# SINTRAN-4 Reference Manual

## P R E L I M I N A R Y  D R A F T

ND 60.???.01

27.03.1981

NORSK DATA internal

# TABLE OF CONTENTS

Section — Page

NOTICE.

This is a working document. Features can be changed, omitted or added suddenly and without any warning. This also means that the reader still has the possibility to suggest specification changes.

The manual is for internal use in Norsk Data. Others need special permission.

# PREFACE.

s is a preliminary proposal for the user specifications of the TRAN 4 Operating system for the NORD-100 and NORD-500 computers.

specifications have the form of a Reference Manual. Most of the ctions are specified, but more details and examples will be added.

ce the specifications are not as detailed as in the final version would be useful if the reader has some background in operating tems.

manual is ordered according to structure. In later versions the mands will be sorted alphabetically. The first part contains a user roduction which will be moved to other manuals. Implementation and figuration dependent parts are not described in any detail.

# 1. INTRODUCTION.

## 1.1. Sintran 4 design highlights

Sintran 4 is an operating system for the NORD-100 and NORD-500 computers.

The system will cover a wide range of application areas to allow integrated solutions for application functions.

High reliability and security. A program system can be split into separate protected address domains with controlled communication. The file system contains checkpoint and fallback mechanisms, and extensive access control for security. Sintran 4 is a multicomputer operating system and the configuration transparency provides easy reconfiguration if a component fails. Data streams can be split into two or more identical streams to be stored or processed at separate places.

Good user interface, for terminal users, application programs and system operation. It is easy to expand the user interface with new functions. It is also easy to implement special user interfaces as packages running under the standard system interface.

Efficiency. Especially I/O, such as discs and communication lines, are used at maximum speed. The module structure of Sintran 4 may be used for a wide range of configuration sizes, and provides easy expansion of system capacity by connecting more computers to a Sintran 4 system.

Economy. The system is written in a high-level, machine-independent system programming language (PLANC). This, together with the use of isolated modules, makes system generating, maintenance, extension and education easy, and gives good portability.

The system support multiprocessing, as well for geographical distributed processing as for configurating for a given processing power. Using the principle of communicating isolated processes, user programs and major parts of the operating system can be configuration transparent.

Application areas:
Timesharing systems
Business and office systems
Process Control Systems
Scientific Oriented On-Line Systems
Data Communication Systems
Data Acquisition systems

## 1.2. Hardware configuration

1. Single NORD-100

2. NORD-100/NORD-500 system
3. Several NORD-100 and NORD-500 in tightly coupled cluster
4. Losely coupled systems of 1, 2 and 3
5. Special processors connected

## .3. Virtual configuration

The communication between processes is standardized and looks the same whether the communicating processes are in the same computer or not. Regardless of the number of processors, the system can appear to be one homogeneous system. Only processing and transfer speeds will be different.

Several virtual homogeneous systems can be coupled together, with explicitly defined access rights.

### 1.3.1. Timesharing

A timesharing user operates from a terminal, seeing a virtual computer system which he can use independent of other users. Relationships with other users can be set up explicitly.

For simple use, a subset of the total system is described in the chapter "Introduction to timesharing".

From the terminal the user can type commands to the system, using full screen or single line editing. The parameters can either be typed on the same line as the command, or the command processor can ask for them, with an explaining text. Command names can be abbreviated. A question mark will produce a description of the current command.

Commands can also be read from a file through a command language processor. A call of a command file has the same form and properties as a command, and can be used as an extension to the standard command set. Command files can also be submitted as batch jobs.

The command language has standard facilities for structured conditions and loop control, and arithmetic and string operations.

A coroutine linkage can route a data stream through a series of subsystems.

The standard log-in procedure asks for user name and password. The user can in addition specify his own program to be successfully executed before the log-in is effective.

Log-in can be made dependent on the terminal.

Execution of a user program can be interrupted from the terminal. The program can be aborted, halted ready for restart, or status information can be extracted, with immediate restart.

The user has commands to communicate with an operator.

A debug system can let the user examine and change locations and set breakpoints in the program.

The terminal type can be defined interactively.

A histogram facility can tell where a program spends its processing time, and data transfer amounts can be measured.

A program system can be split up in domains with full protection between them. Sub-processes can be used to obtain parallel processing.

Exceptional conditions can be trapped and processed by user written programs.

A detach command can release a user task from the terminal, to let the terminal be available during a lengthy processing.

A batch system can queue jobs according to priority. Operators have commands to manipulate the job queues.

A print spooling system can queue print files and order them according to priority and format.

A mail system allows users to send messages to each other. Broadcasts can be sent to all users.

The file system provides data storage. A file is normally dynamic expandable, except when it is defined to be contiguous on the storage medium. Devices, inter-process communication ports and files use the same naming and access calls. The object name directories are hierarchial.

A file can either be an unstructured area or structured as records.

### 1.3.2. Real-time

Real-time programs have the same facilities available as timesharing programs. The main difference is that real-time programs have more explicit control over resources. Processes can be guaranteed a certain percentage of processor time, or a process can be given a fixed priority. Inter-process communication is compatible with file and device access.

### 1.3.3. System supervising

For simple configurations, only a few and simple operations are necessary, so that very little effort is necessary to keep the system up and running.

For complex configurations, more sophisticated facilities may be necessary, for administration of users and resources, configuration control, tuning and backup.

The system is designed to run without operator as far as possible.

There are several levels of system start. A cold start is provided, where the system is started as it was generated. Several system versions can be available.

A warm start is used after a controlled system shut-down or system crash. The system is started with a complete configuration setup. Necessary application initiating can be done by a predefined batch job, so a push on a "Load" button should be enough.

In case of power fail, all information will be retained, so that the system can continue afterwards. Power fail can also be simulated by a command.

The contents of physical memory can also be saved for short hardware maintenance.

For multi-processor systems, the individual processors can be started, stopped and tested.

There are commands to change parameters in the system, and to make measurements and statistics of system usage.

Users can be created and given resources. There is an accounting system to keep track of system use.

The system supervising commands are connected to privileges, which can be given to certain users. The privilege structure can be defined for each installation.

## Timesharing

### .1. Introduction to timesharing

or simple use of the system, a small set of the system facilities re needed. This can include use of standard subsystems, such as ditor, Text processing, FORTRAN and BASIC.

#### 2.1.1. Log-in

```
Escape Escape LOGIN;
USER NAME:_name;
PASSWORD: __;
OK
SYS:
```

#### 2.1.2. Commands

Command names can be abbreviated as long as they are unambiguous.

Utility commands:

HELP gives a list of all available commands. A question mark will produce an explanation on the terminal concerning the operation in progress.

# 1. INTRODUCTION.

## 1.1. Sintran 4 design highlights

Sintran 4 is an operating system for the NORD-100 and NORD-500 computers.

The system will cover a wide range of application areas to allow integrated solutions for application functions.

High reliability and security. A program system can be split into separate protected address domains with controlled communication. The file system contains checkpoint and fallback mechanisms, and extensive access control for security. Sintran 4 is a multicomputer operating system and the configuration transparency provides easy reconfiguration if a component fails. Data streams can be split into two or more identical streams to be stored or processed at separate places.

Good user interface, for terminal users, application programs and system operation. It is easy to expand the user interface with new functions. It is also easy to implement special user interfaces as packages running under the standard system interface.

Efficiency. Especially I/O, such as discs and communication lines, are used at maximum speed. The module structure of Sintran 4 may be used for a wide range of configuration sizes, and provides easy expansion of system capacity by connecting more computers to a Sintran 4 system.

Economy. The system is written in a high-level, machine-independent system programming language (PLANC). This, together with the use of isolated modules, makes system generating, maintenance, extension and education easy, and gives good portability.

The system support multiprocessing, as well for geographical distributed processing as for configurating for a given processing power. Using the principle of communicating isolated processes, user programs and major parts of the operating system can be configuration transparent.

Application areas:
Timesharing systems
Business and office systems
Process Control Systems
Scientific Oriented On-Line Systems
Data Communication Systems
Data Acquisition systems

## 1.2. Hardware configuration

1. Single NORD-100

The command language interpreter is a subsystem which accepts commands from terminal or file, using the standard dialogue processor.

## 2.2.1. Command Types

### 2.2.1.1. Compound macros (user-defined; local)

Created dynamically, these are organized in a tree structure local to the process. Each compound macro will live only as long as its surrounding environment exists. A defined macro can be saved on a file and retrieved later. Normally it should originally be typed in using an editor. The body of a compound macro consist of definitions and/or calls to other compound macros, standard-macro-calls, SINTRAN-4-calls, subsystem-calls or calls to binary executable files, with or without parameters. Or they are used as variables by other commands.

### 2.2.1.2. Standard macros (system-defined; global)

Defined once and for all, there exist few restrictions as to where in a call-chain these may be used. They are - ideally - all single commands with or without parameters, performing restricted, exact and - hopefully - understandable functions, clearly defined by their names.
There are four distinct types of standard macros :

-- basic macros : managing the lives of the compound macros.

-- control macros : controlling the internal command flow.

-- operator macros : handling arithmetic, logic and strings.

-- miscellaneous macros : featuring command debug-facilities.

### 2.2.1.3. SINTRAN-4 calls (monitor-calls; external)

Thoroughly described in other parts of this manual, these need not be handled separately here. As a group they constitute just-another-command, obeying the general syntax described in the section on Command Syntax. The expanded commands are converted to standard subroutine calls. For each of these external commands there exist a corresponding monitor call.

## 2.2.2. Compound macros as variables

Data structures are available:
-- Records can be constructed by letting a macro define a set of sub-macros to be used as variables, and then return by the standard macro DETACH. The sub-macros can then be accessed from outside the macro by using dot notation.
-- Arrays can be constructed by placing macro-calls for indices into the name. Multidimensional arrays could be declared:
<arrayname><index-1><letter><index-2>...

to overcome the problem of how to understand e.g. ARRAY123...
Macro variables may be declared as either MACROs or VARIABLEs. MACROs will get expanded at reference, VARIABLEs will return their bodies without any expansions. A PARAMETER will behave as a VARIABLE in this respect.
Macro variables may be assigned values using the standard macro SET. The assigned values may be formed using the various operator macros. Numeric strings are regarded as decimal integers, unless they are followed by the octal-number-sign "B". Alphameric constants should be included in apostrophes to avoid any erroneous attempts at expansions.

Example:

```
VARIABLE ARR0,0 % sum N first positive integers; N=0,1,..9
FOR INT 1,9; DO; PARAM Q;
   VARIABLE ARR^Q,SUM Q,ARR^DIFF Q,1 % sum(N) = N + sum(N-1)
EXITFOR;DETACH; ENDFOR
DISP ARR* % output produced :
VARIABLE ARR0, 0
VARIABLE ARR1, 1
VARIABLE ARR2, 3
.................
VARIABLE ARR9, 45
```

### 2.2.3. Command Handling

#### 2.2.3.1. Detecting macro-calls (expansion-modes)

-- Through explicit marking, using the special macro-call-character, ^, to signal a macro-call. (mark mode: END-COMMAND-MODE)

-- Through trying to expand all symbols not surrounded by apostrophes, as were they macro-calls. (macro mode: COMMAND-MODE)

#### 2.2.3.2. Search order (modified by expansion-modes)

-- Compound macros

-- Standard macros

-- External commands

-- Binary executable program files

-- Symbolic macro files

#### 2.2.3.3. Expansion order (compound macros)

-- Command name (may include other macro-calls)

-- Macro body (will return with the body itself if it is a VARIABLE or a PARAMETER, else the body will get expanded. Expandable bodies may include anything, e.g.:

-- PARAMETERs (whenever this standard macro is found, the next actual parameter is fetched from the call, expanded and saved like a local macro definition, which may be called from other commands...))

#### 2.2.3.4. Control structure (compound macros)

-- Static. All of a text-block is found and read before it is expanded. This ensures that there is no danger in having some text, e.g. inside an IF-block, expanding, e.g. to ELSE or ENDIF.

#### 2.2.3.5. Command protection (SINTRAN-4 calls)

-- The set of commands available for a user is determined by the privileges granted to him. The command language subsystem will recognize or execute the available commands only.

#### 2.2.3.6. Data streams (input/output-routing)

-- Initially the command language subsystem take input from and send output to the terminal. When ready to accept a command, it will identify itself by a name, SYS:, and wait for input.

-- Input may come from a file - specified by standard macro INCLUDE - or, at macro-calls, from a (file-)macro-body. When finished, input file will be reset to the previous one.

-- Output may go to another device or to a mass-storage-file - specified by standard macro TO. This must be specified for each call if such output routing is wanted.

-- The command language processor may be called from other subsystems, and may itself call other subsystems.

### 2.2.4. Command Syntax

#### 2.2.4.1. Formal Syntax

A command consists of a command-name and zero or more parameters.

