Connection Types 3.1.4

```
TYPE
  fs_file_connection =
    RECORD
      ?: reference;  (* nil if file not opened *)
      ?: reference;
      ?: pool 1;
      ?: semaphore
    END;
  fs_catalog_connection =
    RECORD
      ?: ↑semaphore;  (* ptr to the LFS input semaphore *)
      ?: reference;  (* nil if catalog not connected *)
      ?: reference;
      ?: pool 1 OF ARRAY (1..fs_arg_size) OF byte;
      ?: semaphore
    END;
```

3.2 Volume Handling 3.2

Volumes may be mounted or dismounted:

```
     MOUNT (                                 )
     DISMOUNT (      )
```

Volumes must be mounted before any operation can be performed.

3.2.1 Mount 3.2.1

```
PROCEDURE fs_mount
  (    device: byte;
       volumename: fs_identifier;
       mode: SET OF fs_accessmode;
   VAR rootcat: fs_catalog_connection;
   VAR result: byte);
EXTERNAL;
```

Function:

Mounts a volume upon the device. If exclusive access to the volume is needed, ´fs_exclusive´ must be included in ´mode´.

Parameters:

´device´	The number of the physical device upon which the volume is to be mounted.
´volumename´	The name of the volume. This name is checked against the name in the volume descriptor file.
´mode´	The access rights requested for the rootcatalog.
´rootcat´	The resulting catalog connection for the root catalog.

3.2.2 Dismount 3.2.2

```
PROCEDURE fs_dismount
  (VAR cat: fs_catalog_connection;
   VAR result: byte);
EXTERNAL;
```

Function:

Dismounts a mounted volume.

Parameters:

´cat´	The catalog connection of the root catalog.

3.3 Catalog Handling

A catalog must be connected ("opened") before it can be processed. When a user connects a catalog, he will be supplied with a shorthand reference to this catalog. In fact this is what happens in the function MOUNT of the previous subsection: When a volume is MOUNTed, the root catalog will automatically be connected and a shorthand reference delivered.

A user only knows catalogs through symbolic names (which in their place are found in another catalog). Whenever a (simple) name is specified, it must be accompanied by a reference to the catalog in which to find the name.

```
CONNECT (                    )
```

connects a new catalog. The user describes the catalog by a symbolic name and a catalog reference in which to find the name. Upon return the catalog will be opened.

When the catalog has been processsed it may be "closed" by means of:

```
DISCONNECT (      )
```

A specific name contained in the catalog may be found by means of:

```
LOOKUP (                            )
```

This function returns the attributes of the found name, and the descriptor of the file.

3.3.1 Connect

```
PROCEDURE fs_connect
  (VAR cat: fs_catalog_connection;
       name: fs_identifier;
       mode: SET OF fs_accessmode;
   VAR newcat: fs_catalog_connection;
   VAR result: byte);
EXTERNAL
```

Function:

Retrieves a catalog connection.

Parameters:

´cat´	The connection to the catalog in which the new catalog is to be found.
´name´	The name of the new catalog which is to be connected.
´mode´	The access rights requested for the new catalog.
´newcat´	The resulting catalog connection.

3.3.2 Disconnect 3.3.2

```
PROCEDURE fs_disconnect
  (VAR cat: fs_catalog_connection;
   VAR result: byte);
EXTERNAL
```

Function:

Releases a catalog connection.

Parameters:

´cat´	The catalog connection to be released.

3.3.3 Lookup 3.3.3

```
PROCEDURE fs_lookup
  (VAR cat: fs_catalog_connection;
       name: fs_identifier;
   VAR entry: fs_entry_info_type;
   VAR descriptor: fs_file_info_type;
   VAR result: byte);
```

Function:

Retrieves the catalog information of the entry and important parts of the descriptor page. The referred file is temporarily opened.

<u>Parameters:</u>

´cat´	The connection to the catalog in which the entry is to be found.
´name´	The name of the file to be looked up.
´entry´	The catalog information of the entry.

This record contains the following information:

´name´	The name of the entry.
´authority´	The authorities of the entry.
´entrytype´	The type of the entry, i.e. data or catalog.
´tail´	Further entry information.

´descriptor´	The following information of the file referred by the catalog entry.

´tail´	The descriptor page tail.
´written´	The page number of the next page to be written.
´allocated´	The number of pages allocated for the file including the descriptor page. E.g. if allocated = 3, the file contains a descriptor page and 2 data pages numbered as page number 1 and 2.
´creation´	The creation time of the file.
´last_update´	The time of the last_update of the file.

3.4 File Handling

When a file is to be processed, it must be opened. After processing it may be closed (to release tied-up resources).

```
OPEN (                          )
CLOSE (       )
```

Open retrieves a file connection to be used when accessing the file.

The contents of the file may be processed by:

```
GET (                           )
PUT (                           )
```

which transfer a number of pages, the first page to be transferred to or from the file, and the number of subsequent pages.

New files may be created:

```
CREATE (                  )
```

An entry is inserted into the catalog and the file is created with initialization data (for attributes and descriptors). The file may be a catalog.

A file is deleted by:

```
DELETE (           )
```

Deletes the designated entry, and when the last entry referring the file is deleted, the file is deleted.

A catalog-file cannot be deleted, unless it is empty.

