RC9000-10/RC8000

SW8010 System Utility

User's Guide, Part One

RC Computer

PN 99200389

Keywords:
RC9000-10, RC8000, System Utility, Utility Programs, User's Guide

Abstract:
This manual describes the File Processor, which is the job control program, part of the System Utility package.

Date:
Januar 1989

PN: 991 11263

Table of Contents

1. Introduction

1.1 The File Processor

The File Processor - in the sequel called FP - is a control program which together with the operating system controls the execution of the user's programs and the access to his files.

When an RC9000-10 computer with the basic software installed is ready for use, the system programs are stored partly in memory, partly on the backing storage. The Monitor program and the nucleus of the operating system BOSS are memory resident while the remainder of the programs are stored on the backing storage, usually consisting of one or more disk files. The execution of a job is controlled by commands to two control programs: the operating system and FP. FP may be used in connection with various operating systems - in the sequel we assume that the operating system BOSS is used.

1.2 Files

A file is an unbroken string of data such as a roll of paper tape, one deck of cards, a data area on the backing storage, the data between two tape marks on a magnetic tape reel. A job uses many different files - beside the files containing the input and output data we have the files containing the user's programs and files in the software system (containing compilers, editors etc.).

The files can be divided into different types according to their relation to the job:

Standard files of the job

(1) The job file specifies the tasks of the job. It is entered into the computer as described in ref. (7). The job file contains (except for 'go' jobs) a heading job specification which is interpreted by BOSS. The rest of the job file is forwarded by BOSS to the job as the primary input file.

(2) The current input file is a file from which the job reads commands to FP and various other input. During the job several files may in turn be selected as current input file. At job start the primary input file is selected as current input.

(3) The current output file is a file used for output from the job. During the job several files may in turn be selected as current output file - the file selected at job start is called the primary output file.

(4) Primout is a backing storage area used by BOSS in the spooling of the output printed on the primary output file. After the termination of an on-line job this area is available and contains the data printed on primary output during the job.

In ALGOL/FORTRAN programs the current input and output files are available via the standard zones IN and OUT. (These zones should be used for character input/output only).

System files
A number of files mainly on the backing storage are permanently available to all jobs. These files contain compilers, utility programs and standard library programs.

The paper tape reader, the line printer (and the card reader, if any) are usually considered to contain files, owned by and accessed through BOSS.

Private files
The user's programs and data files may be stored on any media available in the system. The various types of files are described in (7) chapters 5-6.

FP and the utility programs refer to files by means of names. A file name is a small letter followed by at most 10 digits or small letters.

1.3 The FP Command Reading and Execution

The job execution is governed by the commands which FP reads from the current input file. Each command is executed as the call of one or several programs.

In detail FP acts according to the following:

1) FP reads a command from the current input file. The command may be a simple command or a compound command consisting of several simple commands enclosed in brackets.

2) The simple commands are executed one by one. The execution of a simple command means that a program file is loaded into memory and entered. Each program terminates by returning to FP which then executes the next simple command.

3) When the list of simple commands (read as described in 1) is exhausted FP resumes the command reading from the current input file.

Remarks:

ad 2) The program called by an FP command may be one of the user's own binary programs or a utility program which can perform tasks like: editing a text file into another text file, compilation of a source text into a binary program, reselection of the current input or output file, termination of the job, etc.

ad 3) The current input file is used not only by FP but also by the programs called by the FP commands. The programs can therefore read ahead in the current input file before FP starts reading commands again - they may in fact even select another file as current input.

The command reading and execution is more detailed described in chapter 4.

1.4 A Simple Example of FP Commands

An FP command consists of one or several simple commands. A simple command is a text line (terminated by an NL character) and has either the form

```
<result file> = <program name> <parameter list>
```

or

```
<program name> <parameter list>
```

Our example is the example in section 1.1.1 in (7). By removing the job specification we get the primary input file:

```
p=algol
begin real a,b;
  read(in,a,b);
  write(out,a**b);
end
p
2 10
finis
```

FP reads the command 'p=algol' and executes it by starting the ALGOL compiler. The compiler takes input from current input (as no special input file is specified) and reads from the point where FP stopped i.e. starting with 'begin...'. The reading stops when the ALGOL source program is completed i.e. after 'end'. The object program is stored in a backing storage area named 'p' and the compiler terminates by returning to FP which resumes the command reading and thereby reads the command 'p'. This command is executed as a call of the ALGOL object program which reads the two integers 2 and 10 from current input (by the call of procedure READ on the zone IN). After output of the result the program returns to FP which in turn reads the command 'finis' and thereby the utility program *finis* is called and terminates the job.

1.5 Compound Commands

A compound command to FP consists of an opening bracket '(' followed by one or several FP commands (which may again be compound commands) and terminated by a closing bracket ')'. As stated above a compound command is read by FP as an entity. Afterwards the simple commands in the compound command are executed one by one.

The primary input file

```
(
p=algol
p
finis
)
begin real a,b;
  read(in,a,b);
  write(out,a**b);
end
2 10
```

has essentially the same effect as the one above but now FP starts by reading the entire compound command (the first five lines) and next the commands are executed. The first command calls the ALGOL compiler which continues reading from current input where FP stopped. When the translation is done the next command 'p' calls the translated program which reads the integers 2 and 10 as it continues reading where ALGOL left the file. Finally the command 'finis' is executed.

1.6 Creation of File Names

Files are referred to by means of names. New file names can be 'declared' by means of the utility program SET. By the FP command

```
pip=set 40 3
```

an area by the name 'pip' containing 40 segments is created on the backing storage. The parameter '3' specifies that the area should be placed on the disk with the most permanent resources. The command

```
pip=set 40
```

creates an area on the disk with the most temporary resources.

By the FP command

```
pap=set mt16 mt471100 0 3
```

the name 'pap' is declared as pointing to file number 3 on the magnetic tape reel mt471100 (mt16=magnetic tape, 1600 bpi, PE).

Besides these explicit ways of creating a file we have also an implicit creation of files:

If a non existent file is specified as output file for a utility program (or if the file specified is protected) the utility program creates an area on the backing storage and uses it for the output.

In the earlier examples the call of the ALGOL compiler

```
p=algol
```

created the backing storage area 'p' to hold the translated program. An area created implicitly by the call of a utility program is in most cases placed on the disk with the most temporary resources, e.g. the ALGOL/FORTRAN compilers.

Remark: If the access to a magnetic tape is initiated in an ALGOL/FORTRAN program by means of the standard procedures OPEN and SETPOSITION, the name of the tape reel is used (mt471100 above) but a 'file name' as 'pap' above is not needed.

1.7 Further Examples and Remarks

The program text and the data are often too large to be conveniently included in the primary input file. Consider the input to FP:

```
p=algol ptext
if ok.no
finis
if warning.yes
(p=algol ptext list.yes
finis)
p pdata
finis
```

The first line is executed by FP as a call of the ALGOL compiler which takes input from the file 'ptext' (input is not taken from current input because this file is specified). After compilation the utility program IF is called. It tests the 'ok bit' which has been set by the compiler. If there were severe errors in the compilation (input file not found, no room for the output), the 'ok bit' is 'no' and the job is terminated by the following 'finis' command - otherwise the 'ok bit' is 'yes' and the program IF skips the next command 'finis'. Next IF is called once more and tests the 'warning bit' as set by the compiler. If the 'warning bit' is 'no', the next command (in the brackets) is skipped.

Let us assume that there are syntax errors in the program. Then the next command is not skipped and FP executes the two simple commands in the parenthesis. The first causes an extra compilation but now with a listing of the program. After compilation the job is terminated by the 'finis' command.

Next assume that the program was accepted by the compiler. Then the compound command is skipped by IF and FP reads the command 'p pdata'. This command is executed as a call of our program 'p'. The parameter 'pdata' has the function that 'p' takes input from the file 'pdata' (more precisely: the file 'pdata' is current input while the program 'p' is running).

Finaly the job terminates by the 'finis' command.

This example assumes that the files named 'ptext' and 'pdata' are available to the job. There are many ways of obtaining that, for instance:

(1) The files are permanent files on the backing storage.