Command syntax:
command = (s)...(sep0)command-name(sepNparameterN)...term

s = space & term = (;)cr or ;command
(x) = x is optional & x... = x may be repeated

| N=0,1,2,... | sepN: | | | | |
|---|---|---|---|---|---|
| sepN+1: | $ | ! | s(s)... | (s)...,(s)... | |
| $ | X | | | | |
| ! | | X | | | |
| s(s)... | | | X | X | |
| (s)...,(s)... | | | X | X | |

Space(s) may prefix all commands. Separators are $,!,space(s),comma, and all combinations of a comma and one or more spaces. If a separator prefix a command-name, this must be used throughout the command. If no prefix separator was given, then the separator following the command-name must be used. The above scheme show the next (N+1) separator as a function of the current (N) one. As shown, space(s) and comma are fully interchangeable as separators.

Example:

```
SYS: VAR VAR1;
SYS: VAR$VAR NU 2;
SYS: SET VAR1,10
SYS:   SET,VAR1   10
SYS: SET   VAR1;
SYS:   SET$VAR NU 2$VAR1
SYS: SET!VAR N;
SYS:   $VAR NU;
SYS:  SET$VAR NU 2$;  !VAR N
```

### 2.2.4.2. Parameter Values

If an empty string is given as a parameter, a default value is used - even if the empty string resulted from a macro-expansion. Missing parameters get default values if semicolon terminates the command, they will be asked for if carriage return (cr) was the terminator. If separated by semicolons, more than one command may be given on the same line.

A command may be executed more than once using different parameter values, simply by stating the different values in the parameter-list. A special parameter-repeat separator -- / (slant) -- is used to separate the various values to be used for the parameter.

### 2.2.4.3. Special Characters

The command-name may consist of several parts separated by hyphens, and can normally be abbreviated as long as it is unambiguous. Any combination of one or more visible characters are allowed, with the exception of a few special characters:

& % * . ' / ^ ?

& (ampersand) denotes line continuation. The rest of the current line may be used for comments. To make an ampersand a part of a name, two ampersands must be included.

% (percent-sign) denotes start of comment, i.e. the rest of the line will be a comment. To make a percent-sign a part of a name, two percent-signs must be included.

* (asterisk) replaces any character in a name, and also signals call repetition, using all matching names in their order-of-creation. Thus an asterisk cannot be a part of any unambiguous name.

. (dot) denotes levels of naming. The name to the right of a dot is a subsidiary to the one to the left. This reflect the current call-chain. Thus a dot cannot be a part of any single name.

' (apostrophe) starts a string. A string will never be searched to be a macro. A string may consist of any characters whatsoever. Though - to make an apostrophe a part of a string it must be specified twice. String is ended by another apostrophe.

/ (slant) denotes repeated calls. Names separated by slants make up a value-list for the command or parameter in question. To make a slant a part of a name, two slants must be specified.

^ (circumflex or up-arrow) is initially the macro-call-character. The current macro-call-character at any time -- be it circumflex or any other -- must be specified twice to be part of a name.

? (question-mark) will force a syntax-description of the command currently being processed to be output to the terminal. To make a question-mark a part of a name, two marks must be specified. Command-processing will continue.

## 2.2.5. Standard Macros

### 2.2.5.1. Basic Macros

#### 2.2.5.1.1. MACRO ... ENDMACRO

Function:
A local macro is created, and the following text up to the matching ENDMACRO defines the macro body.

Parameters:
<name> - exclusive name identifying the macro. DEF = NO*DEFAULT

Rules:

<name> may not contain asterisk(s).

Example:
MACRO LOCAL-MACRO; ENDMACROcr % empty macro-definition

### 2.2.5.1.2. FILE-MACRO

Function:
A local macro is created, and the contents of the specified file will be used as the macro body.

Parameters:
<name> - exclusive name identifying the macro. DEF = NO*DEFAULT
<filename> - name of mass-storage-file containing the desired macro body. DEF = NO*DEFAULT

Rules:
<name> may not contain asterisk(s). The body must be contained in one file.

Example:
FILE-MAC FI-LOC-MAC,SAVED-DEFINITIONcr

### 2.2.5.1.3. NEW-MACRO

Function:
A local macro is created, and an already existing macro defines the macro body.

Parameters:
<name> - exclusive name identifying the macro. DEF = NO*DEFAULT
<oldname> - name of macro containing the body to be copied. DEF = NO*DEFAULT

Rules:
<name> may not contain asterisk(s). <oldname> must be unambiguous. The new macro will be a true macro, regardless of the old. See also VARIABLE and PARAMETER.

Example:
NEW,NEWIE,OLDIEcr

### 2.2.5.1.4. VARIABLE

Function:
A local 'variable' (i.e. non-expanding macro) is created and an initial value given.

Parameters:
<name> - exclusive name identifying the macro. DEF = NO*DEFAULT
<value> - initial value to be expanded. DEF = nil

Rules:
<name> may not contain asterisk(s). Command-iteration allowed for last parameter. The concatenated value-expansions will constitute the initial value. New variables equaling old ones must be defined using this call with the name of the old variable as the <value>-parameter.

Example:
VARI NEWVAR,OLDVARcr % new will initially equal the old

### 2.2.5.1.5. SET, CONCATENATE

Function:
The value will be expanded and replace(SET) or appended(CONC) to the old macro body.

Parameters:
<name> - name of macro to be changed. DEF = NO*DEFAULT
<value> - value to be expanded. DEF = nil

Rules:
Command-iteration allowed for both parameters. Changes consist of the concatenated value-expansions.

Example:
SET MANUAL-COPY*,CHAPTER*; CONC MANUAL-COPY*,APPENDIX*cr

### 2.2.5.1.6. BODY

Function:
The macro body is returned without expansion.

Parameters:
<name> - name of macro to be copied. DEF = NO*DEFAULT

Rules:
Command-iteration allowed. The concatenated macro-bodies are returned.

Example:
BODY$CHAPTER 10cr % name contain legal separator "space"

### 2.2.5.1.7. CALL

Function:
The contents of a file is used as a macro body. The body gets expanded, the text being echoed to the current output-file. Body is terminated by and local macros normally deleted at "end-of-file".

Parameters:
<filename> - name of mass-storage-file containing the desired macro body. DEF = NO*DEFAULT

(<parameters>) - non-default PARAMETER-values.

Rules:
The command name CALL can be omitted. The filename must be unambiguous.

Example:
CALL FILE-MC-BODY,PAR1cr % or simply: FILE-MC-BODY,PAR1cr

### 2.2.5.1.8. INCLUDE

Function:
Command input will come from the specified file until "end-of-file" is found. The text is echoed to the current output-file. No change of environment is implied; thus local macros will survive.

Parameters:
<filename> - name of mass-storage-file containing the desired command input. DEF = NO*DEFAULT
(<parameters>) - non-default PARAMETER-values.

Rules:
"Infinite" include-levels allowed. The filename must be unambiguous.

Example:
INCLUDE .FILE-INPUTcr

### 2.2.5.1.9. PARAMETER

Function:
A formal parameter is defined like a local macro, and the body is fetched from the call. The leadtext is used for prompting, and the default value is used if no parameter value is given.

Parameters:
<name> - exclusive name identifying the macro. DEF = NO*DEFAULT
<default> - alphanumeric string. DEF = nil
<leadtext> - alphanumeric string. DEF = value

Rules:
<name> may not contain asterisk(s). A PARAMETER is non-expandable; returning its current body wherever specified.

Example:
PARAMSETER$DU OGSAA$HAVREGRYNcr % default contain legal separator "space"

### 2.2.5.1.10. IO

Function:

The specified file will receive the output resulting from the next command or macro-expansion.

Parameters:
<filename> - name of output-device or mass-storage-file.
DEF = NO*DEFAULT

Rules:
Terminal output is presumed for commands not immediately preceded by TO.

Example:
```
TO  L-P;MCCALL$PAR1;cr
```

### 2.2.5.1.11. EXPAND-TO

Function:
The specified file will receive the external commands resulting from the next command- or macro-expansion. These external commands will not be executed.

Parameters:
<filename> - name of output-device or mass-storage-file.
DEF = NO*DEFAULT

Rules:
If the specified file is a mass-storage-file it may later be used as a command input file.
This command is very useful to check that a macro expands to what the user ment it to -- without risking any of the hazards involved if it does not...

Example:
```
EXP-TO S4-MODE;MC-COMPILE SOURCE-P,LIST-P;cr
```

### 2.2.5.1.12. DETACH

Function:
If met during macro-expansion, execution will stop and control given back to the caller without any local macros being deleted. These can be accessed from the calling levels using dot notation.
If met during processing of a FOR- or DO-loop, macros local to the loop will be made local to the surrounding environment on exit, thus dot notation is neither required nor possible in this case.
Local PARAMETERs will in both cases be deleted. Local VARIABLEs are handled as ordinary macros.

Parameters:
None

Rules:
DETACH must be included in every macro in the call-chain from the inner 'point-of-creation'-level to the level next high to the outermost 'point-of-reference'.

Example:
MACRO A; VARIA B,some-value; DETACH; ENDM; A; DISP A.Bcr

### 2.2.5.1.13. SAVE-MACRO

Function:
The specified macro body is copied onto the specified file.

Parameters:
<filename> - name of mass-storage-file to contain the desired definition. DEF = NO*DEFAULT
<name> - name of macro to be saved. DEF = NO*DEFAULT

Rules:
Command-iteration allowed for last parameter, but then the concatenated macro bodies will be copied onto the file as one body...

Example:
SAVE-M SAVED-MACROS,*cr % all macros copied onto a common file as one body

### 2.2.5.1.14. DELETE-MACRO

Function:
The specified macro definition is deleted. File-macros are closed.

Parameters:
<name> - name of macro to be deleted. DEF = NO*DEFAULT

Rules:
Command-iteration allowed. Command is irreversible.

Example:
DELE-M,*;cr % All local macros are deleted

### 2.2.5.1.15. < ... > (quote)

Function:
The text following the < up to the matching ;>; or ;>cr or cr>; or cr>cr will be copied without expansion.

Parameters:
None

Rules:
Remember to insert current separator between < and the text!

Example:
< This is a possible text.;>cr

#### 2.2.5.1.16. COMMAND-MODE, END-COMMAND-MODE

Function:
Decides whether all symbols are to be checked to be macros, or only those preceded by the call-mark-character (initially: ^ ).

Parameters:
None

Rules:
The call-mark-character has of course effect in both modes. Symbols preceded by this character will never be checked to be external commands.

Example:
```
... END-C-Mcr ... ^COMM-Mcr ...
```

### 2.2.5.2. Control Macros

#### 2.2.5.2.1. IF ... ELSIF ... ELSE ... ENDIF

Function:
If a specified condition expands to true, i.e. a non-emoty string, the following text up to a matching ELSIF, ELSE or ENDIF is found, and the rest of the logical construct skipped. If no true condition is found the text in the optional ELSE-sequence will be used. When ENDIF is found, the found text will get expanded.

Parameters:
<condition> - string, macro or logical construct. DEF = false

Rules:
IF-constructs may be nested to "any" depth.

Example:
```
IF GT LOC-A,LOC-B; ... ELSIF EQ L-A,LO-B; ... ELSE; ...
ENDIF
```

#### 2.2.5.2.2. FOR ... DO ... ENDFOR

Function:
The text thus enclosed will be iterated until the parameter-list is exhausted or a WHILE-condition giving false have been found. PARAMETER-calls will read from the parameter-list. The loop may be exited directly through ENDFOR, or through a special-exit-sequence: EXITWHILE or EXITFOR. IF both are used, EXITWHILE must precede EXITFOR. Local macros created within the loop will normally get deleted on exit.

Parameters:
(<p-list>) - list of parameter-values to be read by in-loop PARAMETER-calls.

Function:
More than one PARAMETER may be used. Each must have its own list, separated by commas. Values in a list must be separated by slants. The last PARAMETER declared will run the fastest.

Example:
```
FOR PRI/MED/ULT,1/2/3; DO
PARAM A-P; PARAM B-P; ...
WHILE NE A-P,B-P; ...
EXITWHILE; VAR EXIT-M,'broken'; DETACH
EXITFOR; VAR EXIT-M,'all through'; DETACH
ENDFOR
```

### 2.2.5.2.3. DO ... ENDDO

Function:
The text thus enclosed will be iterated until a WHILE-condition giving false have been found. PARAMETER-calls will read from the parameter-list of the surrounding environment. The loop may be exited directly through ENDDO, or through the special-exit-sequence EXITWHILE. Local macros created within the loop will normally get deleted on exit.

Parameters:
None

Rules:
If a parameter-list get exhausted, values will be asked for. The only way to exit from the loop is through a WHILE-condition expanding to false (empty string), or because of some error-condition not handled locally.

Example:
```
DO; PARAM P1; ... WHILE LE P1,99; ... EXITWHILE; DETACH;
ENDDO
```

### 2.2.5.2.4. WHILE

Function:
If the condition expands to an empty string, the execution of the first enclosing DO-loop, FOR-loop or macro body will be terminated.

Parameters:
<condition> - string, macro or logical construct. DEF = false

Rules:
Always remember to include WHILE-commands in DO-loops, this being the only non-abortal means of exiting from such a loop...