A new entry may be linked to an already existing file by means of:

```
LINK (                          )
```

A new simple name is inserted into the catalog. The entry may reference a file from another catalog.

The name of an entry may be changed (within a catalog) by means of:

```
RENAME (              )
```

Finally a file may be extended or truncated:

```
     EXTEND (            )
     TRUNCATE (            )
```

Finally - as a service function to augmenting file system levels - the following function allows arbitrary data to be written into the descriptor of a file:

```
     SET-TAIL (            )
```

3.4.1 Open 3.4.1

```
PROCEDURE fs_open
  (VAR cat: fs_catalog_connection;
       name: fs_identifier;
       mode: SET OF fs_accessmode;
   VAR FILE: fs_file_connection;
   VAR result: byte);
EXTERNAL;
```

Function:

Retrieves a file connection.

Parameters:

´cat´	The connection to the catalog in which the file is searched for.
´name´	The name of the file to be opened.
´mode´	The requested access rights.
´file´	The resulting file connection.

3.4.2 Close 3.4.2

```
PROCEDURE fs_close
  (VAR: fs_file_connection;
   VAR result: byte);
EXTERNAL;
```

Function:

Closes a file and releases all resources.

Parameters:

´file´ The file connection to be closed.

3.4.3 Get <span>3.4.3</span>

```
PROCEDURE fs_get
  (VAR file: fs_file_connection;
       pageno: integer;
       amount: integer;
   VAR buffer: reference;
   VAR result: byte);
EXTERNAL;
```

Function:

Reads amount number of pages from ´file´ starting at ´pageno´. The pages in a file are numbered 1..n. A page must be written before it can be read.

Parameters:

´file´	The file connection of the file from which the data shall be read.
´pageno´	The file relative page address of the first page to be read.
´amount´	The number of pages to be read.
´buffer´	The message to receive the contents of the pages.

´buffer´ must be of type:

```
RECORD
  first, last, next: integer
  header: fs_page_header;
  ?: ARRAY (6 + fs_page_headersize .. first-1)
                                       of byte;
  data: ARRAY (first .. last) of byte;
END;
```

where:

```
  6 + fs_page_headersize <= first <= last

  first + amount * fs_pagesize <= last
          <= size * 2-1 <= maxint
```

After the operation, next points to the

first unused byte in the buffer.

The contents of header is changed by the routine.

3.4.4 Put

```
PROCEDURE fs_put
  (VAR file: fs_file_connection;
       pageno: integer;
       amount: integer;
   VAR buffer: reference;
   VAR result: byte);
EXTYERNAL;
```

Function:

Writes amount number of pages to ´file´ starting at ´pageno´.

The pages in a file are numbered 1..n. Sufficient pages must be assigned to ´file´ beforehand.

The first write to page n must be performed before the first write to page n+1.

Parameters:

´file´	The file connection of the file to which the data shall be written.
´pageno´	The file relative page address of the first page to be written.
´amount´	The number of pages to be written.
´buffer´	The message containing the data to be written.

´buffer´ must be of type:

```
RECORD
  first, last, next: integer;
  header: fs_page_header;
  ?: array (6 + fs_page_headersize .. first-1)
                                     of byte;
  data: array (first .. last) of byte
END;
```

where:

```
  6 +  fs_page_headersize <= first <= last

  first + amount * fs_pagesize <= last
          <= size * 2-1 <= maxint
```

After the operation, next points to the first unused byte in the buffer.

The contents of header is changed by the routine.

3.4.5 Create

```
PROCEDURE fs_create
  (VAR cat: fs_catalog_connection;
       name: fs_identifier;
   VAR amount: integer;
       authority: fs_authority_type;
       entrytype: fs_entrytype_type;
   VAR result: byte);
EXTERNAL;
```

Function:

Creates a new file and inserts an entry in the specified catalog. The file is allocated amount pages.

Parameters:

´cat´	The connection to the catalog in which the entry is inserted.
´name´	The name of the new file.
´amount´	At entry: The number of requested pages for the file. At exit: The total number of pages occupied by the file.
´authority´	The authorities to be given to the file in cat.

´entrytype´	Specifies whether the file is a catalog or a datafile.

3.4.6 Delete 3.4.6

```
PROCEDURE fs_delete
  (VAR cat: fs_catalog_connection;
       name: fs_identifier;
   VAR amount: integer;
   VAR result: byte);
EXTERNAL;
```

Function:

Deletes the entry from the catalog. Delete access is required, and no connection to the entry must be open.

Deletion of the file is postponed until all entries referring the file are deleted.

Parameters:

´cat´	The connection to the catalog containing the entry to be deleted.
´name´	The name of the entry to be deleted.
´amount´	The number of pages which is or is going to be deallocated.

3.4.7 Link 3.4.7

```
PROCEDURE fs_link
  (VAR cat: fs_catalog_connection;
       name: fs_identifier;
   VAR oldcatalog: fs_catalog_connection;
       oldname: fs_identifier;
   VAR result: byte);
EXTERNAL;
```

Function:

Inserts a link into the catalog. A link is an entry referring the same file as the entry ´oldname´ in the catalog ´oldcatalog´.