(2) The files are available as magnetic tape files. In this case the names 'ptext' and 'pdata' must be declared by FP commands like

```
ptext=set mt16 mt471100 0 1
pdata=set mt16 mt471100 0 2
```

which declare the names 'ptect' and 'pdata' to describe file numbers 1 and 2 respectively on the tape mt471100.

(3) The files are on paper tapes which are loaded prior to job start by load commands to BOSS in the job's specification (7), chapter 3.

(4) The files are kept on backing storage when used - on magnetic tape when not used: the software admits the so called login files on the backing storage, which are retained as long as the user is logged in at a terminal but cancelled at logout time. If the installation has sufficent login ressources the user may start the operations from a terminal with a job which loads the files from magnetic tape to the backing storage by calling the utility program LOAD. The files are now available until the terminal is logged out. If new versions of the files are produced they must be output to magnetic tape by a job which calls the utility program SAVE.

1.8 Reselection of Current Input or Output

The utility program I selects a new file as current input file in such a way that reading from the 'old' file may later be continued (at the point where reading stopped) by a call of the utility program END (I performs a 'stacking' - END an 'unstacking' of the current input file).

The command

```
i commds
```

selects the file named 'commds' as new current input file. When FP resumes the command reading, the commands are input from 'commds' (unless current input has changed again in the meantime).

The compound command

```
(i cola
pip
end)
```

has the effect that the file 'cola' is current input while the program 'pip' is running: the first command selects 'cola' as current input the second calls 'pip' and the third switches current input back again. **Note that FP does not read from the file 'cola'.**

The utility program o selects a new file as current output file.

Consider the FP commands:

```
o specialout
p pdata
o c
convert specialout
```

The first command calls o which creates an area 'specialout' on the backing storage and selects it as current output file. Next the program 'p' is called and produces output on 'specialout'. The second call of o selects the primary output file (denoted by 'c') as current output again. The call of CONVERT tells BOSS to print the contents of the file 'specialout'.

Warning:
If an ALGOL/FORTRAN program compilation with listing is performed while a backing storage area is selected as current output file, the listing and the binary program may be competing for the room on the backing storage. If the program file does not exist in advance, the area for the listing (the current output) should be given a size sufficient to hold the program text (and the error messages). This is done by commands like

```
listout=set 40
o listout
p=algol ptext list.yes
o c
...
```

1.9 System File Names

The following names are used for system purposes but they may be used as names for private files too, as long as the private use does not conflict with the system use expected in the job:

```
boss, c, fp, primout, s, terminal, v
printer, punch, reader
and other names of devices.
```

The names c and v describe the primary output and input files.

The name of a system program may in principle be used as name for a private file but this will make the system program inaccessible for the user. Besides the bulk of system program names standard names for certain files on peripheral devices may be set up as given in appendix B.

1.10 Positionable and Unpositionable Media

Files on magnetic tape or backing storage admit a 'positioning' operation i.e. upspacing or backspacing on the tape station, selection of another segment on backing storage. A similar operation does not exist on the paper tape reader, the paper tape punch or the line printer. This fact is important because a file, when connected, is 'taken from the beginning' (the only exception being the unstacking to a former current input file).

A couple of examples will illustrate the problem:

The names 'text1' and 'text2' denote two text files. If 'f3' is a name pointing to a magnetic tape file the commands:

```
f3=copy text1
f3=copy text2
```

have the effect that 'text1' is output to the file and next the tape is backspaced and 'text2' output erasing the output just made. Contrary to that the commands

```
tpe=copy text1
tpe=copy text2
```

will produce two paper tapes containing 'text1' and 'text2' respectively.

If the data for the binary algol program 'p' is a backing storage area (or a magnetic tape file) named 'pdata' the commands

```
p pdata
p pdata
```

will yield the same output twice. Contrary to that the commands

```
p trf
p trf
```

cause two calls of the program with (usually) different input as each command will request the operator to load the next of the user's paper tapes as input for the program.

The primary input and output files are maintained by BOSS as unpositionable files i.e. one will never get the same part of the primary input file twice during the job and the data written on primary output will never overwrite earlier parts of the output.

2. Command Language

2.1 Meta Language

In the previous section we showed some examples of FP commands. In this section we will describe the syntax of FP commands by means of a modified Backus notation. The new meta-language element introduced is

```
{ <string 1> }
{   ...      }
{ <string N> }
```

With one or more strings above each other. The meaning is that any of these strings may appear at this place in the construction. A sequence of these strings in any order is denoted by:

```
{ <string 1> } a-b
{   ...      }
{ <string n> }
```

where a and b give the minimum and the maximum number of strings in the sequence. The symbol * in the place of b means just a large number of times (determined by limitations in core storage or the like).

In the special case of a = 0, b = 1, the notation

```
[ <string 1> ]
[   ...      ]
[ <string N> ]
```

will be used.

2.2 Syntax for FP Commands

The syntax for legal FP commands will be described in words as well as in the modified Backus form. In most cases the verbal explanation will be a preliminary explanation of the meaning of the definitions given in the modified Backus form. However, the sections 2.2.1 and 2.2.9 will only be in verbal form, and the sections 2.2.7 and 2.2.8 will contain explanations of terms which are context dependent and therefore difficult to explain in the Backus form.

2.2.1 Basic Terms. Classification of Characters.

In the syntax to follow, the term letter will mean one of the letters a...å, A...Å, and digit will mean one of the digits 0...9.

A delimiter will be one of the special characters period (.) or slash (/).

A bracket is either a right hand bracket or a left hand bracket, i.e. either) or (.

A visible ISO character is either a NL, a SP, or a character within the value interval 32 < character value < 127. All other characters with a value < 128 are considered to be blind.

2.2.2 Comments, New Lines, Spaces and Separators

Two special syntactic elements may need a special discussion:

```
                {            <NL>             }
<new line> ::=  { ;<text not containing NL><NL>}
                { *<text not containing NL><NL>}
```

which means that comments may be inserted between semicolon or asterisk and newline.

```
<s> ::= {             <SP>            }1-*
        { ,<text not containing NL><NL>}
```

which means that whenever one space is allowed, either a komma possibly followed by some kind of comment may be used to divide a long simple command into several text lines, or that several spaces may be used. This may facilitate the reading of complicated commands.

A separator is either a new line, a space, a bracket, the equality sign, or a delimiter.

```
                 { <new line>}
                 { <bracket> }
<separator> ::=  {    <s>    }
                 {     =     }
                 {<delimiter>}
```

2.2.3 Command Reading and Brackets

Whenever the command reading is started, FP will input one command terminated by a new line.

A command may be either one simple command or a sequence of commands enclosed in brackets. New lines may be inserted in front of a command or in front of closing brackets. Brackets may be nested.

```
<FP-input> ::= <command><new line>

<command> ::= {<new line>}0-* {<simple command>  }
                              {<compound command>}

<compound command> ::=
              {<command>}1-* {<new line><command>}0-* ,
              {<new line>}0-*
```

2.2.4 The Simple Command

A simple command is the name of a program, possibly preceded by the name of a result file as the left hand side of an equality, and possibly followed by a list of parameters. Parameters are separated by spaces.

```
<simple command> ::= [<result file> =] <program> {<s><param>}0-*
```

2.2.5 Result File Name and Program Name

The result file and the program are described by names, which consist of at least one letter followed by at most 10 letters or digits.

```
<result file> ::= <name>

<program>     ::= <name>

                                     0-10
<name>        ::= <letter>{<letter>}
                           {<digit> }
```

2.2.6 Parameters and Modifiers

After the program name a set of parameters may follow. The parameters are sequences of texts and/or integers concatenated with delimiters. The possible part after a delimiter is called the modifier, a term which is recursively defined. A parameter may consist of a single integer or text item.

```
            { <text>   }
<param> ::= {          } [<delimiter><modifier>]
            {<integer> }

<modifier> ::= <param>
```

2.2.7 Texts

Texts may have several forms. They may be names starting with at least one letter. They may be apostrophized names, which may start with a digit, but where an apostrophe signals that a sequence of digits and/or letters should be treated as a name. They may be generalized names, which may start with a digit but should contain at least one letter not too far from the start of the text. At last, texts may be general texts, where a sequence of characters must be included in quotes. The character sequence may contain capital, as well as small letters, digits, special signs, and spaces, except quotes and NLs.

```
             (       <name>        )
             (<apostrophized name>)
<text> ::=   ( <generalized name> )
             (   <general text>    )

                                        1-11
<apostrophized name> ::= ' (<letter>)
                           (<digit> )

<general text> ::=      "<a sequence of at most 95
                        visible ISO characters neither
                        containing NL nor ">"
```

2.2.7.1 Generalized Names

A generalized name is a sequence of at most 11 letters and/or digits containing at least one letter. The sequence of digits in front of the first letter must not yield an apparent value > 16 777 215. The term apparent value will be described below. The notion generalized name makes it possible for FP to read names as e.g. 023sub5509 without requiring an apostrophe.