Example:
```
WHILE NE 1,2cr % Bad example of DO-loop controller...
```

#### 2.2.5.2.5. ON ... ENDON

Function:
The text thus enclosed will get expanded if an error occurs and the exception handling facility is enabled.

Parameters:
None

Rules:
ON-constructs may not be nested. See also ENABLE-, DISABLE- and RAISE-EXCEPTION.

Example:
```
ON; DU-M MACROS; DU-S STRINGS; ENDON % All current macros
and defined strings are 'dumped' at enabled errors.
```

### 2.2.5.3. Operator Macros

#### 2.2.5.3.1. SUM, DIFFERENCE, PRODUCT, QUOTIENT

Function:
The arithmetic sum, difference, product or (truncated) quotient of two numeric integer strings is returned. A possible minus sign is included in the resulting string.

Parameters:
<first-operand> - numeric integer string or macro-call. DEF = 0
<last-operand> - numeric integer string or macro-call. DEF = 0

Rules:
A divide-by-zero will give hardware-dependent results. All defaults give zero return.

Example:
```
SET A1,SUM A2,PROD A3,QUOT A4,DIFF A5,A6cr
% A2+(A3*(A4/(A5-A6)))=:A1
```

#### 2.2.5.3.2. AND, OR

Function:
The logical product or sum of two strings is evaluated. Logical product (AND) give non-empty string if both parameters are non-empty, otherwise empty. Logical sum (OR) give empty string if both parameters are empty, otherwise non-empty. If the operation results in a non-empty string, a

space (true) is returned, otherwise nothing (false).

Parameters:
<first-operand> - string or macro-call. DEF = nil
<last-operand> - string or macro-call. DEF = nil

Rules:
All defaults give false results.

Example:
```
WHILE AND,L1 OR L2,L3cr % L1 and (L2 or L3) !!!!!
```

### 2.2.5.3.3. NOT

Function:
The logical negation of the operand string is evaluated. If the operand is an empty string, a space (true) is returned, otherwise nothing (false).

Parameters:
<operand> - string or macro-call. DEF = nil

Rules:
Default operand give true result.

Example:
```
WHILE NOT;cr % default operand -- WHILE always true
```

### 2.2.5.3.4. GT, GE, EQ, NE, LE, LT

Function:
The relational operation true-greater(GT), greater-equal(GE), equal(EQ), not-equal(NE), less-equal(LE) or true-less(LT) is performed. If both strings are numeric integers, the numbers will be compared; otherwise the strings will be compared according to their alphameric (-ASCII-) order. True or false will be returned.

Parameters:
<first-operand> - alphanumeric string or macro-call. DEF = 0
<last-operand> - alphanumeric string or macro-call. DEF = 0

Rules:
All defaults give true return for GE,EQ and LE; false for GT,NE and LT.

Example:
```
WHILE NOT EQ L1,L2cr % while NE l1,l2
```

### 2.2.5.3.5. ABBREVIATION

Function:
If the first string is equal to or an abbreviation of the

second string, a space (true) is returned; otherwise nothing (false).

Parameters:
<first-operand> - alphanumeric string or macro-call. DEF = nil
<last-operand> - alphanumeric string or macro-call. DEF = nil

Rules:
Default operands give true result.

Example:
IF ABBR S-ST,F-ST; SET S-ST,F-ST; ENDIF % Note: macro F-ST will be expanded before assignment is executed

### 2.2.5.3.6. INCHARACTER

Function:
A character is read from the specified string. The read(-byte-)pointer is increased by one. The character read is returned.

Parameters:
<string> - name of a current macro. DEF = NO*DEFAULT

Rules:
Command-iteration allowed. One character will be read from each matching string, and the resulting, concatenated string finally returned. The byte-pointer will stay unchanged for strings in the current call-chain.

Example:
SET CURRENT-CHARS,INCHAR *cr % read one character from all current macros.

### 2.2.5.3.7. READPOINTER, WRITEPOINTER

Function:
The current read/write(-byte-)pointer of the specified string is read, and its value is returned as a numeric string. The pointer remains unchanged.

Parameters:
<string> - name of a current macro. DEF = NO*DEFAULT

Rules:
Command-iteration is allowed, though seemingly of no practical value.

Example:
WHILE NE READPOINTER MC-S,WRITEPOIN MC-Scr

### 2.2.5.3.8. SET-READPOINTER, SET-WRITEPOINTER

Function:
The read/write(-byte-)pointer of the specified string is set to the specified value.

Parameters:
<string> - name of a current macro. DEF = NO*DEFAULT
<position> - numeric integer string or macro-call. DEF = 0

Rules:
Command-iteration allowed for both parameters, though iterating the second parameter can have no(?) practical value. Setting the byte-pointers of macros belonging to the current call-chain is strongly dis-advised!

Example:
SET-READPOINTER *,0cr % All macros are reset (if God allow...)

### 2.2.5.3.9. CHARACTER

Function:
The character corresponding to the specified ASCII-number is returned.

Parameters:
<number> - numeric integer string or macro-call. DEF = 32

Rules:
The number must lie in the range 0 through 177B.

Example:
CHAR;cr % A space will be output to current output-file

### 2.2.5.3.10. NUMBER

Function:
The ASCII-number corresponding to the specified character is returned.

Parameters:
<character> - single character or macro-call. DEF = space

Rules:
The parameter should not comprise or expand to more than a single character.

Example:
NUM;cr % The ASCII-value for space will be output to the current output-file

### 2.2.5.3.11. INTERVAL

Function:
The integer values in the range set up by the specified low- and high bounds are returned as an ordered list of increasing values separated by commas. If one or both bounds are specified octally, the list will be made up of octal values only.

Parameters:
<low bound> - numeric integer or macro-call. DEF = 0
<high bound> - numeric integer or macro-call. DEF = 0

Rules:
Both bounds must lie in the range 0 through 177B.

Example:
FOR INT 65,90; DO ... ENDFOR % repeat the sequence for each value in the range 65 through 90

## 2.2.5.4. Miscellaneous macros

### 2.2.5.4.1. HELP

Function:
The standard macros and external commands that match the specified command will be written to the current output file.

Parameters:
<command> - command-abbreviation to be searched for. DEF = nil

Rules:
If parameter is empty, all existing commands will be output. Standard macros will be listed first.

Example:
HELP HELcr % output: HELP

### 2.2.5.4.2. ENABLE-EXCEPTION, DISABLE-EXCEPTION, RAISE-EXCEPTION

Function:
Whenever exception-handling is enabled at errors, control will be given to the last ON-block defined in the call-hierarchy. When disabled, an error flag is set, whilst execution continues from point-of-occuring-error. Programmed errors are triggered using RAISE-EXCEPTION, and will be handled as a normal exception-condition.

Parameters:
None

Rules:
Exception-handling must be enabled/disabled at every call-level. Default is exception-enable.

Example:
IF MC-ERRCOND; ENA-EX; RAIS-EX; ENDIF

### 2.2.5.4.3. LIST-EXPANSION, END-LIST-EXPANSION

Function:
The result from the next macro-expansion(s) will be listed on the current output file, until LIST-EXPANSION is reset.

Parameters:
None

Rules:
Default is nolist. Space should be avoided as parameter-separator when in-list-mode.

Example:
LI-EXP; CALL M-ABBR-EXPANSIONScr

### 2.2.5.4.4. SET-CALL-MARK

Function:
The call-mark-character is redefined.

Parameters:
<character> - single character or macro-call. DEF = ^

Rules:
Any visible character - including space - will do.

Example:
SET-C-MARK @cr

### 2.2.5.4.5. BREAK

Function:
A breakpoint is set at the start of the specified macro. When breakpoint is reached during macro-expansion, execution will stop, 'BP:<name>' be output to the terminal, and control given to the user.

Parameters:
<name> - full name of existing macro or macro yet-to-be-defined. DEF = NO*DEFAULT

Rules:
The specified macro-name must not be abbreviated! One breakpoint only will exist at any given time, this may be redefined by another BREAK or BREAK-RETURN. It is not

possible to break at a VARIABLE or PARAMETER.

Example:
BREAK FULLY-NAMED-MACROcr

### 2.2.5.4.6. BREAK-RETURN

Function:
A breakpoint is set at the end of the current -- idle -- macro, and execution will be immediate resumed (i.e. MACRO-RUN implied). When breakpoint is reached, execution will stop, 'BR:<current-macro>' be output to the terminal, and control given to the user.

Parameters:
None

Rules:
Macros local to current-macro are available for inspection using dot notation. 'DISPLAY <current-macro>.*cr' will thus reveal all macros locally defined.

Example:
BRE-Rcr

### 2.2.5.4.7. RESET-BREAK

Function:
Stepping is reset and breakpoint is removed.

Parameters:
None

Rules:
Default modes are nostep and nobreak.

Example:
RE-BRcr

### 2.2.5.4.8. MACRO-RUN

Function:
Execution will continue from last breakpoint reached.

Parameters:
None

Rules:
Note: This is not the correct command to initiate a macro-expansion! This is done through stating the macroname as a command.

Example:
MA-Rcr

### 2.2.5.4.9. STEP

Function:
Control will be given to the terminal for each command. Immediate carriage-return will cause continuation to the next step; non-empty strings will be executed as commands and control remain with the terminal-user.

Parameters:
<count> - number of commands to pass by before stepping is started. DEF = 0

Rules:
Default mode is nostep. List-expansion is implied when in step-mode. RESET-BREAK will remove stepping.

Example:
STEP;cr % immediate single-step triggered

### 2.2.5.4.10. ACTIVE-MACROS

Function:
The names of all macros in the current call-chain are returned.

Parameters:
None

Rules:
Names presented as the inverse to order-of-calling.

Example:
A-MAcr

### 2.2.5.4.11. DISPLAY

Function:
The body of the specified macro is output to the current output-file. For macros the header and the trailer will be 'MACRO <name>cr' and 'ENDMACROcr' respectively. These are not part of the body. Neither are the 'VARIABLE <name>, ' preceding VARIABLEs or 'PARAMETER <name>, ' preceding PARAMETERS.

Parameters:
<name> - name of macro to be displayed. DEF = NO*DEFAULT

Rules:
Command-iteration is allowed. Each body output will have its own pair of headers/trailers.

Example:
DISP,*cr % all macros local to the current call-level are

output.

### 2.2.5.4.12. DUMP-MACROS

Function:
All macros currently in the macro-definition-tree are dumped in symbolic form onto the specified file. Some useful debug-information are included. To save a macro for later use, use the SAVE-MACRO command!

Parameters:
<filename> - name of file to receive 'dump'. DEF = NO*DEFAULT

Rules:
Primarily a debugging-tool... Macros in the current call-chain will not be affected.

Example:
DU-M MACRO-TREEcr

### 2.2.5.4.13. DUMP-STRINGS

Function:
All strings currently in use are dumped onto the specified file, together with some useful debug-information.

Parameters:
<filename> - name of file to receive 'dump'. DEF = NO*DEFAULT

Rules:
Primarily a debugging-tool... Strings belonging to macros in the current call-chain will not be affected.

Example:
DU-S TERMcr

### 2.2.5.4.14. RECOVER

Function:
The program on the specified binary file is loaded and started.

Parameters:
<filename> - name of binary file. DEF = NO*DEFAULT

Rules:
The command name RECOVER can be omitted.

Example:
RECOVER MACROcr % is equivalent to: MACROcr

#### 2.2.5.4.15. PLACE

Function:
The program on the specified binary file is loaded, but not started.

Parameters:
<filename> - name of binary file. DEF = NO*DEFAULT

Rules:
To be followed by RUN

Example:
PLACE MACROcr

#### 2.2.5.4.16. RUN

Function:
The current binary program -- if any -- is started.

Parameters:
None

Rules:
Program must have been PLACEd earlier.

Example:
RUNcr

## 2.3. Logging in and out

### 2.3.1. Logging in

The terminal is activated by pressing the "escape" key twice and giving the function name LOGIN.

Then the system will write USER NAME:, waiting for the user to type his name. Then it will write PASSWORD:, waiting for the user to type his password (without echo).

If the correct user name and password has been given, the command processor will be called; otherwise the log-in procedure will start over again. If the log-in procedure has not been completed within two minutes, it will be terminated, and the terminal will be released. If five unsuccessful log-ins have been tried, the terminal will be blocked for five minutes.

This is the normal procedure, but there exists some options and alternatives.

Before "USER NAME:", the system can execute an installation dependent routine. It can for example ask for computer system name, so that the terminal can be routed to the wanted computer

system before logging in; or it can write some message to the user.

Normally, the log-in is independent of the terminal. However, the system can associate user privileges to a terminal, so that only a certain set of users are able to log in.

If the user does not rely on the password protection, he can define his own program to be executed immediately after the password is given. If this program terminates with an error exit or is interrupted from the terminal, the log-in is unsuccessful. The command is SET-LOGIN-PROGRAM <file name>. There is a corresponding system command
DELETE-LOGIN-PROGRAM <user name>.

The user can also specify a program to be executed when the log-in-procedure is finished:
SET-INITIAL-PROGRAM <file name>

The user can change his password by using the command SET-PASSWORD <old password> <new password>

## 2.3.2. Logging out

The user will be logged out by giving the command LOGOUT. Some accounting information will be written.