Parameters:

´cat´	The connection to the catalog in which the link is inserted.
´name´	The name of the link.
´oldcatalog´	The connection to the catalog containing the entry referring the file to which the link is going to refer.
´oldname´	The name of the entry referring the file to which the link is going to refer.

3.4.8 Rename

```
PROCEDURE fs_rename
  (VAR cat: fs_catalog_connection;
       name: fs_identifier;
       oldname: fs_identifier;
   VAR result: byte);
EXTERNAL;
```

Function:

Changes the name of the entry in the catalog.

Parameters:

´cat´	The connection to the catalog in which the entry is contained.
´name´	The new name of the entry.
´oldname´	The old name of the entry.

3.4.9 Extend

```
PROCEDURE fs_extend
  (VAR cat: fs_catalog_connection;
       name: fs_identifier;
   VAR amount: integer;
   VAR result: byte);
EXTERNAL;
```

Function:

Assigns additional pages to a file.

Parameters:

´cat´	The connection to the catalog referring the file.
´name´	The name of the file.
´amount´	At entry: The additional number of pages to be allocated. At exit: The total number of allocated pages for the file.

3.4.10 Truncate

```
PROCEDURE fs_truncate
  (VAR cat: fs_catalog_connection;
       name: fs_identifier;
   VAR amount: integer;
   VAR result: byte);
EXTERNAL;
```

Function:

Releases pages from the file.

Parameters:

´cat´	The connection to the catalog referring the file.
´name´	The name of the file.
´amount´	At entry: The number of pages to be released. At exit: The total number of allocated pages for the file.

3.4.11 Set tail

```
PROCEDURE fs_set_tail
  (VAR cat: fs_catalog_connection;
       name: fs_identifier;
       entry: fs_entry_info_type;
       descriptor: fs_file_info_type;
   VAR result: byte);
```

Function:

Updates the tail information in the catalog entry and the descriptor page of the file. The referred file is temporarily opened.

Parameters:

´cat´	The connection to the catalog in which the entry is to be found.
´name´	The name of the entry to be updated.
´entry´	Contains the new entry tail. Only the tail field is used.
´descriptor´	Contains the new file tail. Only the tail field is used.

3.5 **Authorization Enforcement**

The first level of authorization enforcement is the naming system: users can only access the files they can name (if a user only knows one catalog reference, he can only access the subtree (or rather: subgraph) of files of which this catalog is a root).

The second level of authorization enforcement concerns the individual symbolic names of files: Associated with each simple name is a set of authorization-attributes. The total set comprises:

Authorization to:

- RETRIEVE (data)
- UPDATE (data)
- ADJUST (length of file)
- DELETE (file)
- AUTHORIZE (i.e. redefine the set of authorizations)

The set of authorizations for a given name defines the allowed use of the corresponding file.

When a file (possibly catalog) "FILE" (say) is OPENed (CONNECTED), the set of authorizations associated with the file is computed as the intersection of the two sets associated with (1) the entry "FILE" and (2) the catalog in which the entry "FILE" was found.

If AUTHORIZE is contained in the set of authorizations for a name, the authorizations may be redefined by:

```
     SET-AUTHORITY (             )
```

When a new entry is linked (by LINK) to an existing entry, it will inherit a <u>subset</u> of the authorizations of the existing entry. This subset of "inheritable" authorizations may for a given entry be defined by:

```
     SET-INHERITANCE (               )
```

The <u>third level</u> of authorization enforcement is totally local to each entry. As a means to protect an entry against accidental misuse, a set of <u>protections</u> is associated with each entry. The set of protections comprises exactly the same elements as the total set of authorizations (viz. RETRIEVE, UPDATE, ADJUST, DELETE, AUTHORIZE). Whenever a function is performed via a given entry, it is not only tested against the authorization of the entry, but against the intersection of both authorizations and protections.

The difference between authorizations and protections is twofold:

- Protections have no influence whatsoever on inheritance of authorizations.

- Protections may be redefined unconditionally at any time.

Thus, protections are a temporary guard against accidents.

The protections of a given entry is redefined by:

SET-PROTECTION (   )

3.5.1 Set Authority 3.5.1

```
PROCEDURE fs_set_authority
  (VAR cat: fs_catalog_connection;
       name: fs_identifier;
       authorization: SET OF fs_authorities;
   VAR result: byte;
EXTERNAL;
```

Function:

Updates the authority attribute of the entry.

Parameters:

´cat´	The connection to the catalog containing the entry.
´name´	The name of the entry.
´authorization´	The new value of the attribute.

3.5.2 Set Inheritance 3.5.2

```
PROCEDURE fs_set_inheritance
  (VAR cat: fs_catalog_connection;
       name: fs_identifier;
       inheritable: SET OF fs_authorities
   VAR result: byte);
EXTERNAL;
```

Function:

Updates the inheritable attribute of the entry.

Parameters:

´cat´	The connection to the catalog containing the entry.
´name´	The name of the entry.
´inheritable´	The new value of the attribute.

3.5.3 Set Protection

```
PROCEDURE fs_set_protection
  (VAR cat: fs_catalog_connection;
       name: fs_identifier;
       protections: SET OF fs_authorities;
   VAR result: byte);
EXTERNAL;
```

Function:

Updates the protection attribute of the entry.

Parameters:

´cat´	The connection to the catalog containing the entry.
´name´	The name of the entry.
´protections´	The new value of the attribute.