2.2.8 Integers

Integers may be written as usual, i.e. in radix 10, as well as in other radices. A radix is separated from the value to be used by means of a colon. The digits to be used in integer reading must be sensible, i.e. they must be less than the radix. Letters may be used as digits when for instance hexadecimal (radix 16) integers are read. Integers may be signed, but the sign must be placed in front of a possible radix. The sign will be used on the resulting integer.

```
<integer> ::= [ + ] ((<radix>:)0-* (<legal digit>)1-* )
              [ - ]

<radix> ::= (<legal digit>)1-*
```

The terms legal digit and apparent value are dependent on the implementation and on their context.

2.2.8.1 The Apparent Value

Integers are represented in 24 bits. Let the term the apparent value mean the value that you get when reading a digit sequence. This apparent value must not exceed 16 777 215. However, apparent values >8 388 607 will be treated as negative integers, i.e. with a value = -(16 777 216 - apparent value).

2.2.8.2 Legal Digits

A legal digit is a digit symbol with a value not exceeding the value of the radix. A digit symbol is either a digit or a letter. The digits 0..9 have the values 0..9, and the letters a..å have the values 10..38. In integer reading, the initial radix is 10, thus allowing precisely the digits 0..9 as legal digits.

2.2.8.3 Leading Zeroes

In principle leading zeroes do not distort the apparent value. However, as the syntax accepts generalized names, a sequence of more than 11 digits not following a plus (+), a minus (-), or a radix symbol (:), will be classified as an illegal generalized name.

2.2.8.4. Examples of Legal Integers

```
8388607             will be read as decimal   + 8 388 607
-123456                       - " -           - 123 456
-16:f                         - " -           - 15
+16:ffffff                    - " -           - 1
-16:ffffff                    - " -           + 1
 40:abc                       - " -           + 16 452
100:a:8:12:123                - " -           + 123
16777214                      - " -           - 2
```

Note that in case of a faulty radix <=0, all digits and letters will become illegal, thus the attempt of representing 83 as 16777206:123 will result in a syntax alarm.

2.2.9 Modifications of the Syntax

In order to facilitate reading and typing of long and complicated commands, the following five modifications of the principal syntax described above have been included.

2.2.9.1 Blind Spaces

The separating syntactic entity <s> may be inserted freely in front of or after names, texts, separators, and integers.

2.2.9.2 Missing Unquote at the End of a Command

General texts may be terminated by a NL in stead of a terminating quote. If a NL is used as terminator, the NL is considered to terminate the general text as well as the (simple) command.

2.2.9.3 Missing Spaces in Front of Special Characters

A missing separator in front of a sign, an apostrophe, or a quote is considered to be a space.

2.2.9.4 Missing Space after Unquote

A missing separator after a terminating quote of a general text is considered to be a space.

2.2.9.5 FP Cancel

If a question mark (?) appears outside comments and outside general texts, it will cancel everything that has been typed in the current command, whether it is compound or simple. This is a way of getting out of a bracket structure in online mode. FP will resume input immediately after the question mark.

2.3 Semantics of FP Commands

The command (simple or compound) read by FP is stored in the FP command stack (a part of the memory area for the job). Next the simple commands are executed one by one. The simple command

```
[<result file> =]   <program> <parameter list>
```

is executed as a call of the program named <program>. The program will usually examine the simple command which caused the call of the program in order to get the parameter list and find the name of a possible result file.

The use of result file and parameters depends on the program in question but as general rules we have:

Result file:
For most utility programs this name specifies an output file. If no file with this name exists or if the file found is protected, an area on the backing storage is created and used for the output. For some utility programs (SET, ENTRY) the result file name specifies a catalog entry which is to be created or changed. In the call of a translated ALGOL/FORTRAN program the result file name has only the function that it is available from the program by a suitable call of procedure SYSTEM.

Parameter list:
The parameters in the parameter list specify input files, various modes of operation for the program etc. For programs requiring text input (i.e. compilers, assembler) we have the convention that input is taken from current input if no input files are specified and otherwise from the specified files. If the first parameter (following the program name) in the call of a translated ALGOL/FORTRAN program is a single name (not followed by a point), the file given by this name is used as current input for this program (compile time option). If the parameter is a single integer the program overwrites FP (Also compile time option, cf (5), 10.3).

A translated ALGOL/FORTRAN program may examine the parameter list by means of the procedure SYSTEM.

2.4 Format of the FP Command Stack

The FP command stack consists of items each containing a separator and the succeeding name, text, or integer (if any).

The heading word of an item has the format:

```
<separator> shift 12 + <length>
```

The separator is an integer with the values

```
 -4: -2: end of command list
   0: 2: right hand (or end-) bracket
   4: 6: left hand (or begin-) bracket
      8: new line
         space (<s>)
         equality (=)
         periode (.) or slash (/)
```

The length is an integer with the values

```
   0: 2: nothing follows (i.e. end command list)
  4: 10: the next separator follows
         an integer follows
 8*n+10: a name, apostrophized named, or generalized name
         follows
         a general text follows (n = 1, 2, ... , 7)
```

Integers are stored in 24 bit words, and names are stored as 8 bit characters with 3 characters per word, supplemented with at least one NUL character to fill up the item.

2.4.1 Example

The command

```
pip = prog avs.2:11 2.muks "BIG ONE"
```

appears as follows in the FP command stack:

```
stacktop + 0:   2 shift 12 + 10 ; new line, name follows
         + 2:   pip             ; name, 4 words
         +10:   6 shift 12 + 10 ; equality, name follows
         +12:   prog            ; name, 4 words
         +20:   4 shift 12 + 10 ; space, name follows
         +22:   avs             ; name, 4 words
         +30:   8 shift 12 + 4  ; period, integer follows
         +32:   3               ; integer, one word
         +34:   4 shift 12 + 4  ; space, integer follows
         +36:   2               ; integer, one word
         +38:   8 shift 12 + 10 ; period, name follows
         +40:   muks            ; name, 4 words
         +48:   4 shift 12 + 18 ; space, general text follows
```

```
+50:   BIG ONE            ; general text, here: 8 words
+66:   2 shift 12 + 2     ; new line, nothing follows
+68:   -4 shift 12 + 0    ; end command stack
```

The item which terminates the simple command (here new line) is not available using the procedure SYSTEM in ALGOL/FORTRAN.

2.5 The Action on End Medium

Whenever an end medium character is encountered during command reading, FP will unstack current input to the previously selected file and resume input from here. No message will be issued during the unstacking, if it performs smoothly. The end medium character is considered to be blind. As usual unstacking the primary input file is blind.

This rule has been implemented in order to take care of a missing end terminating a currently selected input file. It may be used to augment large commands ending with the construction ,<NL><EM> or with an open ended bracket structure.

2.6 The Action on Syntax Errors

Whenever a syntax error is found during command reading, FP will issue a message which may take one of two forms, either

*****fp syntax:**
```
<at most 14 characters up to the error>
read from <the name of the input file>
unstacking to <the file which selected it>
```

or

*****fp syntax:**
```
<at most 14 characters up to the error>
read from primary input
```

If the syntax error has been found in a file which is not the primary input file, FP will unstack the current input file, forget about what has been read up to the error, and try to read from the previously selected current input file.

If, however, the syntax error has been found in the primary input file, and if FP has found at least 3 consecutive syntax errors, i.e. without finding one syntactically legal command, the job will be terminated with the alarm message

*****fp job termination**

Otherwise FP will read up to the next new line character, and start interpretation from there.