The logout command is defined as a trap condition, so that the user can write a trap handler to override the system logout handling. Only users with a certain privilege can do this. This means that terminals can be dedicated to specific applications where the standard login and logout is not used.

## 2.3.3. The PAUSE command

PAUSE
To continue using the terminal, the password must be given, or the user may be logged out.
Example:

```
PAUSE
USER GUEST, PAUSE FROM 17.25
LOG OUT? NO
PASSWORD
OK
SYS:
```

## 2.3.4. Users and systems

The user name has the syntax
<system name>. <user name>
Within one system, the user name is unique, and the system name can be omitted. If several systems are connected in a network, the system name will be a unique identification of the system.

Internally the full name, including system name, will be used.

## 2.3.5. Management of users and terminals

All users known to a system are identified by a symbolic user name. Terminals are are also identified by a symbolic terminal name,and in addition they have a low level terminal id.Associated with each user and each terminal are certain attributes,constituting a user profile and a terminal profile.Some of the attributes in the user profile may be set or changed by the user himself,while others require supervisor or operator privilege to be changed.

All terminal profile management is restricted to users with operator or supervisor privileges.

### 2.3.5.1. User management commands

#### 2.3.5.1.1. Public commands

These commands may be used by any user,and they effect only the profile of the user issuing them.

##### 2.3.5.1.1.1. SET-DEFAULT-NAME-ENVIRONMENT

<name string> <=status>
This command defines the name environment that will be effective after subsequent LOGINs with the user name.

##### 2.3.5.1.1.2. SET-LOGIN-PROGRAM

<file name> <=status>
This command defines a file (internally represented in full context) that will be loaded and executed during LOGIN.If the program terminates by an ERROR-EXIT, the LOGIN is considered to be unsuccessful.

##### 2.3.5.1.1.3. DELETE-LOGIN-PROGRAM

Clears a previous SET-LOGIN-PROGRAM.

##### 2.3.5.1.1.4. SET-INITIAL-COMMAND

<command string> <=status>
Defines a command string to be given to the command processor after LOGIN has been successfully completed.It may for example start a subsystem, execute a macro file,or whatever is needed.It must be remembered that the DEFAULT-NAME-ENVIRONMENT will be effective when this command is executed.

2.3.5.1.1.5. DELETE-INITIAL-COMMAND

Clears a previous SET-INITIAL-COMMAND.

2.3.5.1.1.6. SET-PASSWORD

<old password> <new password> <=status>
Defines or redefines the password for the user.

2.3.5.1.2. Restricted user profile management commands

The following commands may only be used by users with operator or supervisor privileges.They may effect the profile of any user in the system.

2.3.5.1.2.1. CREATE-USER

<user name> <=status>
Creates a brand new user profile with all default values for the attributes. If a user with <user name> is already defined,its attributes are not altered ,but an error status is returned.

2.3.5.1.2.2. DELETE-USER

<user name> <=status>
Deletes a user profile.

2.3.5.1.2.3. RENAME-USER

<user name> <=status>
Associates a new name with an existing user profile.

2.3.5.1.2.4. CLEAR PASSWORD

<user name>
Sets the password of a user to an empty string,making it possible to give only a carriage return when asked for password during LOGIN.

2.3.5.1.2.5. SET-USER-PROFILE

<user name>
This a general command to inspect and set all attributes of a user profile,except the password.The user may now inspect/alter whatever attribute he wants by giving its name followed by a slant (/).The current value will be displayed, and a new value may be entered,followed by a carriage return.Carriage return alone retains old value.Then the name and old value of the next attribute in a predefined sequence will be displayed.The user may thus step through all attributes sequentially,go to a new by giving its name followed by a slant,or end the whole

command by giving a double slant.A carriage return entered instead of the first attribute name ,starts at the beginning of the predefined sequence.
The attribute names are :

DEFAULT-NAME-ENVIRONMENT

INITIAL-COMMAND

LOGIN-PROGRAM

FIXED-ACCOUNT-ID

FIXED-SYSTEM

FIXED-PROCESSOR

USER-CLASS

TIME-LIMIT ( number of cpu seconds LOGIN and LOGOUT )

PRIVILEGE : OPERATOR
.....
.....
.....
PRIVILEGE : SUPERVISOR

### 2.3.5.2. Terminal management commands

All terminals in a system should have a symbolic terminal name. In addition,all terminals on which it should be possible to log in,must have a terminal profile. All these commands are restricted.

#### 2.3.5.2.1. DEFINE-TERMINAL

<terminal id> <terminal name> <=status>
Defines a terminal name.For the contents of <terminal id >,see system documentation.

#### 2.3.5.2.2. DELETE-TERMINAL

<terminal name> <=status>
Deletes terminal name definition.This also implies that it will be impossible to log in on the terminal.

#### 2.3.5.2.3. SET-TERMINAL-PROFILE

This a general command to inspect and modify all attributes in a terminal profile.It is used the same way as SET-USER-PROFILE ( see chapter xx.xx.xx). The attribute names are :

FIXED-USER-NAME ( if non empty : don't ask for user name )

INITIAL-COMMAND

FIXED-ACCOUNT-ID

MINIMUM-USER-CLASS

FIXED-SYSTEM ( only applicable if
SYSTEM-SELECTION-STRATEGY = 1 )

FIXED-PROCESSOR ( only applicable if
PROCESSOR-SELECTION-STRATEGY = 2 )

SYSTEM-SELECTION-STRATEGY = 0 : don't care
= 1 : use fixed system
= 2 : ask user
= 3 : this terminal is strictly for local use, no rerouting during LOGIN even if user profile should specify it,and no CREATE-PROCESS to foreign systems.

PROCESSOR-SELECTION-STRATEGY
= 0 : don't care
= 1 : selection based upon actual load
= 2 : use fixed processor
= 3 : ask user

#### 2.3.5.2.4. DELETE-TERMINAL-PROFILE

<terminal name> <=status>
Deletes the terminal profile of a terminal,making it unavailable for timesharing use.

### 2.3.6. Program interrupt

The program can be interrupted from the terminal for different reasons. This can be done by typing "escape" twice and a function name. If a third "escape" or a carriage return is typed instead of function name, the function name will be asked for.

The functions are:

LOGIN - log in on the terminal; will give error message if the terminal is already in use.

WAIT - the user program will be stopped and put in a waiting state, without closing files or other interference with the program context.The terminal stream will be given back to the parent process.If the terminal is held by the top level process,this command will have no effect. The CONTINUE command will resume the program where it left.

ABORT - the user program will be stopped and the files closed. The CONTINUE command will restart the program in a predefined restart address.

RETURN - the terminal will be given back to the parent process (corresponding to "RUBOUT" for the REMOTE function in SINTRAN III). The process that looses control of the terminal stream,will still be active, but it will be hung in a waiting state if it asks for input from the terminal. This command will have no effect if entered while control is at the top level process.

OUTPUT - discard terminal output. A second <esc><esc>OUTPUT will resume normal output.

STATUS - give status information on the program without stopping it.

USER - the user can use it as a trap; ignored by the system.

## 2.4. Subsystem control

### 2.4.1. PLACE-SUBSYSTEM

<file name>, <object index for domain>, <capability index>

The subsystem will be placed on the given domain. An indirect segment will be placed in the callers domain, giving subroutine access to the new subsystem. The new subsystem can have higher privileges than the caller. The file type should begin with :PROG, followed by computer type; for example :PROG-N500.

### 2.4.2. START-SUBSYSTEM

<capability index>, <routine number>

The subsystem can have several entry points. Normally the first one will be an initial entrypoint and the second a restart entry point.

### 2.4.3. EXIT

A trap condition will be raised, and control will be given to the nearest trap handler.

### 2.4.4. ERROR-EXIT

<reason>

A trap condition will be raised. The parameter <reason> can be read by the trap handler.

### 2.4.5. SUBSYSTEM-STATUS

<flag no.> <=flag value>

The system maintains a set of flags, in the range 0 through 31. The first 16 flags can only be set by the system, while the rest can be set by the user. The Boolean value is returned. The standard system flags are:

0: Last subsystem had error exit
1: Batch processor is running
2: In macro expansion

#### 2.4.6. SET-SUBSYSTEM-STATUS

<flag no.> <flag value>

The <flag no.> can be in the range 15 through 31. All flags are cleared on log-in.

#### 2.4.7. EXECUTE-COMMAND

<command string>

The string is sent to the command processor, where it will be interpreted and executed.

### 2.5. Measurements and statistics

#### 2.5.1. SET-HISTOGRAM

<start address> <max. address> <number of parts>

The address space interval will be divided into up to 64 parts. The processor usage percentage for each part will be recorded.

#### 2.5.2. HISTOGRAM

<=result>

The results will be returned.

#### 2.5.3. STOP-HISTOGRAM

The recording will be stopped.

#### 2.5.4. RESTART-HISTOGRAM

The recording will be restarted.

#### 2.5.5. OBJECT-STATISTICS

<object index> <=result>

Usage information will be returned - such as number of bytes and records transferred since OPEN, and busy percentage.

### 2.5.6. TIME

<=time>

The point of time (milliseconds since system start) is returned.

### 2.5.7. TIME-USED

<=processor time> <=terminal time>

The processor time and terminal time used since login will be returned (milliseconds).

### 2.5.8. CLOCK

<=ms> <=sec.> <=min.> <=hour> <=day> <=month> <=year>

The time of the day and the date will be returned.

### 2.5.9. SYSTEM-TITLE

<=string>

The name and version of the operating system will be returned.

### 2.5.10. PROCESS-STRUCTURE

<object index for process> <=result>

Information on the process tree will be returned. Default value is current terminal task (can be made restricted).

### 2.5.11. DOMAIN-STRUCTURE

<=result>

Information on the domains of the current process will be returned.

### 2.5.12. WHO-IS-ON

<=result>

The terminal identification and the user name of the logged-on users will be listed (can be made restricted).

# 3. Object naming and input/output system

## 3.1. Object naming system

### 3.1.1. Introduction

The purpose of the object naming system is to enable the user to associate symbolic names to resources in the SINTRAN IV system. Resources that may be given names by the object naming system are called objects. Examples of objects are files, ports, input/output devices, processes, etc.

The objectname as argument and returns an "object index" to the user. The object index may be used by the user for subsequent operations on the object. When the user no longer needs this "handle" on the object, he informs the system about this by executing a CLOSE call taking the object index as argument.

To enable the user to structure the object name directory into groups of objects related in some way, a directory may contain subdirectories up to 10 levels. This is done conceptually by saying that a directory is also an object type. The relationship between object in a subdirectory is entirely up to the user to decide (or the system supervisor for the uppermost levels). Examples of relationships between objects in a subdirectory are: Objects owned by one single user, objects used on one project, objects used by a group of users, the section of a book, the source files of a program system, etc.

### 3.1.2. Object naming structure

An object name has the following general syntax:

```
<name>   ::= <string>:<type>!<string>.<name>
<string>::= (1 to 32 alphanumeric characters)
<type>   ::= <string>
```

Examples:

```
OPR.KETIL.FILE-SYSTEM:SYMB
OPR.TRYGVE.SINTRAN-IV:BRF
COMM.MARTIN.XMSG.1:SYMB
COMM.MARTIN.XMSG.2:SYMB
```

The string to the right of the rightmost dot identifies an object while the other parts of the full name identify directories containing names of lower level directories or objects. Thus, a full object name forms the path through a directory hierarchy ending up with the wanted object.

A directory does not necessarily correspond to a physical unit of storage (e.g. a disc pack). A physical unit of storage will be called a "partition". A directory is a list of object names.

Upon lookup in a directory, all names may be abbreviated as far

as they are non-ambiguous.

### 3.1.3. Definition of environments

Specification of the full object name for every OPEN-OBJECT would be very annoying. Therefore, a process (user) may specify his private environment to be a specific directory. All names given by this user will then be prefixed by the system with his current environment to form a full objectname.

Example:

> If the current environment is OPR.KETIL, a specification of the object name FILE-SYSTEM:BRF would give access to the object OPR.KETIL.FILE-SYSTEM:BRF.

If a object is not found in the current environment, a backtrace up from the current environment will occur to find the first matching object name directly connected to one of the owner directories.

Example:

> If the object QED is directly connected to the highest level directory, it will be available from all directories not having defined QED as a lower level. The highest level directory has the standard name ROOT. This part of a object name is only necessary if one wants to override a lower level name defined equal to a top level name.

On installations where the fastest possible access to global system objects is important, these system objects may be held in one directory and this directory defined to be the "system directory". An eventual system directory will always be the first one searched, regardless of the current environment. How to define a system directory is described later (Section *X*).

### 3.1.4. User names and Access Rights

When a user logs on he must identify himself by his user name. This user name gives him different access rights to different objects, but has nothing to do with the identification of objects. Thus the user name is not part of the object name.

Users are divided into user classes 0-15, where user class 15 usually, but not necessarily, is most privileged.

Objects may allow one or more of the basic access modes read, write, append, common, and delete to itself. The allowed access to an object may be different for different users.