3.6 Results

ok

The operation has been correctly executed.

catconnection

The catalog connection is in an improper state. Disconnected when a connected connection is required or vice versa.

fileconnection

The file connection is in an improper state. Closed when an open connection is required or vice versa.

name_inexistant

The specified name does not exist in

	the catalog.
name_exists	The name to be inserted in a catalog already exists in the catalog.
name_error	The specified name is an invalid filename.
unauthorized	The operation violates the authority of the connection(s).
deleted	The file has been deleted.
filelimit	A page outside the accessible range of pages in the file is referred. A page must be written before it can be read. Page n must be written before page n + 1 can be written (n > 0).
buffersize	The first and last fields in the buffer_head define an illegal number of bytes.
catalog_full	No more entries can be inserted in the catalog.
rejected	The operation cannot be performed because a file is opened.
mount_error	A volume is not mounted, or mount is called and a volume is already mounted at the device.
openlimit	Too many files have been opened.
includelimit	Too many open´s to the same file.
indexlimit	The maximum number of checks is ex-

	ceeded during allocation because of fragmentation.
flt_overflow	The maximum number of different files is exceeded.
volumelimit	No more free slices on this volume.
disconnected	The device used for the operation is disconnected, possibly due to hardware problems.
parity	An error is detected during read or write of one of the pages accessed in the operation. The page is marked in the MBT-table.
harderror	The disk controller is unable to perform the requested operation.
version	The volume is formatted with an illegal version of the FS nucleus.
not_implemented	This operation is not implemented.
software	An inconsistency in the FS nucleus is detected.

4. FILE SYSTEM MASTER PROCESS

The File System Master Process (FS) controls the process incarnations comprising the RC-FS complex.

The RC-FS complex consists of the processes LFS, BFS, FOSM, and DSM from the RC-FS Nucleus, besides a number of driver processes. The driver processes are at present an SAI Driver for the SAI201 interface to the SCSI bus, and a Winchester Disc Subdriver for a Winchester disc connected to the SCSI bus via the DTC510 Disc Controller.

The RC-FS Nucleus communicates with the Winchester Disc Subdriver, which communicates with the Winchester disc via the SAI Driver.

The incarnation structure is shown on fig. 7.

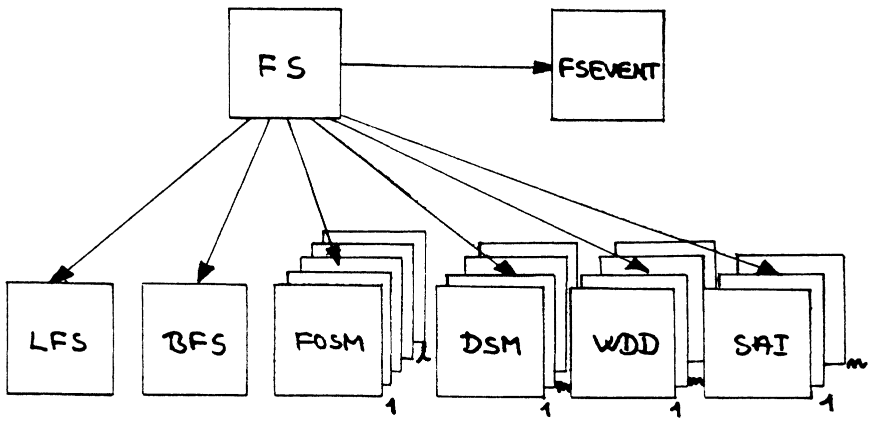

Fig. 7. RC-FS Nucleus Incarnation Structure

In brief the following functions are performed by the FS processes:

LFS	Logical File System transforms a user-supplied symbolic file name to an internal unique file identifier.
BFS	Basic File System finds and maintains the corresponding file descriptor.
FOSM	File Organization Strategy Modules translate the logical page addresses to physical page adresses.
DSM	Device Strategy Modules produce an optimal sequence of requests and administer the allocation and deallocation of pages on the device.
WDD	Winchester Disc Driver executes the operations supported by the DTC510 Disc Controller.
SAI	SAI Driver acts as an SCSI Bus Initiator Control Unit.
FSEVENT	supports event handling from the WDD and SAI.

See section 4.3 for further information.

4.1 File System Configuration 4.1

The RC-FS Nucleus is configured through compiling the FS Process with suitable configuration parameters.

The parameters are collected in the environment ´fsmenvir´. An example of such a configuration is shown in appendix C.

For dimensioning purposes of the RC-FS Nucleus four constants and a ´disc_table´, defining the physical coordinates of the connected Winchester Discs, should be specified by the user.

MAX_FOSM - defines the number of FOSM incarnations. At most MAX_FOSM distinct files (Catalogs or Data Files) may be ´connected´ or ´opened´ simultaneously via the RC-FS Nucleus.

MAX_FOSM_TICKET - defines the maximum number of simultaneous connections to one and the same file.

MAX_DSM - defines the number of DSM (and WDD) incarnations. Actually, MAX_DSM is the number of physical discs connected to this RC3502.

MAX_SAI201 - defines the number of SAI201 interfaces through which MAX_DSM discs are connected.

DISC_TABLE - holds the physical descriptions of all connected discs. One disc is specified by a record in the table.