These rules have been introduced in order to prevent infinite looping in error messages whenever edit commands or program texts are accidentally selected as command reading files. On the other hand, if syntactically correct commands have been read, the programmer should not be punished by only being allowed a limited number of errors during one session.

The mode bits OK and WARNING (see section 1.7.) are not affected by syntax errors. They keep the value set by the previously executed command.

3. Job and Operating System

3.1 Job And Parent

The phrase 'the operating system' is somewhat ambiguous as several operating systems may be present. A BOSS job may in fact act as an operating system and start a 'child' job inside its own memory area.

We will use the term parent to denote the operating system for the job considered.

3.2 Parent Messages

A job communicates with its parent by sending parent messages. A parent message is sent when the job needs the help of the operator (mounting of magnetic tapes etc.) or when an action from the parent is needed (the job is through and to be removed, etc.).

Most parent messages are sent automatically by FP and the other programs when needed (e.g. mounting of magnetic tapes), some parent messages like

FINIS, MOUNTSPEC, TIMER, CONVERT

are sent by calling special utility programs. cf. (8) for a complete list of the parent messages.

3.3 Job Start, Initialization of FP

At job start the parent inputs FP (or rather a part of FP) to the foremost part of the job area and starts the initialization of FP with information about **primary input and output**. During the initialization of FP the job creates catalog entries named v and c describing the primary input and output files respectively (if such entries are already present at job start they are removed by the job, unless they point to the proper files, in which case no new v and c are created). The initialization ends by connecting the primary input and output files as current input and output files and the FP command reading is entered.

The above initialization is activated by FP itself in case of

- troubles reading from primary input
- troubles writing on primary output
- troubles writing a device status alarm on current output, cf 4.5
- the job is restarted with an answer to a parent break message, cf. 4.5

or forced by the command INIT.

If this **reinitalization** is done more than 10 times, the job is terminated, sending a FINIS parent messages, cf. 3.4 or a 'troubles with c or v' parent message, cf. B.4.

At job start the parent imposes three catalog bases on the job: **standard base, user base and max base**. These bases determine which files on the backing storage the job may access and how the catalog entries created by the job are placed in the catalog (ref. (7), 5.2).

The **resource claims** of the job are fixed at job start. The housekeeping of the backing storage, message buffer and area process claims during the job execution is done by the monitor (and the actual values may be found in the monitor's process description of the job process, cf. (2)). The other resources are maintained by the parent all the time.

Before entering any program FP zeroes all **modebits**, except 'initmess', which is set 'true', and 'bswait', which is set 'true' if the name of the parent is 's', (in case of reinitialization 'ok' and 'warning' are set, cf. 4.2), masks off the **overflow/underflow interrupts** (integer overflow, floating point overflow/underflow) and removes **area processes** and **pending answers** as mentioned in 4.5.

3.4 Job Termination

When the job is terminated by the FP command FINIS, the following happens: the current output buffer is emptied, the entries v and c are removed and a 'finis' message is sent to the parent. The finis message causes BOSS to remove the job and afterwards scan the catalog and remove all temporary catalog entries belonging to the job which just finished. The operating system may remove the job without request from the job (a time limit is exceeded, the job is killed by the operator etc.). In this case BOSS performs a 'provoked break' on the job (see below). If the FP code is intact (which is normally the case) an error text is printed on current output (***break 8) and a 'break' message is sent to the parent (alias BOSS) who removes the job.

3.5 Break Actions

In some severe error situations the FP break routine is entered. The break routine outputs an error text on current output, empties the buffer and sends a 'break' message to the parent. When BOSS receives the break message it makes a partial clearing after the job and if the job has not used all of its run time and not read all of its primary input file the job is restarted with a fresh FP (ref. (8), section 3.4). The error text is:

```
***break <cause>  (<instruction counter>)
                  (<break 10 reason>    )
```

The integer <cause> explains why the break routine was entered:

cause = 0: Internal interrupt
Caused by attempt to execute an illegal instruction (may for instance occur in an ALGOL/FORTRAN program with index error and translated with 'index.no').

cause = 2: Integer overflow

cause = 4: Floating point overflow/underflow

cause = 6: Parameter error in monitor call
This error is provoked by the i/o system if there are not 'enough message buffers' - it may also be caused by for instance a wrong parameter to one of the monitor procedures in an undebugged code procedure.

cause = 8: Parent break
Breakpoint caused by the parent, cf. above.

cause = 10: Zone stack error
The break routine was entered because of troubles during stacking or unstacking of a zone (cf. the next chapter). The zone stack error may occur for various reasons. The most common is

```
***break 10 1
```

caused by resource limitations (lack of entries or segments on the backing storage). In details we have the following possibilities:

reason = 0:
The zone has too many shares - erroneous zone stacking in the utility program.

reason = 1:
The job does not have the resources (entries or segments) on the backing storage for stacking the zone.

reason = 2:
i/o troubles during zone stacking.

reason = 3:
The entire buffer area does not comprise a multiple of 512 storage halfwords - erroneous zone stacking in the utility program.

reason = 4:
Same as reason = 3 but during a zone unstacking.

reason = 5:
The zone unstacking cannot proceed because a previously stacked zone is not found in the catalog.

reason = 6:
i/o troubles during unstacking of the zone.

4. The Execution of FP Commands

The reading and execution of FP commands are performed by the command reading routine, the program loading and the program termination routine in FP. By setting the mode bits the programmer may modify the function of these routines in various ways.

4.1 Current Input and Output, Zone Stacking

The FP commands are read from the current input file. At job start, after a break or by a reinitialization of FP, the primary input and output files are selected as current input and output files.

The current input and output files may be reselected during the run (cf. section 1.8). The selection of a new current input file by the I command uses a **zone stacking** where the actual contents of the data buffer are stored in an area on the backing storage (the stacked zone) before the new file is connected. The reselection of the former file by the END command is the opposite process - a **zone unstacking** - where the former contents of the data buffer are restored from the stacked zone.

Many of the utility programs use zone stacking for internal purposes. The programmer need normally not care for that, but if the resources (entries and segments on the backing storage) needed for the zone stacking are not present it may, however, result in a 'break 10' in unexpected situations.

The current input and output files are available for character input and output respectively from ALGOL/FORTRAN programs via the standard zones IN and OUT (cf. sections 1.2 and 1.8).

Warning: Block oriented input/output procedures (INREC, OUTREC) or the procedures OPEN and CLOSE should not be applied to the zones IN or OUT as this may have a serious influence on the function of FP. If a certain file is wanted as current input while an ALGOL/FORTRAN program is running, the file should be given as parameter in the program call (cf. section 2.3). If a certain file is wanted as current output the O command is at hand.

4.2 The Mode Bits

FP contains 24 mode bits each of which has the value 'yes' or 'no'. The mode bits are numbered 0, ..., 23. They are set by the MODE command and tested by the IF command. Furthermore FP sets some of the bits at each program termination.

The bits with numbers 0 to 11 may be used by the programmer as 'flags', the other bits have special functions. These special mode bits have names. At present the following special mode bits are in use:

bit 23: list
Governs the 'list mode' of FP: In the list mode each FP command is listed on current output just prior to execution (cf. section 4.4).

bit 20: pause
If this bit is 'yes' the break routine of FP is entered after program termination (cf. section 4.5).

bit 19: error
If this bit is 'yes' and a program terminates unsuccessfully (with 'ok no' or 'warning yes'), the FP break routine is entered (cf. section 4.5).

bit 18: ok

bit 17: warning
The ok and warning bits are set by FP at program termination reflecting the success of the program just executed.

bit 16: if
Used internally by FP.

bit 15: listing
This bit is tested by assembler and compilers. If it is 'yes' the source program is listed unless 'list.no' is stated in the FP command calling the assembler (compiler).

bit 14: initmess
If this bit is 'yes' and FP reinitializes itself, either because of the reasons mentioned in 3.3 or forced by the command INIT, the FP version and release identification is listed on primary output just after connection.

bit 13: bswait
If this bit is set, and the I/O system (FP or ALGOL/FORTRAN) during output to a backing storage area needs extra disk resources, the parrent message

```
bs <disk name> <segments> 0
```

will leave the wait bit set, i.e. the process will wait for the resources to be given.