An object will have an associated "allowed access" for each user class.

An object discriminates between three types of accessors. The owner, the friends and the public. The following rules apply:

- If the accessor is the OWNER, his access rights are defined

by the "own access" associated to the object.

- If the accessor is a FRIEND, his access rights are defined by his individual "friend access" associated to the object. Note that this access may be more limited than the public access.

- If the accessor is in the public category, his access rights are defined by the intersection of the rights his user class gives him and the "public access" associated to the object.

Because access rights may be set both for directories and end objects, access of both the contents of the objects and their names may be controlled.

Example:

> If a user has read access to the directory OLE.PER, an attempt to open OLE.PER for write will give the error return "NOT WRITE ACCESS TO OLE.PER". If the user does not have read access to the directory OLE, the error return would have been "NOT READ ACCESS TO DIRECTORY OLE".

When a new object (directory or end object) is created, the default access rights will be the same as the current access right on the next higher directory level.

When a user logs on, his name environment is set to a default value associated with the user. A user may change his current environment by the command:

SET-NAME-ENVIRONMENT

and change his default environment by

SET-DEFAULT-NAME-ENVIRONMENT

### 3.1.5. Partitions physical units of storage

The directory hierarchy is stored in a random access memory together with user files. Because of the storage size required, secondary storage (f.ex. disc) is usually used. A partition is the minimum unit of storage that can be independently removed from the system without destroying the consistency of objects and directories lying in it. The whole of a partition must be accessible to the system before any of the information lying on it is available. A partition will usually correspond to one physical unit of storage (e.g. disc pack), but may also comprise two or more physical storage units. Subdirectories cannot go accross partition boundaries. However, directories (content files) can contain names of subdirectories and/or objects located on different partitions, even on different systems.

As mentioned above, a partition will usually correspond to one physical unit of storage. But it is also possible to have a partition stored in a object! This form of recursion may be used for instance to build a floppy partition image on a disc object. Another example is to have a partition in a object that is fixed in memory for very fast access.

RETURN - the terminal will be given back to the parent process (corresponding to "RUBOUT" for the REMOTE function in SINTRAN III). The process that looses control of the terminal stream,will still be active, but it will be hung in a waiting state if it asks for input from the terminal. This command will have no effect if entered while control is at the top level process.

OUTPUT - discard terminal output. A second <esc><esc>OUTPUT will resume normal output.

STATUS - give status information on the program without stopping it.

USER - the user can use it as a trap; ignored by the system.

## 2.4. Subsystem control

### 2.4.1. PLACE-SUBSYSTEM

<file name>, <object index for domain>, <capability index>

The subsystem will be placed on the given domain. An indirect segment will be placed in the callers domain, giving subroutine access to the new subsystem. The new subsystem can have higher privileges than the caller. The file type should begin with :PROG, followed by computer type; for example :PROG-N500.

### 2.4.2. START-SUBSYSTEM

<capability index>, <routine number>

The subsystem can have several entry points. Normally the first one will be an initial entrypoint and the second a restart entry point.

### 2.4.3. EXIT

A trap condition will be raised, and control will be given to the nearest trap handler.

### 2.4.4. ERROR-EXIT

<reason>

A trap condition will be raised. The parameter <reason> can be read by the trap handler.

### 2.4.5. SUBSYSTEM-STATUS

<flag no.> <=flag value>

The relation between partitions, directories and objects is illustrated below:

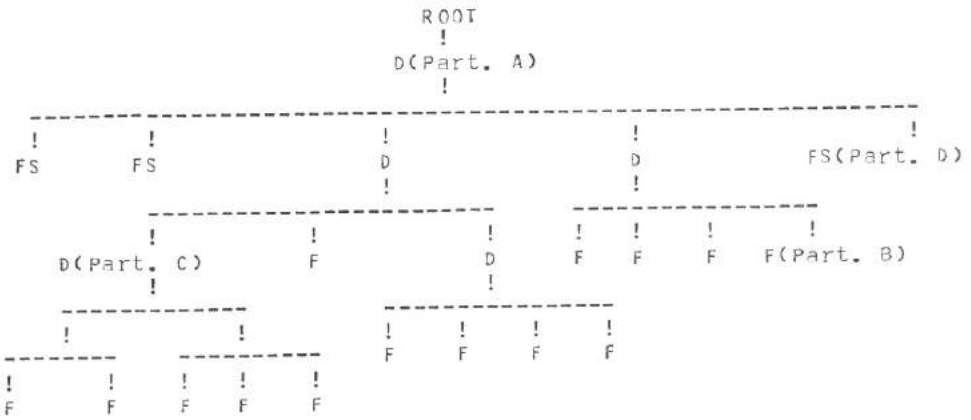



Figure 1

The partition A is a typical "system partition" (big disc). The FS's will be globally known objects (regardless of environment).

Partition B and D is a typical "private partitions" (floppy) containing one file.

Partition C may be owned by a group of users, or a more complex single user object structure.

Partitions B, C and D may be located on a remote system. The location of the partitions is specified when the object entries F(Part. B), D(Part. C), and FS(Part. D) are created.

### 3.1.6. How to get started - an example

Assume that your current environment is ROOT and that you want to create a directory on a just mounted empty disc pack, set that directory as your current environment and create a object in it. The following commands may then be used:

```
CREATE-PARTITION MYDISC,2048,,XY
CREATE-DIRECTORY MYDIR,,,MYDISC;
SET-NAME-ENVIRONMENT MYDIR
SET-DEFAULT-NAME-ENVIRONMENT MYDIR
CREATE-INDEXED-FILE MYFILE,10,10.500;
```

CREATE-PARTITION initializes the partition and makes it known to the system (enters it). Parameters:

1) MYDISC is the name given to the partition.

2) The allocation unit (page size) for the partition is set to

2048 bytes.

3) Bit object address. Default value for this parameter is the middle of the disc addressing space.

4) This is the low level name (not held by the object naming system) of the device where the partition is mounted.

CREATE-DIRECTORY initializes the top level directory MYDIR on partition MYDISC, and enters it in the current environment. Parameters:

1) MYDIR is the name given to the created directory.

2) Maximum number of pages the created directory and its objects and sub-directories may use. Default value is all free pages in the parent directory or partition (in this case the whole partition).

3) Reserved number of pages for the created directory and its objects and sub-directories. These pages are taken from the pages reserved for the parent directory and reserved for the created directory.

4) Initial number of pages in directory file. Default is 2.

5) MYDISC is the name of the partition where the directory is to be placed. Default value is the same partition as the parent directory. When the parent directory is ROOT the partition name must always be specified.

SET-NAME-ENVIRONMENT sets MYDIR as current environment. Hereafter, all objects will first be searched for in this directory (but after an optional "system directory", as mentioned earlier).

SET-DEFAULT-NAME-ENVIRONMENT sets MYDIR as default name environment for the currently entered user. The user default name environment is the environment set when the user logs on.

CREATE-INDEXED-FILE creates the indexed object MYFIL in the current environment (MYDIR).

1) MYFIL is the name of the created object. The full name of the object will will be ROOT.MYDIR.MYFIL, but as long as the current environment is ROOT.MYDIR, only MYFIL will be sufficient to identify the object. MYDIR.MYFIL will be sufficient to identify the object from all environments, except if MYDIR.MYFIL has another meaning in the current environment. In that case the full name ROOT.MYDIR.MYFIL WILL HAVE TO BE USED.

2) The initial number of pages in the object will be 10. These pages will ,if possible, be allocated contiguously on the partition.

3) The minimum last page number of the file will be 10. This means that even if the file is cleared (for instance copying an empty file to it) it will have 10 data pages allocated to it.

4) The maximum page number is set to 500.

### 3.1.7. Object types

Aassociated to every object is an "object type" telling what kind of object it is. The object types supported by the system are:

File (section 3.3)
Directory (section 3.1)
Device (section 3.6)
Volume (section 3.3)
Synonym (section 3.1.9)
Process dependent synonym (section 3.1.10)
Port (section 3.4)
Channel (section 3.4)
System extensions (section 3.7)
Process (section )
Trapobject (section 3.7)
Resource lock (section 3.7)
Port reference (section 3.4)
Area (section 3.7)

In addition, users may define their own object types. User defined object types may be opened by OPEN-OBJECT, and their object entry may be read and partly written by the user.

### 3.1.8. Object security

Aassociated with every object is a security attribute which may have the values Free, Controlled, Secret.

If an object is defined to a controlled object or a secret object, all attempts (successful or unsuccessful) to open the object will be reported to the SINTRAN-IV logging system.

If an object is defined to be a secret object, the system will prohibit all copying of the object to an object with lower minimum user class allowed to read, and to non secret objects. This gives protection against leakage of secret information to objects or devices accessible by non privileged users. It is not enough for a data thief to succeed in logging on as a privileged user, he must also have access to a privileged output device.

### 3.1.9. Synonyms

A synonym is a kind of "alias" name. When an object of the type synonym is opened, a recursive call to OPEN-OBJECT is generated (internally in the object name system) with the value of the found synonym as argument. The synonym value must be an unambiguous object name. A synonym name as the value of another synonym is allowed.

Note that the lookup of a synonym value may depend on the current name environment, implying that the effect of opening a synonym may depend on the current name environment. This

environment dependency may be avoided by defining the synonym value as a full name starting from ROOT.

Example:

The command CREATE-SYNONYM ANNE, OLE.DAUGHTER will define the synonym ANNE in the current environment equivalent to the string OLE.DAUGHTER. That means that as far as object names in this environment is concerned the strings ANNE and OLE.DAUGHTER may be used interchangeably. For example will the strings ANNE.FILE and OLE.DAUGHTER.FILE denote the same object.

### 3.1.10. Process dependent synonyms

A process dependent synonym works in the same way as a synonym (section 3.1.10) except that its value depends on, and must be defined by, the actual process running. This is useful for instance if it is convenient to have a common global name for a resource that many processes have their own copy of.

### 3.1.11. Name specifications in commands and monitor calls

All names in the object naming system (object names, directory names, partition names etc.) may be abbreviated according to the standard SINTRAN 4 rules.

When a user specifies a name string he may have two different intentions. Either he wants to identify a single object (file, directory etc.) or he wants to specify a set of objects to be acted upon. In the SINTRAN 4 object naming system these two cases are separated explicitly by the syntax of the name specification. The following rules apply:

A) If a name specification contains no * and + character it is interpreted as an intention to identify a single object. If the name specification is ambiguous, an error message is given.

B) If a name specification contains one or more *, it is interpreted as an intention to identify a set of objects. The set identified is the set of all objects giving full or partial match with the name specification, where an * at the end of the name is defined to match any character string, and an * elsewhere in the string is defined to match any single character except "-" (dash) and "." (period).

C) A set of objects may be specified by separating strings according to rule A and/or B by a + character.

The ordering of a set of objects will be from left to right according to rule C and alphabetic within each (sub)set specified according to rule B.

Example:

```
FIL**OLE+PER.*
```

This string specifies the following set of objects:

All objects starting with FIL alphabetically ordered.
The single object OLE.
All objects in the directory PER and its subdirectories in alphabetic order.

If the user wants to explicitly state that a string would match a directory name, it may be ended by :DIRECTORY. (Type DIRECTORY is illegal on other object types).

## 3.1.12. Monitor calls to the object naming system

### 3.1.12.1. CREATE-PARTITION

(<partition name>, <bit file address>, <device name>, <=status>)

Create an empty partition on a device (disc pack).

### 3.1.12.2. ENTER-PARTITION

(<partition name>, <device name>, <=status>)

If <device name> has more than one unit, all units will searched for the specified partition. <device name> may also be an object in another partition. Default for <device name>: Search all devices in "default partition device table".

### 3.1.12.3. CREATE-DIRECTORY

(<dir.name>, <max.number of pages>, <reserved number of pages>, <initial number of dir.pages>, <partition name>, <device name>, <=status>)

If <dir.name> is a single name (no dot) the directory will be created in the current environment.

If <dir.name> is a multi level name , all levels to the left of the rightmost dot must be existing directories. The rightmost part of the name will then be created as a directory in the environment defined by the names to the left of the rightmost dot.

<max.number of pages> is the maximum number of pages the files and subdirectories in the created directory may use. <reserved number of pages> are exclusively reserved for the files and subdirectories in the created directory. These pages are taken from the pages of the parent directory, or from the partition entry if top level directory.

<partition name> must be specified if the directory to be created is to be placed on another partition than it's parent directory.

<device name> may be omitted if the partition is already

entered.

Friend table and access rights will be copied from the parent directory to the created directory.

<initial number of dir.pages> specifies the initial number of pages to allocate for the directory itself. The directory itself will be indexed and expandable.

### 3.1.12.4. ENTER-DIRECTORY

(<dir.name>, <partition name>, <device name>, <=status>)

This monitor call is used to enter an existing directory or too level object into the directory hierarchy.

The rightmost part of <dir.name> must be a top level directory or object in the partition specified by <partition name>. If this partition is already entered, the <device name> parameter is dummy, if not all devices in the default partition device table will be searched. If <device name> has more than one unit, all will be searched for the specified <partition name>.