The first record corresponds to ´device1´, the second record ´device2´, etc. This device number is used in the MOUNT call.

Three coordinates are defined in one record:

SAI201_LEVEL is the interruption level of the data channel of the SAI201 interface to the SCSI bus, where this disc is connected. By convention the control channel of SAI201 is allocated interruption level ´SAI201_LEVEL-1´.

WDD_CU is the SCSI Bus Control Unit address of the DTC510 Disc Controller, where this disc is attached.

WDD_LUN is the Logical Unit Number of this disc at the DTC510 Disc Controller.

4.2 File System Initialization 4.2

The FS Master Process may run as a child of ADAM.

The FS may generate the following messages during initialization:

*** disc table overlap , result = xx

- xx disc address definitions overlap. Redefine the contents of ´disc_table´ in FSMENVIR and recompile FS.

*** name_semaphore <s_name> , result = 2

- No resources, a semaphore cannot be catalogued because of lack of memory resources.

*** adam_name_semaphore <s_name> , result = xx

- ADAM cannot catalogue this semaphore due to name overlap (result = 16) or no resources (result = 17).

*** link <processname> , result = xx

- FS receives result xx from a LINK call.

*** create <incarnationname> , result = xx

- FS receives result xx from a CREATE call.

*** start <incarnationname> , result = xx,yy

- FS receives result u2,u3 = xx,yy from a ´start´ request to the SAI Driver.

  Consult appendix D for further details.

4.3 Events

Three types of events are generated by the RC-FS Nucleus. One event type originates from the SAI Driver and two from the WDD. The event is pushed upon the driver request. The fsevent process interpretes and prints the contents of the events.

4.3.1 sai_event

This event is generated each time the SAI Driver returns a result indicating a transient error (basic result 2).

The event message looks as follows:

u1	sai_event (= 7)
u2	result
u3	cu
u4	unchanged

The databuffer contains an event record of 9 bytes positioned according to the value of first.

The result value of u2 is interpreted as follows:

2 transient error

with the following subfield interpretation:

```
---------------------------
! phase ! PE ! TO !       !
---------------------------
0       2    3    4   5    7
```

+ 0 + phase * 32.

An illegal phase is entered. This is the usual result, if the Winchester disc controller detects an error in a read or write command (the status phase is entered instead of a datdain or dataout phase).

The phases are numbered as follows:

0	selection
1	command
2	datain
3	dataout
4	status
5	message
6	error

+ 8 + phase * 32	timeout
+ 16 + phase * 32	parity error

An sai_event will be followed by an event from the WDD driver.

4.3.2 wdd status event 4.3.2

This event is generated, when the Winchester disc controller has detected an error.

The event message looks as follows:

```
u1   wdd_status_event (= 2)
u2   status
u3   lun * 8 + cu
u4   unchanged
```

The status byte is interpreted as follows:

```
-----------------------
! LUN ! SPARE ! E ! P !
-----------------------
 0   2 3     5   6   7
```

P   Indicates that a parity error occurred during transfer from host to controller.

E   Indicates that an error occurred during the execution of a command.

LUN The logical unit number of the drive where the error occurred.

The data buffer contains four additional status bytes positioned according to the value of first.

```
0:   sense byte
1:   lun * 32 + lad2
2:   lad1
3:   lad0
```

The sense byte describes the details or the nature of an error status. The bits within the sense byte are defined as:

```
---------------------------------
! 0 ! 1 ! 2 ! 3 ! 4 ! 5 ! 6 ! 7 !
---------------------------------
  !   !   !   !   !   !   !   !
  !   !   !   !   ----------------- Error Code
  !   !   ------------------------- Error Type
  !   ----------------------------- Spare
  --------------------------------- LAD Valid
```

The valid LAD bit indicates that the Logical Address, the LAD in bytes 1 through 3, contains the valid address of the block at which the error occurred.

The two error type bits describe the general type of error as follows:

00:	Drive related error
01:	Controller related error
10:	Command related error
11:	Miscellaneous error

The error code bits define the error under each of the four types of errors defined by the error type.

Drive related errors:

0	No Eror Status
1	No Index Signal
2	No Seek Complete
3	Write Fault
4	Drive Not Ready
5	Drive Not Selected
6	No Track 00
7	Multiple Winchester Drives Selected
D	Seek in Progress

Controller related errors:

0	ID Read Error (ECC error in the ID field)
1	Uncorrectable Data Error During a Read
2	ID Address Mark Not Found
3	Data Address Mark Not Found
4	Record Not Found (Found correct cylinder and head but not sector)
5	Seek Error (R/W head positioned on wrong cylinder and/or selected a wrong head)
6	Unused
7	Write Protected
8	Correctable Data Field Error
9	Bad Block Found
A	Format Error (the controller detected during a Check Track command that the format on the drive was not as expected)
C	Unable to Read an Alternate Track Address
E	Attempted to direct access on Alternate Track
F	Sequence Time Out During Disk or Host Transfer

Command related errors:

0	Invalid Command Received from the Host
1	Illegal Disk Address (address beyond the maximum address)
2	Illegal Function for Type of Drive Specified
3	Volume Overflow - Maximum seactor address was passed during a Multiple Sector read or write

Miscellaneous errors:

0	Ram error

4.3.3 wdd error event 4.3.3

This event is generated each time the WDD detects an error in the communication with the Winchester disc controller.