At job start and after a 'break' all the mode bits have the value 'no'. The mode bits 'ok' and 'warning' are set by FP at each program termination, the other mode bits may be changed by the MODE command. A call of the command INIT, or a severe error which causes a reinitialization of FP but not a 'break', cf. 3.3, sets the 'ok' and 'warning' bits but the other mode bits are left unchanged.

4.3 Command Reading

The FP command reading is entered at job start or whenever all the simple FP commands read so far have been executed (command stack empty). It proceeds as follows:

An FP command (simple or compound, cf. chapter 2) is read from current input, syntax checked and stored in the FP command stack in the job process.

The FP command stack pointer is set and the FP load program routine is entered.

If an EM character is found during the command reading, the current input file is unstacked and the command reading continued.

The FP action on syntax error is described in 2.6.

4.4 Program Loading

The FP program loading routine proceeds as follows:

The FP command stack pointer is upspaced and if the command stack is exhausted, the command reading routine is entered.

The program name in the actual simple FP command is looked up in the catalog and it is checked whether the file is a binary program file (contents key, cf. section 5.3).

If the 'list bit' is 'yes' the command is listed on current output.

If the program name is found in the catalog to be a binary program, the program is loaded into memory and entered.

If the program name is not found in the catalog, if the name does not describe a program file or if the loading of the program causes troubles (memory area suze too small, i/o troubles), an error text is printed on current output and the FP program termination routine is entered (instead of the program) as after an unsuccessful execution.

4.5 Program Termination

A program can terminate in four different ways:

1) Exit to the FP program termination routine.
2) Termination caused by hard error on a file (i/o troubles).

3) Exit to the FP break routine.
4) Exit to FP job finis.

In the two last cases the 'break' or 'finis' action as described in chapter 3 is performed and the FP code, which is currently in the job memory area, does not return to normal operation: the parent may remove the job or load a fresh FP.

In case of break, though, if an answer to the parent message is received and the job is restarted, FP goes on to reinitialize itself. If 'error' or 'pause' is 'yes', FP will exit from reinitialization via end program into FP break routine again. If both bits are 'no', FP will go from reinitialization to command reading, cf. program termination routine below.

If the termination is caused by i/o troubles, an error text ***** device status...** identifying the file and the error is printed on current output and the FP program termination routine is entered with 'ok.no' and 'warning.yes'. (Hard errors on current input or output causes further action before the program termination routine is entered).

The FP program termination routine has the following function:

The 'ok' and 'warning' bits are set as signalled by the program. If the 'pause' bit is 'yes' or if the 'error' bit is 'yes' and either the 'ok' bit is 'no' or the 'warning' bit is 'yes' the FP break action is entered.

Remark: The IF and MODE programs make an anomalous exit to FP which bypasses the actions described so far.

The overflow/underflow interrupts are masked off.

A NULL character is printed on current output. If current output is connected to a character oriented device (terminal, printer, punch), the data buffer is output. If the current input zone has been stacked by the program for internal purposes, the zone is unstacked. (The I program tells that the current input zone should not be unstacked by setting the 'i-bit': bit 1 shift 0 in the give up mask' in current input zone).

The area processes in the monitor are scanned. If the job is user of an area process the area process is removed.

The event queue of the job process is scanned and pending answers not belonging to the current input file are waited for. If the command stack is empty, the FP command reading else the FP load program routine is entered. (The terms: area process, event queue, answer are explained in (1) and (2)).

4.6 Resource Requirements

The File Processor needs a minimum memory area of 3640 storage halfwords (plus space for the command, at least 10 halfwords, cf. 2.4.1) in order to be able to operate. The memory area is used as follows:

2616 storage halfwords are occupied by the resident FP code, buffers for current input and output and 'process descriptions' for primary input and output.

A variable part (usually small) is used for the command stack.

1024 further storage halfwords are used by FP between execution of the programs.

When a program is executed, a storage area of the size:

```
job size - 2616 - command stack size
```

is available for the program. This storage area must be at least 1024 halfwords.

Besides storage area the programs and FP need other system resources like message buffers, area processes, segments and entries on the backing storage etc. Note that many utility programs perform one or several zone stackings each of which uses an entry and one or two slices on backing storage.

The standard resources of a BOSS job are usually chosen to be enough to execute any of the utility programs.

5. References to Files

5.1 Document Name of a File

All data transfers in RC9000-10 are under supervision of the monitor: the transfer of a data block is initiated by a call of the monitor procedure 'send message' and the completion of the transfer is awaited by a call of the monitor procedure 'wait answer'. An 'i/o message' sent by a 'send message' is addressed to a process which is so to say the monitor's representative of the data file. The i/o messages are sent automatically by the i/o system. The name of the process (representing the data file) is called the document name of the file.

Corresponding to the different types of peripheral equipment, the monitor has various types of processes: the line printer corresponds to a process named 'printer', the paper tape reader to a process named 'reader', the paper tape punch to a process named 'punch', the console and terminals to processes with names like 'console1', 'terminal3', '0henrik31' etc. A magnetic tape station corresponds to a process carrying the same name as the magnetic tape reel, which is currently mounted on the station.

The backing storage is treated in a special way because one single device (a logical disk) is divided into several files (data areas). An area on the backing storage is identified by its name and **this area name becomes the document name** when the area is used for input/output: the i/o system prepares the access to the area by calling the monitor procedure 'create area process' with the area name as parameter. This results in an area process to which the i/o messages are addressed.

Remark: Each logical disk has a name which distinguishes it among other logical disks. This name is of interest to the programmer in other connections, for instance when a new area is created. The use of the term 'document name' in the monitor manual to denote this device name should not be confused with the above concept of document name for a file.

5.2 File Descriptor, File Name

The software has two i/o systems, the ALGOL/FORTRAN i/o system and the FP i/o system. The first one is used by translated

ALGOL/FORTRAN programs, the second one by FP itself and the assembler coded utility programs. The two i/o systems differ in the way the programmer has to specify files.

The information needed in order to connect a file forms a file descriptor. It includes (among other things) the document name of the file.

When a file is connected by the FP i/o system, the file name is used to specify the file (cf. chapter 1). This file name is the name of a catalog entry containing the file descriptor for our file. The use of the catalog entry is described in section 5.5.

When an ALGOL/FORTRAN program connects a file, the document name is given in the list of parameters (or in the zone) to the procedures OPEN and SETPOSITION.

5.3 The Constituents of a File Descriptor

Document name:
The significance of this name is explained above.

Kind:
This integer selects the actions to be taken by the i/o system when the file is connected, when the use of the file is terminated and if special situations should occur during a data ransfer (see the next chapter for further details). Each kind corresponds roughly to a type (or a class of types) of peripheral equipment.

Mode:
This integer specifies a certain hardware mode (e.g. density on magnetic tape, blind input form a terminal) or a code conversion (e.g. conversion from flexo to ISO code by paper tape input). The mode is a part of the i/o message which starts the transfer of a data block and the mode specified is contained in each i/o message.

File count:
Integer, relevant for magnetic tape only. A magnetic tape reel is divided into files numbered 0, 1, 2, ... by tape marks. Usually the file number 0 contains an ISO label identifying the tape reel ref. (7), 6.1.

Block count:
Integer, relevant for backing storage and magnetic tape. The blocks are numbered 0, 1, 2, 3... By specifying a block count different from zero, the 'subfile' starting at this block is obtained.

Contents key:
Integer, specifying the intended use of the contents of the file (e.g. text file, binary program etc.). A list of the values is given in appendix B.3.

Entry point:
Integer, relevant for binary programs only. Specifies the entry point address relative to the start of the program.

Load length:
Integer, specifies for a binary program the number of halfwords which should be loaded into memory before the program is entered (for a program using segmentation only a part of the program needs to be loaded).

The combination of mode and kind is called the mode-kind. For each kind only certain modes can be used. The commonly used modekinds are listed in appendix B.1.

5.4 Catalog Entries

The monitor maintains a file catalog on the backing storage. This catalog is itself a backing storage area named 'catalog' and consists of records called catalog entries. Changes in the catalog i.e. creation, change or removal of catalog entries are done by the monitor on request from internal processes (e.g. the job, BOSS) calling the special monitor procedures 'create entry', 'change entry', 'rename entry', 'remove entry' etc. The use of these 'catalog procedures' are subject to certain restraints as described in (1), (2) and (7).