If <dir.name> is a single name (no dot) the directory will be entered in the current environment.

### 3.1.12.5. RELEASE-DIRECTORY

(<dir.name>, <=status>)

This is the inverse of ENTER-DIRECTORY. The specified directory is disconnected from the directory hierarchy.

### 3.1.12.6. RELEASE-PARTITION

(<partition name>, <=status>)

If a partition is released without releasing a directory on it, the operator will be asked to mount the partition if an attempt to access an object on it is done.

The entering and realising of partitions should follow the physical mounting and dismounting of the partition.

### 3.1.12.7. SET-OWNER-ACCESS

(<object name>, <owner name>, <access>, <=status>)

Change the owner name and/or owner access of the specified object. Write access to parent directory is required.

<object name> This parameter may be specified in a number of ways:
A single object
A single directory
If the parameter string ends with an *, the *

will match all strings and all matching objects will be changed
Examples:
"OLE.*": This will change all end objects under the directory OLE.
"OLE:DIRECTORY": This will change directory OLE

<owner name> New owner name. Default = no change.

<access> R, W, A, C, D or any combination, representing read, write, append, common and delete access respectively. From monitor call <access> is given as an integer value where R = bit 0, W = bit 1, A = bit 2, C = bit 3, D = bit 4. Exs: RW = 3, W = 2, WC = 12.
100 octal (64 decimal) will give no change in access.

### 3.1.12.8. CREATE-FRIEND

(<object name>, <friend name>, <access>, <=status>)

Create a friend to an object and specify his access. Write access to parent directory is required.

<object name> same as for SET-OWNER ACCESS (section 3.1.12.7)

### 3.1.12.9. DELETE-FRIEND

(<object name>, <friend name>, <=status>)

Delete a friend from an object. Write access to parent directory is required. After he is deleted, the friend will belong to the public category for this object.

### 3.1.12.10. SET-PUBLIC-ACCESS

(<object name>, <access>, <=status>)

Write access to parent directory is required.

### 3.1.12.11. SET-USER-CLASS-ACCESS

(<object name>, <access array>, <=status>)

The <access array> is a 16 element array with one element for each user class. The elements have the format described for access in section 3.1.12.7.

### 3.1.12.12. SET-DOMAIN-PRIVILEGE-ACCESS

(<object name>, <required privileges for read>, <required privileges for write>, <required privileges for append>, <required privileges for common>, <required privileges for delete>)

The "required privileges" parameters are a 32 bits word where a one bit specifies that the corresponding privilege must be present to access the object.

### 3.1.12.13. GET-OBJECT-ACCESS

(<object name>, <user>, <=access>, <=status>)

Returns the access of the specified user to the specified object.

### 3.1.12.14. CREATE-OBJECT

(<object name>, <object type dependent part of object entry>, <partitionname>, <devicename>, <=status>)

This is the general monitor call to create all kinds of objects. The call associates <object name> to the object described by the contents of the second parameter. If <object name> is a single name (no dot), the object will be placed in the current environment directory. If <object name> is a multilevel name, all levels to the left of the rightmost dot must be existing directories. The rightmost part of the name will then be created in the directory defined by the names to the left of the rightmost dot.

<partition name> must be specified if the object entry is to be placed in the top level directory on another partition than the parent directory.

<device name> may be omitted if <partition name> is already entered or mounted on a "default partition device". If <device name> is unambiguously specified, <partitionname> may be omitted.

This monitor call may be used to create user defined objects. The <object type dependent part of object entry> parameter is an 80 bytes array containing information to be stored in the object entry. The format of this array must obey certain rules. See system documentation for further details.

### 3.1.12.15. CREATE-NAME

(<object name>,<object index>,<=status>)
This monitor call may be used to name an opened object. Only previously unnamed objects may be named by this call (f.ex. Port or Nameless file). If <object name> is a single name (no dot), the object will be placed in the current environment directory. If <object name> is a multilevel name, all levels to the left of the rightmost dot must be existing directories. The rightmost part of the name will then be created in the directory defined by the names to the left of the rightmost dot.

3.1.12.16. CREATE-SYNONYM

(<syn.name>, <syn.value>, <=status>)

<syn.name> is defined to be equivalent to <syn.value> in the current or specified environment (if <syn.name> contains one or more dots). <syn.name> must be an unique not previously defined name in this environment. <syn.value> must uniquely define another object.

Note that if the object identified with <syn.value> is deleted the corresponding synonym is not automatically deleted. An attempt to access a deleted object through a synonym will give an error return.

3.1.12.17. CREATE-PROCESSDEPENDENT-SYNONYM

(<syn.name>, <=status>)

<syn.name> is defined as an object of type Processdependent synonym. The value of the synonym must be set by each individual process using it by the SET-PROCESSLOCAL-SYNONYM-VALUE call (section 3.1.12.17)

3.1.12.18. SET-PROCESSLOCAL-SYNONYM-VALUE

(syn.name>, <syn.value>, <=status>)

<syn.name> must identify an object of type Processdependent synonym. The <syn.value> will be set as the value of the synonym for the process issuing the call.

3.1.12.19. SET-OBJECT-SECURITY

(<object name>, <security>, <=status>)

Set the security attribute for an object. <security> may have the values Free, Controlled, and Secret. The security attribute is described in section 3.1.8

3.1.12.20. GET-OBJECT-SECURITY

(<object name>, <=security>, <=status>)

Returns the security attribute for an object.

3.1.12.21. SET-OBJECT-PROGRAM-PRIVILEGES

(<object name>, <privileges>, <=status>)

For objects (usually files) containing an executable program, this call may be used to define the domain privileges the program stored in the file should have when it is executed. The <privileges> parameter must specify privileges less or equal to the privileges of the domain executing the SET-

OBJECT-PROGRAM-PRIVILEGES call.

### 3.1.12.22. GET-OBJECT-PROGRAM-PRIVILEGES

(<object name>, <=privileges>, <=status>)

Returns the domain privileges of the program stored in <object name> (set by SET-OBJECT-PROGRAM-PRIVILEGES).

### 3.1.12.23. RENAME-OBJECT

(<object name>, <new object name>, <=status>)

Change the name of an object.

### 3.1.12.24. SET-NAME-ENVIRONMENT

(<directory name>, <=status>)

The specified directory is set as current name environment for the process issuing call.

### 3.1.12.25. SET-DEFAULT-NAME-ENVIRONMENT

(<directory name>, <=status>)

The specified directory is set as default name environment for the user issuing the call. The default name environment is used when a user is logging on to set his initial name environment.

### 3.1.12.26. SET-SYSTEM-DIRECTORY

(<directory name>, <=status>)

The specified directory will be set as "system directory". This directory will be searched first in OPEN-OBJECT. This command may be used on installation where fast access to global ("system") files is important.

If <directory name> is an empty string, no system directory is set.

### 3.1.12.27. DELETE-NAMED-OBJECT

(<object name>, <=status>)

The object(s) specified is(are) deleted.

### 3.1.12.28. OPEN-OBJECT

(<object name>, <default type>, <access mode>, <access method>, <=object index>, <=status>)

<object name> may specify a single object or a set of objects. If the <object name> parameter contains an * (asterisk), all objects matching this name are opened and concatenated. The ordering of the opened objects will be alphabetic on directory and objects names, higher level names being more significant than lower level ones. If the <object name> string ends with an *, the * is interpreted as matching any character string. All objects giving match according to this rule will be opened and concatenated in alphabetic order. Finally, the <object name> string may consist of a number of object names separated by the character +. In this case all the objects names separated by the string will be opened and concatenated in the order they appear in the <object name> string. If <object name> has the type string 'DIRECTORY', the directory will be opened. Opening of directories for write is only allowed for highly privileged programs. The default block size will be the object entry size. For access of directories, see section 3.5.

<default type> is appended to the object name(s) if the <object name> does not contain a type specification.

<access mode> is a combination of Read, Write, Append, Common, Delete, and Parent represented by bit 0,1,2,3,4, and 5 respectively. If bit 5 (Parent) is set, the purpose of the opening is to read or alter the information in the object entry. In that case the wanted access is equired to the parent directory, not to the object itself.

<access method> is direct (0), unstructured (1) or record access (2). An object may be opened at the same time with different access method and different object numbers from one process. Default = unstructured. This parameter is not significant for all object types.

<=object index> is reference number returned from the object naming system. If the object is already opened from this process, this will be reported through <=status> and the (old) object number returned in <=object index>.

In addition to error codes, <=status> may return the values Single file, File set, and File already opened by yyou.

After an object has been opened, it may be accessed by a number of monitor calls taking the object index as argument. Some access monitor calls will only be relevant to one object type. These monitor calls are described in the sections on their respective object types.

There will be an implementation dependent maximum for the range of object indexes and the number of concurrently opened objects.

The monitor calls to the object naming system taking an object index as argument are described in section 3.1.13. The general input/output monitor calls to access any open i/o object are described in section 3.2.

## 3.1.12.29. Object naming monitor calls acting on opened objects

### 3.1.12.29.1. CLOSE

(<object index>, <special action>, <=status>)

This is the inverse of OPEN-OBJECT. The object index is released and the object is "cleaned up".

<object index> = -1 has the special meaning: close all opened objects not permanently opened (see section 3.1.13.2).

<object index> = -2 has the special meaning: close all opened objects.

The normal value of <special action> is 0. For some object types it may have another value indicating some special action to be taken upon close. See the section on the object type in question for further details.

### 3.1.12.29.2. SET-PERMANENT-OPEN

(<object index>, <=status>)

The object identified by <object index> is set permanently opened and thus, it is not closed by CLOSE(-1).

### 3.1.12.29.3. CHECK-POINT

(<object index>, <generation>, <maxversion>, <=status>)

The action taken upon CHECK-POINT will be object type dependent, but the purpose of the call is to bring the object into some welldefined state, for instance by emptying buffers etc.

The parameters <generation> and <maxversion> are only relevant to files (section 3.3)

### 3.1.12.29.4. GET-OBJECT-ENTRY

(<object index>, <memory buffer>, <number of bytes>, <=bytes transferred>, <=status>)

This monitor call returns the object entry of an opened object.

### 3.1.12.29.5. UPDATE-OBJECT-ENTRY

(<object index>, <object type dependent part of object entry>, <=status>)

This call may be used to set the object type dependent part of user defined object entries. The call is illegal on system supported object types.

### 3.1.12.29.6. GET-CURRENT-OBJECT

(<object index>, <=object name>, <=restname>, <=object type>, <=generation>, <=status>)

Return information on the currently opened object referenced by <object index>.

<=object name> return the full (unabbreviated) name of the object.

<=restname> returns the reststring to the right of the object name in the OPEN-OBJECT call.
Example: Assume OLE is an object name in current environment.

```
OPEN-OBJECT(OLE.USERINFO,.......)
```

GET-CURRENT-OBJECT will now return '.USERINFO' in <=restname>

<=object type> returns the object type

<=generation> is only relevant for files (section 3.3)

### 3.1.12.29.7. GET-NEXT-OBJECT

(<object index>, <=object name>, <=restname>, <=object type>, <=generation, <=status>

Set the next object in the set of objects opened on <object index> s current and return information about this object. Parameters as described in section 3.1.13.6.

### 3.1.12.29.8. GET-PREVIOUS-OBJECT

(<object index>, <=object name>, <=restname>, <=object type>, <=generation>, <=status>)

Set the previous object in the set of objects opened on <object index> as current and return information about this object. Parameters as described in section 3.1.13.6.

### 3.1.12.29.9. RESERVE

(<objectindex>, <access>, <=status>)

This monitor call reserves the object identified by <object index> and <access> for exclusive use by the issuing process. If the object is currently reserved by another process the first process will enter waiting state until the object is released. maximum waiting time may be defined by

SET-TIMEOUT. <access> is 1 for input 2 for output 3 for both.

### 3.1.12.29.10. RELEASE

(<object index>, <access>, <=status>

Release the previously reserved object identified by <object index> and <access>. <access> is 1 for input 2 for output and 3 for both.

### 3.1.12.29.11. MULTI-RESERVE

(<object index array>, <number of units>, <=status>)

Reserve all the objects specified by <object index array>. If one of more of them is already reserved by another process nothing will be reserved and the calling process will enter waiting state until all specified objects are free.

The two most significant bits in the elements of <object index array> contains the access (1 for input, 2 for output, 3 for both) and the rest contains the object index.

### 3.1.12.29.12. MULTI-RELEASE

(<object index array>, <number of units>, <=status>)

Release all the objects specified by <object index array>.

### 3.1.12.29.13. WHERE-IS-OBJECT

(<object index>, <access>, <=process id.>, <=status>)

Returns the process identification of the currently reserving process (if any) of <object index>.

### 3.1.12.29.14. TRANSFER-OBJECT-INDEX

(<from object index>, <from domain>, <to object index>, <to domain>, <=to object index>, <=status>)

Transfer the open object reference identified by <from open index> for the domain <from domain> to the domain <to domain> and give it the object index specified by <to object index> there. If <to object index> is zero, a free value will be found by the system and returned through <=to object index>. After this call the object will not be accessible from <from domain>.