The event message looks as follows:

u1	wdd_error_event (= 3)
u2	result
u3	lun * 8 + CU
u4	unchanged

Result is one of the following:

2 + 8	Parity Error on the SCSI-bus
2 + 16	SCSI-driver error

A. REFERENCES

1. RCSL No. 52-AA1167
   RC3502/2 REAL-TIME PASCAL
   Reference Manual
   Bo Bagger Laursen, October 1983

2. RCSL No. 31-D608
   Design of a Portable
   File System Nucleur
   Peter Mikkelsen, September 1980
   (Internal use only)

3. (Technical manuals
   for SAI201 and
   WDD201/WDD202.
   Titles and RCSL Nos.
   will be supplied later.)

B. fsenvir B.

```
(*****************************************************X
(*
RC File System

Name:    File System ENVIRonment
Author: Peter Mikkelsen /pmik, Jens Kristian Kjærgård / jkk
Date:    1981-01-26, 1984-05-16, 1984-06-04, 1984-07-11
Version: 2.00

Declarations nescessary for an application program using the
file system.
*)

    fsenvir;

      (* error codes *)
CONST
  ok=0;
  fs_catconnection       =  35;    fs_fileconnection    =  36;
  fs_name_inexistant     =   6;    fs_name_exists       =   5;
  fs_name_error          =  12;    fs_unauthorized      =  30;
  fs_deleted             =   7;    fs_filelimit         =  29;
  fs_buffersize          =  21;    fs_catalog_full      =  13;
  fs_rejected            =  28;    fs_mount_error       =   9;

  fs_openlimit           =  17;    fs_includelimit      =  18;
  fs_indexlimit          =  15;    fs_flt_overflow      =   8;
  fs_volumelimit         =  16;

  fs_parity              =   2;    fs_harderror         =   3;
  fs_disconnected        =  25;

  fs_version             =  22;    fs_not_implemented   =  23;
  fs_software            =  14;


TYPE
  fs_pageheader_type =
  RECORD
    ?, ?: integer;  (* page address *)
    ?   : integer;  (* page count   *)
    ?   : integer;  (* reserved     *)
  END;
CONST
  fs_pageheader_size =   8;  (* bytes *)

CONST
```

```
  fs_pagesize        = 512;  (* bytes *)
TYPE
  fs_pagedata_type = ARRAY (1..fs_pagesize) OF byte;

  fs_page_type =
  RECORD
    page_head: fs_pageheader_type;
    data:      fs_pagedata_type;
  END;

  fs_pagebuffer_type =
  RECORD
    first,last,next: integer;
    page_head:       fs_pageheader_type;
    data:            fs_pagedata_type;
  END;

CONST
  fs_identifier_size = 16;
TYPE
  fs_identifier = ARRAY(1..fs_identifier_size) OF char;
CONST
  fs_noname = fs_identifier (fs_identifier_size *** " ");


TYPE
  fs_authorities = (fs_retrieve_authority,
                    fs_update_authority,
                    fs_adjust_authority,
                    fs_delete_authority,
                    fs_authorize_authority);
  fs_authority_type =
  RECORD
    authorizations,
    inheritable,
    protections: SET OF fs_authorities;
  END;
CONST
  fs_maxauthorities= (.fs_retrieve_authority,
                      fs_update_authority,
                      fs_adjust_authority,
                      fs_delete_authority,
                      fs_authorize_authority.);
  fs_maxauthority =
  fs_authority_type (fs_maxauthorities,
                     fs_maxauthorities,
                     fs_maxauthorities);
TYPE
  fs_accessmode = (fs_retrieve,fs_update,fs_exclusive);
```

```
TYPE
  fs_entrytype_type = (fs_datafile, fs_catalog);

TYPE
  fs_time =
  RECORD
    date: coded_date;
    time: coded_time;
  END;

CONST
  fs_file_tail_size = 8;
TYPE
  fs_file_tail_type = ARRAY(1..fs_file_tail_size) OF byte;
  fs_file_info_type =
  RECORD
    tail:          fs_file_tail_type;
    written:      !integer;
    allocated:    !integer;
    creation:     !fs_time;
    last_update:  !fs_time;
  END;

CONST
  fs_entry_info_size = 32;
  fs_entry_tail_size = fs_entry_info_size -
    ( fs_identifier_size + 6 + 2 + 2 + 1 );
TYPE
  fs_entry_tail_type = ARRAY(1..fs_entry_tail_size) OF byte;
  fs_entry_info_type =
  RECORD
    name:         !fs_identifier;
    authority:    !fs_authority_type;
    ?, ?:          integer;
    entrytype:    !fs_entrytype_type;
    tail:          fs_entry_tail_type;
  END;

CONST
  fs_arg_size = 37;

TYPE
  fs_file_connection =
  RECORD
    ?: reference; (* nil if file not opened *)
    ?: reference;
    ?: pool 1;
    ?: semaphore
  END;
  fs_catalog_connection =
RECORD
  ?: semaphore; (* ptr to the LFS input semaphore *)
  ?: reference; (* nil if catalog not connected *)
  ?: reference;
  ?: pool 1 OF ARRAY(1..fs_arg_size) OF byte;
  ?: semaphore
END;

  PROCEDURE fs_mount
    (    device: byte;
    volumename: fs_identifier;
    mode: SET OF fs_accessmode;
    VAR rootcat: fs_catalog_connection;
    VAR result: byte);
  EXTERNAL;

  PROCEDURE fs_dismount
    (VAR cat: fs_catalog_connection;
    VAR result: byte);
  EXTERNAL;

  PROCEDURE fs_connect
    (VAR cat: fs_catalog_connection;
    name: fs_identifier;
    mode: SET OF fs_accessmode;
    VAR newcat: fs_catalog_connection;
    VAR result: byte);
  EXTERNAL;

  PROCEDURE fs_disconnect
    (VAR cat: fs_catalog_connection;
    VAR result: byte);
  EXTERNAL;

  PROCEDURE fs_lookup
    (VAR cat: fs_catalog_connection;
    name: fs_identifier;
    VAR entry: fs_entry_info_type;
    VAR descriptor: fs_file_info_type;
    VAR result: byte);
  EXTERNAL;

  PROCEDURE fs_open
    (VAR cat: fs_catalog_connection;
    name: fs_identifier;
    mode: SET OF fs_accessmode;
    VAR FILE: fs_file_connection;
    VAR result: byte);
  EXTERNAL;
```

```
PROCEDURE fs_close
  (VAR FILE: fs_file_connection;
  VAR result: byte);
EXTERNAL;

PROCEDURE fs_get
  (VAR FILE: fs_file_connection;
  pageno: integer;
  amount: integer;
  VAR buffer: reference;
  VAR result: byte);
EXTERNAL;

PROCEDURE fs_put
  (VAR FILE: fs_file_connection;
  pageno: integer;
  amount: integer;
  VAR buffer: reference;
  VAR result: byte);
EXTERNAL;

PROCEDURE fs_create
  (VAR cat: fs_catalog_connection;
  name: fs_identifier;
  VAR amount: integer;
  authority: fs_authority_type;
  entrytype: fs_entrytype_type;
  VAR result: byte);
EXTERNAL;

PROCEDURE fs_delete
  (VAR cat: fs_catalog_connection;
  name: fs_identifier;
  VAR amount: integer;
  VAR result: byte);
EXTERNAL;

PROCEDURE fs_link
  (VAR cat: fs_catalog_connection;
  name: fs_identifier;
  VAR oldcatalog: fs_catalog_connection;
  oldname: fs_identifier;
  VAR result: byte);
EXTERNAL;

PROCEDURE fs_rename
  (VAR cat: fs_catalog_connection;
  name: fs_identifier;
  oldname: fs_identifier;
  VAR result: byte);
```

```
EXTERNAL;

PROCEDURE fs_extend
  (VAR cat: fs_catalog_connection;
  name: fs_identifier;
  VAR amount: integer;
  VAR result: byte);
EXTERNAL;

PROCEDURE fs_truncate
  (VAR cat: fs_catalog_connection;
  name: fs_identifier;
  VAR amount: integer;
  VAR result: byte);
EXTERNAL;

PROCEDURE fs_set_authority
  (VAR cat: fs_catalog_connection;
  name: fs_identifier;
  authorization: SET OF fs_authorities;
  VAR result: byte);
EXTERNAL;

PROCEDURE fs_set_inheritance
  (VAR cat: fs_catalog_connection;
  name: fs_identifier;
  inheritable: SET OF fs_authorities;
  VAR result: byte);
EXTERNAL;

PROCEDURE fs_set_protection
  (VAR cat: fs_catalog_connection;
  name: fs_identifier;
  protections: SET OF fs_authorities;
  VAR result: byte);
EXTERNAL;

PROCEDURE fs_set_tail
  (VAR cat: fs_catalog_connection;
  name: fs_identifier;
  entry: fs_entry_info_type;
  descriptor: fs_file_info_type;
  VAR result: byte);
EXTERNAL;

  (***** end fsenvir *****)
.
```