A catalog entry consists of a 7 word entry head and a 10 word entry tail: when a catalog entry is created or changed, the name and the entry tail is specified (and based on this, the monitor computes the entry head). The utility programs SET, SCOPE, a.o. create or change catalog entries by calling the relevant monitor procedures. The entry name and tail in these monitor procedure calls are taken from the parameters in the SET or SCOPE command.

By means of the sign of the first word in the entry tail, the monitor distinguishes between two types of catalog entries. If the first word is non-negative the entry is an area entry, otherwise the entry is a non-area entry. The area entries are used by the monitor in the management of the backing storage. Each area entry defines a data area where the size and physical location is determined by means of the entry head and the first five words of the entry tail. The first word in the entry tail contains the number of segments in the area, the next four contains the name of the logical disk on which the area is located.

5.5 Formation of the File Descriptor

The connection of a file by the FP i/o system starts with a catalog lookup for the file name. The tail of the entry found is used to form the file descriptor as follows:

a) **Document name, mode, kind:**

a1) Area entry. If the entry is an area entry the file name is used as document name and the values
`mode=0, kind=4`
as mode-kind. This means simply that we are going to connect the data area determined by the area entry.

a2) Non-area entry: Document name, mode and kind are taken from the first five words of the entry tail as follows:

```
word 1 : 1 shift 23 + mode shift 12 + kind
word 2-5: document name
```

b) The rest of the file descriptor:

The rest of the file descriptor is determined by word 6-10 in the entry tail. The use of this part of the entry tail depends on the value of the left byte of word number 9 (the contents key)

b1) Contents key < > 4 and < 32:

```
word 6 :  not used
word 7 :  file count
word 8 :  block count
word 9 :  contents key shift 12 + entry point
word 10:  load length
```

b2) Contents key = 4 or > = 32:

The file is an ALGOL/FORTRAN procedure.

The values

```
file count = block count = 0
```

are used. Entry point and load length are irrelevant, as FP does not interpret the file as a program file. The five last words in the entry tail are used (by the ALGOL/FORTRAN compilers) as follows:

```
word 6 :   procedure code entry specification
word 7-8:  procedure parameter specifications
word 9 :   contents key shift 12 + start ext. list
           code segm. shift 12 + halfs in own memory
word 10 :  area.
```

Further details are found in (4).

5.6 Entry Tails

By collecting the information above, we find that there are four types of entry tails:

I. Area entry, not ALGOL/FORTRAN procedure:

```
word 1      : number of segments in the area
word 2-5    : name of logical disk
word 6      : shortclock
word 7      : file count
word 8      : block count
word 9      : contents key shift 12 + entry point
word 10     : load length
```

Remark: The area entries are characterized by word 1 > = 0. The name in word 2-5 is not used by FP when the file is connected, but the entry name is used as document name. The value of contents key is < > 4 and < 32.

II. Area entry describing ALGOL/FORTRAN procedure:

```
word 1     :  number of segments in the area
word 2-5   :  name of logical disk
word 6     :  procedure code entry specification
word 7-8   :  procedure parameter specifications
word 9     :  contents key shift 12 + start ext. list
word 10    :  code segments shift 12 + halfs in own memory
              area
```

Remark: Further details are given in (4).

III. Non-area entry, not ALGOL/FORTRAN procedure:

```
word 1      : 1 shift 23 + mode shift 12 + kind
word 2-5    : document name
word 6      : not used
word 7      : file count
word 8      : block count
word 9      : contents key shift 12 + entry point
word 10     : load length
```

IV. Non-area entry describing ALGOL/FORTRAN procedure:

```
word 1     :  1 shift 23 + mode shift 12 + kind
word 2-5   :  document name
word 6     :  procedure code entry specification
word 7-8   :  procedure parameter specification
word 9     :  contents key shift 12 + start ext. list
word 10    :  code segments shift 12 + halfs in own memory
              area
```

6. The FP Input/Output System

6.1 Text Files and EM Characters

The i/o system is concerned with the proper transfer of the data only, and not with the meaning of the contents of the data blocks. This fact is important in dealing with text files, where the appearance of an EM character signals the end of the text. As the i/o system does not examine the individual characters, the EM character does not cause any 'end text signal' from the i/o system but the program which is processing the text, has instead to discover the EM character by inspecting each character in the input.

An EM character need not be present, but the file may instead just finish (e.g. file mark on a magnetic tape). In this situation the i/o system simulates the input of a data block containing an EM character and in this way the program still gets the proper information about the text end.

The utility programs write a terminating EM character in text files on backing storage or magnetic tape but not in text files on other media. It is advisable to do so whenever the output of a text file is terminated.

6.2 Connection of a File

The connection of a file is based on a file descriptor (obtained from the file name as described in the previous chapter). The connection includes initialization of various tables (zone and share descriptions) and some sort of initialization of the process associated to the file. The i/o system is able to operate under the primitive operating system s as well as the advanced operating system BOSS. In the latter case some of the devices (terminals, tape reader, card reader, line printer) are spooled and the 'i/o conversation' goes via pseudoprocesses (ref. (2), 2.80). The i/o system is suited to deal with this type of processes too.

The connection proceeds according to the kind specified in the file descriptor:

Kind = 0:
(Internal process). The maximum buffer length is set to 512 halfwords (768 characters) and the existenc e of the process is checked.

Kind = 2:
(Clock process). Not allowed.

Kind = 4:
(Backing storage area process). The maximum buffer length is set to 512 halfwords (768 characters). If the process is not already present, the area process is created. The connection may also, depending on the circumstances, include creation of the area.

kind = 6:
(Disk process). As for area process, except no area or area process is created. Instead the existence of the disk process with executing process as user is checked.

Kind = 8:
(Terminals). The maximum buffer length is set to 104 halfwords (156 characters) and the existence of the process is checked.

Kind = 10:
(Paper tape reader). The maximum buffer length is set to 36 halfwords (54 characters). The process is reserved and input messages are sent until 'empty reader' is sensed. Then a 'load reader' message is sent to the parent and the mounting of the tape is awaited by attempting a block input once every second until a non-empty block is obtained. If the reader was reserved by another process, a 'wait for reader' message is sent to the parent and the job awaits the reader by making an attempt to reserve it once every second until the reservation is successful. (Under BOSS the major part of these actions are dummy).

Kind = 12:
(Paper tape punch). The maximum buffer length is set to 80 halfwords (120 characters). The process is reserved and 100 NULL characters (blank tape feed) are output.

Kind = 14:
(Line printer). The maximum buffer length is set to 80 halfwords (120 characters) and the process is reserved.

Kind = 16:
(Card reader). The maximum buffer length is set to 80 halfwords (120 characters). Apart from that the connection proceeds as for kind = 10 (paper tape reader).

Kind = 18:
(Magnetic tape). The maximum buffer length is set to 512 halfwords (768 characters) and the process is reserved. If the process is not available for the job, a 'mount tape' message is sent to the parent. If the file is to be used for output and the tape is not write enabled, a write enable message is sent to the parent. Finally a 'set mode' and a 'position' message is sent to the process -the latter starts the positioning to the file and block count given in the file descriptor.

6.3 Termination of the Use of a File

When the use of a file is terminated, the process is released in order to make it available to others, and the area process (if any) is removed in order to retain the area claims. On a punch (kind = 12) a tape feed of 100 NULL characters is output. For magnetic tape output two tape marks are written after the last block and the tape is positioned after the first one.

Note that a 'release message' is not sent to the parent when a magnetic tape file is terminated and hence BOSS (if it is the parent) will keep the magnetic tape on the station so that a new mounting is not needed if the tape is used later in the job. The release message to the parent may be sent by a RELEASE command. In this way the station is made available for mounting of another tape reel (cf. (7), 6.1).

6.4 Data Transfers, Status Word

When the transfer of a data block is checked, the outcome of the transfer is expressed by the number of storage halfwords transferred and a 24 bit status word. The 12 leftmost status bits are generated by the monitor which takes most of the bits directly from the hardware, the other bits are generated by the i/o system. The two i/o systems (ALGOL/FORTRAN and FP) use the same status bits ((5), chapter 2).