#### 3.1.12.29.15. COPY-OBJECT-INDEX

(<from object index>, <from domain>, <to object index>, <to domain>, <=to object index>, <=status>)

Same as TRANSFER-OBJECT-INDEX (section 3.1.15.15) except that the reference identified by <from object index> is not removed by the call. After the call, the object will be accessible from both domains.

## 3.2. General I/O monitor calls

### 3.2.1. Introduction

This section describes the input/output monitor calls acting on open objects. The calls may be used on all I/O object types, and represents the same function on all object types although the internal action taken by the system may be different for different object types.

The purpose of these calls is to offer a data medium, independent I/O interface to the user. This interface enables the user to write programs doing I/O against different object types (files, ports, devices etc). Without changes and often without even knowing what kind of I/O medium it is working against. In this way programs will be less fragile to changing environments. The drawback of such an approach is that the general I/O calls can not be taylored to the special functions of each individual object type. Therefore there also exists special I/O calls for some object types giving the user control over all special features of the object type. The general calls will be translated internally in the system to a special call with default values on the special parameters not included in the general call. The special calls are described in the sections on the respective object types.

### 3.2.2. READ-BLOCK

(<object index>, <block no.>, <memory address>, <number of bytes to read>, <=bytes actually read>, <=status>)

This is the general monitor call to read data into memory.

<object index> — Object index returned from OPEN-OBJECT.

<block no.> — This parameter contains addressing information, but all values of this parameter will not be relevant to all object types.

<block no.> >= 0 : Absolute addressing of input medium. Only relevant to absolute addressable object types

<block no.> = -1 : This value may be used on all input object types specifying sequential input.

<block no.> = -4 -5 : These values are only relevant for input media (object types) dividing the data into addressable records (e.g. files, section 3.3). Object types with no record concept are treated as one single record.

<block no.> = -2 : Read from start of next record. This value is only relevant for input media (object types) dividing the data into records. Object types with no record concept are treated as one single record.

<block no.> = -3 : This value specifies sequential input, but before the data is read an implicite GET-NEXT-OBJECT (section 3.1.) is performed giving a change to the next object in the opened object set.

<block no.> = -6 : This value specifies sequential input, but before the data is read an implicite GET-PREVIOUS-OBJECT (section 3.1.) is performed giving a change to the previous object in the opened object set.

Other values of <block no.> are illegal

<memory address> : Where to place the data read

<number of bytes to read> : Amount of data wanted

<=bytes actually read> : Returns the amount of data transferred to user memory. This amount will be less than or equal to <number of bytes to read>.

### 3.2.3. WRITE-BLOCK

(<object index>, <block no.>, <memory address>, <number of bytes>, <bytes actually written>, <=status>)

This is the general monitor call to transfer data from memory to an output medium or external storage.

<object index> : Object index returned from OPEN-OBJECT.

<block no.> : This parameter contains addressing information, but all values of this parameter will not be relevant to all object types.

<block no.> >= 0 : Absolute addressing of output medium. Only relevant to absolute addressable object types.

<block no.> = -1 : Sequential output. May be used on all output object types.

<block no.> = -2 : The data is written from current byte pointer and current record terminated. Only relevant to output media (object types) dividing the data into records.

<block no.> = -3 : The data is written from current byte pointer. Then a change to the next object in the opened object set is performed.

Other values of <block no.> are illegal.

<memory address> : Location in memory of the data to write.

<number of bytes> : Amount of data to write.

<=bytes actually written> : Return amount of data actually written. This amount will be less than or equal to <number of bytes>

### 3.2.4. SET-TIMEOUT

(<object index>, <access>, <time>, <=status>)

This call may be used to set the maximum wait time for an I/O access. The timeout set by this call will be in effect for all calls to an object index that may involve waiting, except if the call itself contains a timeout parameter.

<object index> : Object index returned from OPEN-OBJECT

<access> : 1 for input, 2 for output

<time> : This parameter is interpreted as follows:

<time> >= 0 : Specifies timeout time in milliseconds. If a call involving a data transfer returns on timeout, the transfer will still be going on an it is the user responsibility not to go interfere by accessing the data buffer (if he wants the transfer to complete).

<time> = -1 : Infinite timeout. No call will return until the operation is finished. A WRITE-BLOCK operation is defined to be finished when the data is moved out of the users data area.

<time> = -2 : A call will give "immediate" return, but an attempt (by the user) to access the segment containing the data buffer will hang the user process until the operation is finished.

### 3.2.5. WAIT-FOR-IO

(<object index>, <access>, <timeout>, <=object index>, <=status>)

This call may be used to wait for another call that returned on timeout to finish. A transfer is said to be unterminated if the call initiating it returned on timeout and a TERMINATE call (section 3.3.6 & 3.3.7.) has not been issued to terminate it. If there is no unterminated transfers on the object index(es) specified, WAIT-FOR-IO will give an error return.

<object index> : If this parameter is nonzero, a wait on this transfer will be performed. If this parameter is zero, a wait on all unterminated transfers will be performed, and return given when the first one finishes.

<access> : 1 for input, 2 for output.

<timeout> : Maximum wait time in WAIT-FOR-IO in millisecond.

<=object index> : Returns identification of the transfer that has finished.

### 3.2.6. TERMINATE-READ-BLOCK

(<object index>, <=bytes actually read>, <=status>)

This call may be used to get the return parameters from an earlier READ-BLOCK call that returned on timeout. The call also terminates the unterminated READ-BLOCK transfer. If TERMINATE-READ-BLOCK is called before the READ-BLOCK operation is finished, this will be reported through <=status> and the transfer aborted.

### 3.2.7. TERMINATE-WRITE-BLOCK

(<object index>, <=bytes actually written>, <=status>)

This call may be used to get the return parameters from an earlier WRITE-BLOCK call that returned on timeout. The call also terminates the unterminated WRITE-BLOCK transfer. If TERMINATE-

PREPARE-READ-BLOCK is executed. The user program will then continue in parallel with the disc transfer, and when the READ-BLOCK is executed by the user program the data will typically be available in a system buffer (disc cache) and only a copying from the system buffer to the user area will take place.

### 3.2.9. BUFSIZE

(<object index>, <access>, <=records available>, <=bytes available>, <=status>)

This monitor call returns the number of records and number of bytes available on <object index> for read (<access>=1) or write (<access>=2) without the calling program entring waiting state. For object types without the record (or message) concept, <=record available> will always be 1.

### 3.2.10. SET-PARALLEL-OUTPUT

(<common object index>, <object index 1>, <object index 2>, <=status>)

After this monitor call is done, everything written to <common object index> goes to both the output media identified by <object index 1> and <object index 2>. If more than two parallel data streams are wanted, repetitive calls to SET-PARALLEL-OUTPUT may be used. The output calls and parameters allowed on a set of parallel output stream is equal to the ones allowed for the most restricted of them. If for example a disc file and a lineprinter is set parallel, only sequential write will be allowed.

## 3.3. File system

### 3.3.1. Introduction

The service offered by the file system to the user is to enable him to store away data for later retrieval by himself or another user. A file is a collection of data stored in a read/write memory (usually mass storage) and accessible by the input/output monitor calls. The minimum addressable and accessible unit of data is one byte (8 consecutive bits).

The file system is designed to hold the physical properties of the storeage medium from the user. However, to avoid prohibitive inefficiencies, the distinction between random access and sequential storage media will be visible to the user. Therefore, there are two main types of files: Partition files and Volume files. Partition files are stored in a random access medium (f.ex. disc), while Volume files may be stored in a sequential access medium (f.ex. magnetic tape). These differencies in the storage medium affects the physical storage format and, more important to the user, the allowed operations will be more limited on Volume files than on Partition files.

### 3.3.2. Partition files

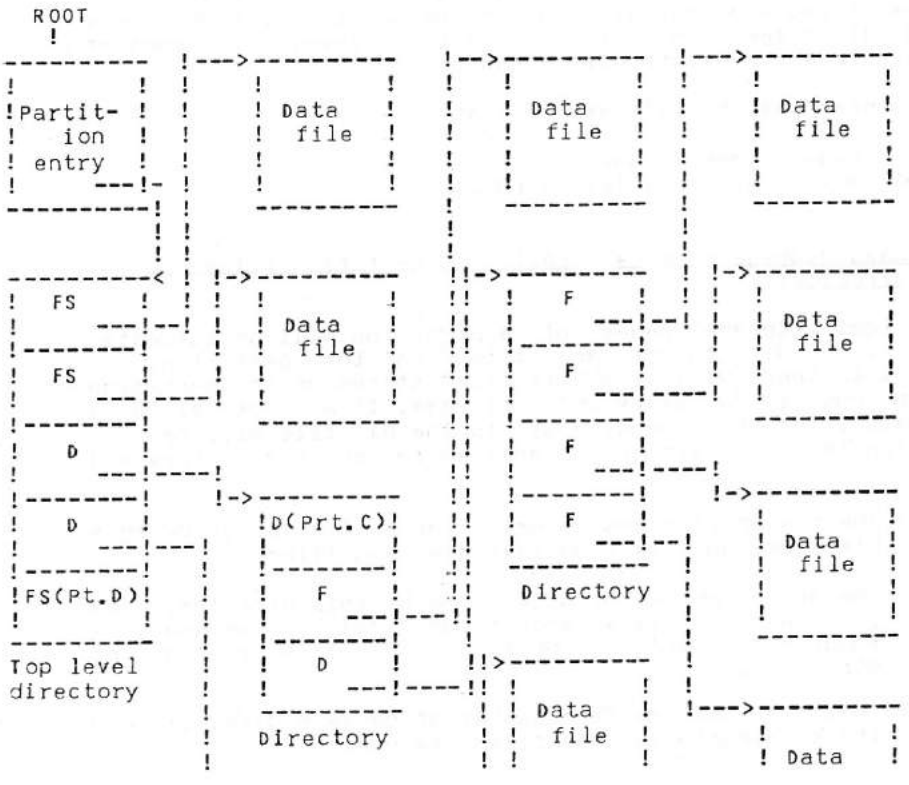
Partition files have a closer relationship to the object naming system than other objects, because the directories themselves are stored in Partition files. This implies that directories and partition files the storage space (pages) from the same pool of free space on a partition. Apart from this space sharing with directories there is no principal difference, as seen from the user, between a partition file and any other object type.

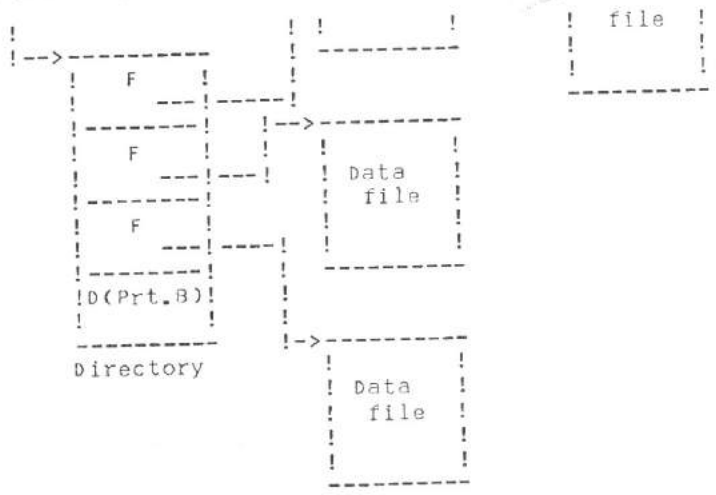
At a lower implementation level in the file system (level 1), there is an interface used by both the object naming system and a highest level (level 3) of the user file system. In principle, level 1 sees no difference for storing directories.

IO allow dynamic expansion of data files and directories, the data pages of a partition file may be indexed by 1, 2, or 3 index levels as illustrated in section 3.3.3. The page indexing of files is completely invisible to the user except for the fact that indexed files are dynamically expandable.

### 3.3.3. Partition format

The partition layout below corresponds to partition A in the figure in section 3.1.5.






Note that the top level directory has no name, and may be seen as a variable length extension to the partition entry. The top level directory is present even if the partition consists of only one top level file.

The "partition entry" is one allocation unit (page) at a predefined fixed address. All other boxes in the figure above are general files with 0, 1, 2, or 3 index levels. The root of the index hierarchy is the file pointer.

A file pointer has the following format

Bit 0-29 Page number on device
Bit 30-31 Number of index levels (0-3)

### 5.3.4. Mass storage space allocation to partition files and directories

The physical storage pages of a partition will be allocated dynamically to directories and files on the partition. To control this there will be a "bit file" stored on the partition containing one bit for every physical page. If a physical page is allocated, the corresponding bit in the bit file will be set. In addition three numbers are maintained for every directory and file:

RESPAGES: The number of pages reserved for exclusive use by this directory and its subdirectories and files.

USPAGES: The actual number of pages used by this directory and its subdirectories and files. Pages reserved by a subdirectory will be counted as used in its parent directory.

MAXPAGES: This is the maximum number of pages a directory and its subdirectories and files may use.