C. fsmenvir

```
   fsmenvir;
(*****************************************************X
(*                                                     *)
(*  fsmenvir is used for configuration of the RC-FS Nucleus  *)
(*  Set the parameters according to your installation        *)
(*  and compile the FS process by the call                   *)
(*                                                           *)
(*     bfs = rtp35022 stack.xxx fsenvir fsmenvir tfs         *)
(*                                                           *)
(*****************************************************X

  CONST
   max_fosm          = 8;  (* Number of File Organisation *)
   ;                       (*    Strategy Modules         *)
   max_fosm_ticket   = 2;  (* Max number of simultanous   *)
   ;                       (*    connections to a file    *)
   max_dsm           = 2;  (* Number of Device Strategy   *)
   ;                       (*    Modules = No of discs    *)
   max_sai201        = 1;  (* Number of SAI201 interfaces *)

  TYPE
   disc_address =
   PACKED RECORD
     sai201_level   : 0..127; (* interruption level of      *)
     ;                        (*    sai201 data channel     *)
     wdd_cu         : 0..7;   (* SCSI control unit address  *)
     ;                        (*    of the DTC510 disc      *)
     ;                        (*    controller              *)
     wdd_lun        : 0..1;   (* logical unit number at the *)
     ;                        (*    DTC510 disc controller  *)
   END;

   disc_table_type =
   ARRAY (1..max_dsm) OF disc_address;

  CONST
     (* Note: one saidriver takes level as data level and *)
     (* level-1 as control level.                          *)
     (* ..... So just specify the data level and make      *)
     (* room for the control level                         *)
   disc_table =
     (* each entry defines one disc *)
   disc_table_type (
                  disc_address (85, 0, 0), (* device 1 *)
                  disc_address (85, 0, 1)  (* device 2 *)
   );
   .
```