The meaning of the bits is as follows:

1 shift 23:
(Intervention). The device was in the local mode.

1 shift 22:
(Parity error). A parity error was detected during the transfer.

1 shift 21:
(Timer). The operation was not completed within a certain time defined by the hardware or the monitor.

1 shift 20:
(Data overun). The RC9000-10 system bus was overloaded and could not transfer the data.

1 shift 19:
(Block length error). A block input from magnetic tape was longer than the buffer area allowed for it.

1 shift 18:
(End of document). Means various things on the different types of devices: data transfer outside the backing storage area was attempted, the end of tape reel was sensed on magnetic tape, the paper tape reader was empty, the paper tape was exhausted on the punch, the paper supply was low on the printer, the input hopper was empty on the card reader.

1 shift 17:
(Load point). The load point was sensed after an operation on magnetic tape.

1 shift 16:
(Tape mark or attention). The attention/escape key was pushed during i/o to the terminal, a tape mark was sensed or written on the magnetic tape.

1 shift 15:
(Writing enabled). The magnetic tape was write enabled.

1 shift 14:
(Mode error). A wrong mode (density) was selected on the magnetic tape station.

1 shift 13:
(Read error). I/O error on disk or read error on the card.

1 shift 12:
(Card rejected). The card was rejected by the card reader.

1 shift 11:
(Checksum error). Checksum error detected by the invar/outvar system.

1 shift 10:
(Bit 13). Not used.

1 shift 9:
(Bit 14). Not used.

1 shift 8:
(Stopped). Less than wanted was output to a file of any kind or no data was input from a backing storage area or disk process. The bit appears for instance if the job was stopped (swopped) during the data transfer.

1 shift 7:
(Word defect). The number of characters transferred to or from a magnetic tape is not divisible by the number of words transferred, i.e. only a part of the last word was transferred.

1 shift 6:
(Position error). The position on the magnetic tape (file and block count) reported by the monitor differs from the position expected (e.g. an unexisting position was specified in a positioning, by mistake the magnetic tape was used for two purposes at the same time and the present one was output).

1 shift 5:
(Process does not exist). The document name does not correspond to any process. For backing storage this may indicate that the area does not exist or that the job does not have the resources to create the area process (area claim too small).

1 shift 4:
(Disconnected). The power on the device was switched off.

1 shift 3:
(Unintelligible). The operation attempted is illegal on that device (e.g. input from a printer).

1 shift 2:
(Rejected). The job is not allowed to use the process as it should be reserved first (the device was not claimed in the job specification, the area is protected against output from the job. Can also occur if the file by mistake was used for two purposes at the same time and then released by the termination of one of the uses).

1 shift 1:
(Normal answer). None of the status bits 1 shift 5 to 1 shift 2 are set, i.e. the monitor has accepted the operation and the device has attempted to execute the operation.

1 shift 0:
(Give up). The standard recovery actions could not succeed, i.e. hard error on the transfer.

If a hard error on a file causes a program termination, a 'device status' error text containing the status word of the unsuccessful transfer is printed. The status bits are given by the labelling texts in the brackets above (the bits 1 shift 1 and 1 shift 0 are ignored in printing the error text).

If the error is caused by hardware malfunction, the FP end program routine reports the error not only to the programmer (by the 'device status' text) but also to the parent by sending a 'status' message. The parent may then attend the operator (BOSS displays the status message on the main console).

6.5 Standard Recovery Actions

The FP I/O system has a standard recovery routine which is entered if an anomalous status word appears. The recovery proceeds according to the kind specified. All situations not covered are treated as hard errors. A hard error causes the FP i/o system to give up, i.e. termination of the program and output of an error text on current output (see above). If the hard error is on the current input or output file, special measures are taken before the error text is output (cf. section 6.6).

Kind = 0:
Intervention: Ignored.
End of document during input: Ignored.
End of document during output: A 'change' message is sent to the parent. Upon the receipt of the answer from the parent, the remaining part of the data block is output.
Stopped: The remaining part of the data block is transferred.

Kind = 4:
(Backing storage area).
Data overrun: The transfer is repeated.
End of document during input: If nothing has been transferred, the input of two halfwwords containing three EM characters is simulated,

else the bit is ignored.
End of document during output: The area is enlarged and the transfer is repeated. If the area cannot be enlarged because of empty claims for the disk, a parent message

```
bs <disk name> <segments> 0
```

is sent, specifying the size of the expansion tried. If the mode bit 'bswait' is set, the wait bit of the parent message is set, giving the user an opportunity to add to the disk claims of the process before starting it again.
Stopped: If the end document bit is not present, the remaining part of the data block is transferred.
Process does not exist: The area process is created and furthermore reserved if the operation is output. After this the transfer is repeated.
Rejected during output: The area process is reserved and the transfer is repeated.
Rejected during input: Hard error.

Kind = 6:
(Disk process).
Data overrun: The transfer is repeated.
End of document during input: As for area process.
End of document during all other operations: Hard error.
Stopped: May appear at all operations. The operation is repeated except if it has been overruled by the end of document action or the two actions below.
Process does not exist: An area process is created and the action proceeds as for area process.
Rejected: If the disk process does not exist or the calling process is not a user, it is a hard error. If the operation is initialize, cleantrack or output the disk process is reserved for exclusive access. If this is not possible it is a hard error. Now the operation is repeated.
Note that if the process does not exist, an area process will be created only if an entry of the process name exists in the main catalog. If not so, it is a hard error.

Kind = 8:
(Terminals).
Timer during input: Ignored.
Tape mark or attention (attention/escape key pushed): Ignored as the action on the stopped bit triggers the necessary repetition of the transfer.
Stopped: The transfer of the remaining part of the data block is repeated.

Kind = 10:
(Paper tape reader).
Intervention: Ignored.
Parity error: Ignored. (The monitor replaces the invalid character by a SUB character).
End of document: If the number of halfwords transferred is zero, the input of an EM character is simulated.
Load point: Ignored.
Tape mark or attention: Ignored.
Read error: Ignored
Card rejected: Ignored.

Kind = 12:
(Paper tape punch).
Intervention: Ignored.
End of document: A 'change' message is sent to the parent.
Upon the receipt of the answer from the parent, the remaining part of the data block is output.
Stopped: If the end of document bit is not present, the remaining part of the data block is output.

Kind = 14:
(Line printer).
Same actions as for kind = 12.

Kind = 16:
(Card reader).
Same actions as for kind = 10.

Kind = 18:
(Magnetic tape).
Intervention: Ignored.
Parity error: The tape is repositioned and the operation is repeated up to five times. In case of output the bad spot on the tape is erased.
Data overrun: Treated as parity error (no erase).
Block length error: Treated as parity error (no erase).
Load point: Ignored.
Tape mark: The expected position on the tape is recalculated as the tape mark may indicate shift to another file, and next the position error bit is recalculated by comparing the position obtained with the one given by the monitor. If a tape mark is read, the input of an EM character is simulated.
Writing enabled: This bit is checked during the action on the stopped bit but does not in itself cause any special action.
Stopped: If the writing enabled bit is set (tape write enabled) the output transfer is repeated. Otherwise a write enable message is sent to the parent and the mounting awaited. When the answer from the parent is received the process is reserved, the tape is positioned and the transfer is repeated.
Word defect: Treated as parity error (no erase).
Position error: Not output: as parity (no erase).
Output: Hard error if anything was transferred (but the presence of the tape mark bit may cause a recalculation of the position which removes the error). If nothing was transferred, the action is as for parity.
Process does not exist: A 'mount tape' message is sent to the parent. When the answer is received, the process is reserved, the tape is positioned and the transfer is repeated.
Rejected: The process is reserved and the operation is repeated.

If anything goes wrong during a recovery action (reservation impossible, area claim exceeded, no segments available for extension of the area, etc.), the error is classified as a hard error.

Some of the utility programs have private recovery actions different from the standard ones (especially programs dealing with files which are not text files).

6.6 Errors on Current Input or Output

Hard errors on current input or output are treated in a special way because of the key role played by these files.

Hard error on the current input file:
The primary output file is selected as current output file. The 'device status' error text is printed. The primary input file is selected as current input file and the chain of stacked current input zones is abandoned. The remaining part of the FP command stack (if any) is skipped.

Hard error on the current output file:
The primary output file is selected as current output file. The 'device status' error text is printed.

Hard error on the primary input or primary output file:
FP reinitializes itself, cf. 3.3.

Appendix A. References

Part numbers in references are subject to change as new editions are issued and are listed as an identification aid only. To order, use package number.

1)	For RC8000:	
	Monitor, Part 1 System Design	PN: 991 03577
	For RC9000-10:	
	RC9000-10 System Software	PN: 991 11255
	Part of SW9910I-D, System Overview	
2)	For RC8000:	
	Monitor, Part 2 Reference Manual	PN: 991 03588
	For RC9000-10:	
	Monitor Reference Manual	PN: 991 11259
	Part of SW9890I-D, Monitor Manual Set	
3)	*Monitor, Part 3 Definition of External Processes*	PN: 991 03435
4)	*Code Procedures and Run Time Organization of ALGOL Programs*	PN: 991 11296
5)	*ALGOL8 User's Manual, Part 1*	PN: 991 11279
	Part of SW8585I-D, Compiler Collection Manual Set	
6)	*ALGOL 8 User's Guide, Part 2*	PN: 991 11280
	Part of SW8585I-D, Compiler Collection Manual Set	
7)	*BOSS User's Guide*	PN: 991 11275
	Part of SW8101I-D, BOSS Manual Set	
8)	*Parent Messages in BOSS*	PN: 991 11277
	Part of SW8101I-D, BOSS Manual Set	

Appendix B. Tables

B.1 Mode-kinds

The list contains the commonly used mode-kinds together with the abbreviations used by the ENTRY, SET, LOOKUP and SEARCH programs.

Abbr.			Mode	Kind	Use of the mode-kind	
ip			0	0	i/o via internal process	
bs			0	4	backing storage	
tw			0	10	typewriter	
tro			0	10	tape reader, odd parity	
tre			2	10	tape reader, even parity	
trn			4	10	tape reader, no parity	
trf			6	10	tape reader, flexo code	
trz			8	10	tape reader, no parity, nulls read	
tpo			0	12	tape punch, odd parity	
tpe			2	12	tape punch, even parity	
tpn			4	12	tape punch, no parity	
tpf			6	12	tape punch, flexo code	
tpt			8	12	tape punch, teletype code	
lp			0	14	line printer	
crb			0	16	card reader, binary	
crd			8	16	card reader, decimal	
crc			10	16	card reader, EBCDIC	
mt62	mtlh	mto	0	18	mag.tape, 6200 bpi, GCR, or	low speed, high density, odd
		mte	2	18	mag.tape,	low speed, high density, even
mt16	mtll	nrz	4	18	mag.tape, 1600 bpi, PE, or	low speed, low density, odd
		nrze	6	18	mag.tape, low speed,	low density, even
mt32			8	18	mag.tape, 3200 bpi, PE	
mt08			12	18	mag.tape, 800 bpi, NRZ	
	mthh		128	18	mag.tape, high speed,	high density
	mthl		132	18	mag.tape, high speed,	low density

B.2 Standard File Names and File Descriptors

The software may be expanded with a number of standard file names corresponding to commonly used files on peripheral devices. A standard file name is the name of a catalog entry containing a file descriptor (cf. chapter 5) of the file in question. The standard file names may be set up by any job, and they presume that the peripheral devices have the standard names e.g. reader, printer, punch, as it is normally the case. Most of the standard file names coincide with modekind abbreviations but this does not cause any conflict as the use of the mode-kind abbreviations is 'a private agreement' between the four programs SET, ENTRY, LOOKUP and SEARCH.

At present the following standard names may be set up:

File name	documentname	mode	kind	mode-kind abb.
term	terminal	0	8	tw
tro	reader	0	10	tro
tre	reader	2	10	tre
trn	reader	4	10	trn
trf	reader	6	10	trf
trz	reader	8	10	trz
tpo	punch	0	12	tpo
tpe	punch	2	12	tpe
tpn	punch	4	12	tpn
tpf	punch	6	12	tpf
tpt	punch	8	12	tpt
lp	printer	0	14	lp
crb	cardreader	0	16	crb
crd	cardreader	8	16	crd
crc	cardreader	10	16	crc

B.3 Contents Keys

0	Text file
1	Card text
2	Binary program to be loaded by FP i.e. a utility program, a translated ALGOL/FORTRAN program etc.
3	Directly executable program. FP itself is of this type.
4	Translated ALGOL/FORTRAN procedure.
5	Stacked zone (cf. section 4.1).
6	Program file in logical blocks with the block length in the first word of each logical block.
7	Dumped core area.
8	'Self contained' binary program, i.e. a program which can be loaded by FP, instead of FP, as well as instead of s. The program BOS, which is loaded when BOSS is started, is of this type.
9	Virtual core in ALGOL, initialized context data.
10	Files written by ACP or FTS.
11	COBOL, object program.
13	COBOL, data file.
14	Update mark in RC8000 SHIPPING
15	Program to be loaded by the RC8000 loader/paging system.
16	RTP 3502 files
17	Reserved by GIER simulator.
20	Files belonging to the bs-system.
21	Files belonging to the sq-system
22	Files belonging to the isq-system.
23	Files belonging to the sys80 system.
29	CMCL files, RC8000 TAS
30	Files compressed by the program lib
31	Reserved for various installations.
>=32	Reserved for special purposes in the ALGOL/FORTRAN system.

B.4 Error Messages

The list contains only the error messages from FP itself. An error message from a utility program has the form

***** <program name> <text>**

The meaning of the error text is found in the description of the program.

FP can output the following error messages:

*****break <cause> <instruction counter/break 10 reason>**
The break routine of FP was entered because of some severe error (see list of causes in section 3.5). BOSS restarts the job with a fresh FP and continues with the next line in the job file. With the operating system 's' you may just restart the process, and FP will reinitialize itself.

***breakpoint <testoutput>**
Private test output from a utility program. The program continues after printing of the test output.

*****device status <document name> <status word>**
Hard error on the file specified. The status bits are given by text lines (cf. section 6.4). The actual program is terminated with 'ok no' and 'warning yes'. If the file is the current input file, the current input and output files are switched back to the primary input and output files. If the file is the current output file, the current output file is switched back to primary output. If the file is primary input or primary output file, FP reinitializes itself up to 10 times before giving up and sending a FINIS message.

*****fp call <program>**
The name specified was not the name of a program file (cf. section 4.4). FP continues with 'ok no' and 'warning yes'.

*****fp cancel**
A line was cancelled during the command reading because of the appearance of a CAN character (cf. section 2.2). FP continues the command reading.

*****fp connect <program>**
The program file could not be connected (cf. section 4.4). FP continues with 'ok no' and 'warning yes'.

*****fp init troubles**
In the FP initialization (or reinitialization) the creation or connection of/to the files c or v could not succeed and the job is terminated. For a BOSS job the error message is displayed on the main console (the error message is actually a parent message).

*****fp job termination**
The job was terminated because 3 syntax errors in sequence were found in the input to FP, or FP reinitialized itself 10 times.

*****fp name <program>**
The program name was not found in the catalog (cf. section 4.4). FP continues with 'ok no' and 'warning yes'.

*****fp reinitialized**
The FP initialization was entered because of some severe error, as a result of the command INIT or because the job was answered and restarted after a break. (cf. sections 3.3 and 4.5).

*****fp size <program>**
The memory area could not hold the program or the entry point was outside the program (cf. section 4.4). FP continues with 'ok no' and 'warning yes'.

*****fp stack <last few characters input>**
Overflow of the FP command stack.

*****fp syntax <last few characters input>**
Syntax error in the input to FP cf. 2.6.

*****fp troubles with c or v**
The job was terminated because
- of hard error on current input and FP reinitialized itself more than 10 times
- troubles with creating or connecting the file c (primary output) when writing a "device status" alarm, and FP reinitialized itself more than 10 times. For a BOSS job the message is printed on the main console.

Appendix C. Index

The entries below refer to chapter or section numbers.

System Utility, User's Guide, Part One
PN: 991 11 263