D. sai start errors D.

u2 = 0 ok

u2 = 3 + 8 * x u3 = 0

Cannot reserve interrupt channels or unable to create internal process. x is result from reservech or create.

u2 = 3 + 8 u3 >= 128

Conmmunication through the I/O channels to the SCSI chip is malfunctioning. The following modifications to u3 = 128 can be found:

+ 1:	Data written through the control channel to the testregister and read through the control channel is incorrect.
+ 2:	Data written through the control channel to the testregister and read through the data channel is incorrect.
+ 4:	Data written through the data channel to the testregister and read through the control channel is incorrect.
+ 8:	Data written through the control channel to an SCSI (NCR 5385) chip register and read through the control channel is incorrect.

u2 = 3 + 8 64 <= u3 < 128

The following modifications to u3 = 64 can be found:

+ 1:	Missing interrupt from SAI.
+ 2:	No Function Complete.
+ 4:	No Self Diagnostic Complete.
+ 8:	Error in data read after turnaround.

u2 = 3 + 8 u3 < 64

u3 contains the 6 least significant bits of diagnostic status register of the SCSI chip.

E. FORMATTING A VOLUME

A volume is formatted by the program ´fsformat´. ´fsformat´ checks the pages on the volume and initialises the volume descriptor file.

´fsformat´ presumes that an FS master process including an entry in ´disc_table´ for the device, on which the volume to be formatted is placed, is started.

´fsformat´ can be executed by the ´opsys´ command:

```
run fsformat
```

The following parameters must be entered as replies to the prompts:

device -

The index in disc_table (device in mount is supplied).

volume -

The name of the volume.

pages -

The number of pages on the volume. If 0 is replied, the ´fsformat´ program will scan the volume with the internal ´check dist´, until an illegal track answer is received.

slice size-

The size of a slice in pages. A volume can contain a maximum of 2048 slices.

files -

The maximum number of distinct files on the volume, i.e. entries in the File Location Table.

root files-

The number of files to be allocated for the root catalog.

check dist-

> The interval between check read of pages. The value 1 means that it is checked that all pages can be read. At least one page on each track should be read.

After the formatting the actual number of pages on the volume is displayed.

F. INDICES F.

F.1 Survey of Figures F.1

F.2 Catchword Index F.2

Regnecentralen a·s

DOKIDENTIFIKATION

Rekvirent MUJ/JLK	Afdeling tt65	Dato 901120	PN: 990-00-261
Varebetegnelse (Titel) RC-FS PROGRAMMERS REF.MAN.			

Anvendes i	Masterformat ☐ A5 [✓ A4] ☐ Andet:	Antal sider incl. indeksblad 60
12 mdr. behov	Klassifikation (Biblio) [✓ A] ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G	Gl. PN:

REKVIRENT

Varens data, kvalitets spec. og tolerancer evt. bilag

Format: [✓ A4] ☐ A5 ☐ Andet: ______

Papir: Kulør: [✓ Hvidt] ☐ Andet: ______

Kvalitet: [✓ Standard (80 g)] ☐ Andet: ______

Indeksblad ☐ Fotosættes (manuskript vedlagt)

☐ Trykkes på karton

Færdiggørelse [✓ Trykkes på begge sider] ☐ Enkeltsidet

[✓ Hulles med 4 huller RC Standard] ☐ Hulles spec: ______

☐ Hæftes ☐ Spiralryg [✓ Indsvejses i plast] ☐ Andet: ______

Fortrykt forside ved A4- format ☐ Regnecentralen ☐ RC Computer

☐ Regnecentralen (Internt brug) ☐ RC Computer (Internal use only)

Andre informationer som er nødvendige for at fremstille varen:

Bilag ☐

PTA

Lagerføringsenhed	Forbrugsenhed	Omsætningsfaktor	Gruppekode
Deponeringskode	Anmærkningskode	Træktid Dage	Hovedproduktgruppe
Indgangskontrol	Leveringslokation		

INDKØB

Leveringstid		Leverandør nr.		Aktuel indkøbspris	Årets standard kostpris
Lev. 1.	Lev. 2.	Lev. 3.	Lev. 4.	Min. ordre	Pakn.størrelse
					Arkiv (Biblio)

RUTE	PTA GLO	DOK	INDKØB	REG
INITIALER		HAKS		
DATO				

PN: 99200394

PN. 99000761

DENNE MANUAL ER I ORIGINALFORM OVERDRAGET TIL:

TRYKKERIET (HAKJ) : X

PTA / BAL. (AJ) :

DATO: 920730

DENNE SIKKERHEDSKOPI MÅ IKKE FJERNES FRA ARKIVET.

llm/DIV/PNtrykkeri.911115 LLM