The system checks that USPAGES<=MAXPAGES is always true. Note that MAXPAGES (and thus USPAGES) may be greater than RESPAGES. However, the pages exceeding RESPAGES will not be exclusively reserved for this directory/file, so the directory/file will have to compete for this pages with other directories/files in the same parent directory and therefore these pages can not be guaranteed to be available (overbooking).

Initially, when an empty partition is created, all pages are given to the top level directory. That is, RESPAGES=MAXPAGES= number of physical pages on partition, and USPAGES= number of pages initially used by the system (for bit file etc.). Pages may then be reserved for lower level files/directories by RESERVE-PAGES (section 3.3.11.14) and transferred between lower level files/directories in the same parent directory by TRANSFER-PAGES (section 3.3.11.15).

### 8.3.2.1. INIT-ACCOUNTING

<desired> <max.>

<desired> is the desired decimal number of accounts. After the file has reached this limit the warning APPROACHING END OF ACCOUNTING FILE is written out on every log off. Default value is 500. <max.> is the maximum decimal number of accounts permitted. After the file has reached this limit the warning END OF ACCOUNTING FILE ENCOUNTERED is written out on every log off. No accounting is done after this number is reached. Default value is 600.

### 8.3.2.2. START-ACCOUNTING

This command starts the accounting, but does not initiate the accounting file.

### 8.3.2.3. STOP-ACCOUNTING

This command stops the accounting system. The accounting files is not affected.

### .3.2.4. ACCOUNTS

This is a subsystem to print the results of the accounting.

difference is that an indexed file dynamically expandable because one or more index levels will be automatically generated if necessary.

Directories will always be indexed.

An indexed file may also be logically discontiguous. That means that pages never written into may not be physically allocated. An attempt to read data from nonallocated areas ("holes") of a file will give an error return.

True indexed files are not allowed on volumes. However, to allow backup of partition files with holes on volume files, a degenerated version of indexed files are allowed even on volumes. This is done by storing a logical byte number together with each physical page on the volume. An indexed volume file may only be written with ascending logical addresses, and read in physical sequence.

### 3.3.7. File allocation attributes

These attributes, which are mutually exclusive, decide the lifetime of a file name and its data.

#### 3.3.7.1. Permanent files

This is the "normal" file. Lifetime of both file name and data pages are controlled by user commands.

#### 3.3.7.2. Scratch files

During a terminal session (or batch job) a user may create scratch files. These files are deleted when the user is logging off.

Scratch files are illegal on volumes.

#### 3.3.7.3. Temporary files

A temporary file is a "read once" file. That is, all pages in the file are deleted when it is closed after an open for read (not read/write).

Temporary files must be indexed files and are illegal on volumes.

#### 3.3.7.4. Nameless files

If an empty string is specified between double quotes ("") in an OPEN-OBJECT command, a nameless file will be created in current environment. This file disappear as soon as it is closed.

Nameless files are illegal on volumes.

### 8.1.4. SAVE-SYSTEM

<file name>

The system will be stopped as by STOP-SYSTEM, and all the physical memory will be copied to a contiguous file. The system can be restarted by an offline bootstrap program.

### 8.1.5. INITIATE-TERMINALS

When a system is started, only one terminal in active. This command will prepare the other connected terminals for time-sharing.

### 8.1.6. INITIATE-NETWORK

Connections to other systems are established.

### 8.1.7. SET-INITIAL-COMMAND

<macro call>

This is a macro call to be executed when the system is restarted.

### 8.1.8. SET-AVAILABLE

It is possible to log in on the terminals.

### 8.1.9. SET-UNAVAILABLE

<text>

It is not possible to log in on terminals other than the console terminal. Instead, the message given by <text> will appear on the terminal. When all users have logged out, a message is returned.

### 8.1.10. CONTROL-PROCESSOR

<processor name>

This subsystem can control the processor completely, starting and stopping it, and running test programs.

## 8.2. System modifications

The system can be modified by the subsystem
SINTRAN-SERVICE
The modifications include

I/O-device-handling
Processor and memory usage tuning

#### 3.3.8.2.3. Keyed organization

Files with keyed organization have the same properties as sequential files, and in addition, have a prime key associated with each record. The order of the records in the file will be the order of the prime key values. The prime key must be unique, and can therefore be used to access any one record in the file. The user may change between sequential and keyed access at will.

As many secondary keys as necessary may be specified. Such keys need not be unique. Access via these keys will supply the records in the order of the chosen secondary key's value. If there are several instances of a record to one key value, the records will be supplied in the order in which they were written - i.e. the oldest record will be supplied first.

Prime keys may be variable in length, in which case they must occur first in the record, but secondary keys must have a fixed length, and a fixed position in the record, relative to the end of the prime key. This restriction is necessary at present as most programming languages do not provide suitable constructions to deal with variable length structures.

Keyed organization is illegal on volume files.

#### 3.3.8.2.4. Relative organization

A nascent relatively organized file consists of a number of empty records, each the maximum number of bytes long. These records are accessed via an integer value - the value 1 will refer to the first record, the value 2 the second, and so on.

Relative organization is illegal on volume files.

### 3.3.9. Special file attributes

These attributes, which are mutually exclusive, define some special properties of files used for special purposes.

#### 3.3.9.1. Normal files

This is the normal file having no special property. This is the only special attribute allowed on volume files.

#### 3.3.9.2. Ringbuffer files

A ringnumber file works as a ringbuffer. When the file reaches its maximum size, the writing program may enter waiting state (depending on the timeout set). When data is read from the file, the space it occupied is released. When the file is empty the reading program may enter waiting state (depending on the timeout set).

A new standard format is created. The message to appear on the console terminal can be used for describing the paper and hardware setting on the printer, if these are easily selectable and an operator is usually present.

### 7.7.7. DELETE-FORMAT

<format no.>,<=status>

The format is deleted.

### 7.7.8. SET-FORMAT

<format no.>, <device name>, <=status>

From now on, the device will be reserved for files of a particular format. Not all fields in format <format no.> need be specified; format compliance is only enforced for the fields specified. However, if format parameters regarding printer hardware settings are unspecified, the user should also leave these fields unspecified, since exact match is always required here (cf. sect. 5).

## 7.8. Print access

The print access checks performed are installation dependent. It is always true, though, that for privileged commands, caller access to any file(s) concerned is not checked. Public commands are normally restricted to the appending user and operator privileged users. The append command does always check the calling user's access to the file. For the PRINT-QUEUE and PRINT-SYSTEM-STATUS commands, file and user names are replaced by empty spaces whenever the caller does not possess access rights or operator privilege.

If several instances of the same file are present in the spooling queue, the commands specifying an action to be performed on a <file name> will affect all matching files. (For MOVE-PRINT, the first occurrence of <before file> is assumed.) To enable individual file access, and also to make life easier for the operator, each file is given a unique tag value consisting of four alphanumeric bytes. Whenever a file name is to be specified, a tag value may be substituted, preceded by the '#' tag indicator (ex.: #12AB). The tag is returned from the APPEND-PRINT command, and may also be looked up by calling PRINT-STATUS or PRINT-QUEUE.

If only an incomplete (not bottom-level) device name is specified for a command signifying an operation to be performed on a device (BACKSPACE-, STOP-, SUSPEND-, etc.), all matching devices will be affected, provided the operation is meaningful and the caller has access to any print concerned.

When a "delayed update" file is closed all unmodified pages are transferred to the new file and the old (unmodified) version of the modified pages are deleted. This applies for all pages concerning the file, both index pages and data pages. In this way a consistent version of the file is always guaranteed if the system should go down during the file update. Either all or none of the modifications made between open and close/checkpoint will be effective.

### 3.3.9.6. Shared file versions

If the directory immediately above the data file level (lowest level) is defined to be a "file version" directory, the file versions may be defined as shared. This means that the different file versions may share common data and index pages. While a file in a shared file version directory is open it acts like a "reentrant" and "delayed update" file. When it is closed, or a checkpoint is taken, neither the old nor the new version of updated pages are deleted. The new version of the updated pages plus the unchanged pages are set as the first file version, and the other versions are (unchanged) shifted one version up. The pages unique to the highest version, or the first one having reached its maximum version number, are deleted.

In this way the first version of the file will always represent the file state at the last checkpoint, while the higher versions will represent earlier checkpoints.

Shared files may be defined by SET-DEFAULT-FILE-ATTRIBUTES before they are created, or by SET-FILE-ATTRIBUTES to the files in a file version directory. Checkpoint is taken by CLOSE or CHECK-POINT.

## 3.3.10. File access methods

A file may be accessed by three access methods: Direct access, unstructured access and record access. Note that the access method is not a file attribute. It defines how a file is to be accessed and is in principle independent of the file type. However, some file types does not allow all access methods.

Direct access:
By this access method the data pages of a file is read or written as they are stored, including any control information and free space (deleted records). By this access method the user may decode and organize the control information in his own way.

Unstructured access:
By this access method the file is seen from the user as a contiguous unstructured string of bytes. All control information will be hidden from the user, and the boundaries between records will be invisible.

For unstructured files, direct access and unstructured access will be equivalent.

Record access:

### 7.6.4. START-PRINT

<file name>, <device name>,<=status>

A file previously stopped by calling STOP-PRINT or by specifying 'yes' at <stop for each file> (APPEND-PRINT), will be resumed.

### 7.6.5. BACKSPACE-PRINT

<file name>, <device name>, <no. of pages>,<=status>

The print will be backspaced a number of pages. If backspace past the start of the file is specified, the print is restarted completely, including any spooling header.

### 7.6.6. FORWARD-SPACE-PRINT

<file name>, <device name>, <no. of pages>,<=status>

A number of pages will be skipped.

### 7.6.7. PRINT-STATUS

<file name>,<=status>

The status of the file (position in the queue, amount left to print, format, etc.) will be returned.

### 7.6.8. PRINT-QUEUE

<device name>,<=status>

The print queue for the device will be listed. The device may specify a group of devices, as explained earlier. All files that might be output on any of the devices in the group are displayed. Unspecified device gives the whole spooling queue.

### 7.6.9. PRINT-SYSTEM-STATUS

An overall status of the print spooling system will be returned.

### 7.6.10. FORMATS

A list of the available standard formats will be returned.

## 7.7. Privileged commands

The following commands are available to operator privileged users only:

Default: The system will find a free area for the file.

<number of pages> Size of file

<partition name> must be specified if the file is to be placed in another partition than the parent directory. Default: Same partition as parent directory

<device name> may be specified instead of <partition name>

### 3.3.11.5. CREATE-INDEXED-FILE

(<file name>, <initial number of pages>, <min. last page>, <max. page>, <partition name>, <device name>, <=status>)

Create a dynamically expandable file with <initial number of pages> pages initially. If possible these pages will be allocated contiguously and no page index will be created before it is necessary because the logical pages of the file are not physically contiguous.

Unwritten logical pages higher than <min. last page> will be deleted on CLOSE if the file has been opened for WA (rewritten).

<max. page> specifies the maximum allowed logical page number in the file.

<partition name> and/or <device name> must be specified if the file is to be placed on another partition than its parent directory.

### 3.3.11.6. SET-FILE-ATTRIBUTES

(<file name>, <allocation>, <special>, <structure>, <initial number of pages>, <min. last page>, <max. page>, <=status>)

<allocation> may have the values: Permanent (1), Scratch (2), and Temporary (3).

<special> may have the values: Normal (1), Reentrant (2), Delayed update (3), Shared versions (4), Ringbuffer (5), and multiversion (6). Either all or none of the file of the files in a subdirectory may have the special attribute Shared versions or Multiversion.

<structure> may have the values: Unstructured (1) and Record structured (2). If a file is set to be recordstructured, its organization must be further specified by the SET-FILE-STRUCTURE call (section 3.3.11.11)

<initial number of pages>, <min. last page>, and <max. page> are as described for for CREATE-INDEXED-FILE (section 3.3.11.5).

## 7.4. Formats

When a file is output on a print device, it is printed according to a standard format pertaining to the device. To be able to alter the output format according to particular user needs, a format description may accompany files to be printed. Also, a format restriction may be set on a device, forcing files to comply with hardware device settings.

The user may specify such items as page size (logical, printed page size, not paper sheet size) and a message to be printed on the console terminal before the print is started. These wishes are normally met and no print delay caused. The user may further specify values regarding hardware printer settings. If these values do not comply with the current settings, as specified by the operator, the print will remain in the spooling queue until the proper changes are performed by the operator.

The spooling system will contain a number of standard formats which the user can call upon when needed. In addition to standard page sizes and printer settings, these may indicate special functions like "convert to upper case only", "drop spooling header and trailer", "apply device-dependent text-formatting", etc.

## 7.5. Public commands

### 7.5.1. Appending files

As we have seen, spooling files may be created and entered in the spooling queue by outputting to a print device. Already existing files may be appended to the queue by calling:

### 7.5.2. APPEND-PRINT

<file name>, <device name>, <no. of copies>, <priority>, <format no.>, <characters per line>, <lines per page>, <message>, <stop for each file>, <paper type>, <character set>,<=status>

The file <file name> will be appended to the spooling queue. <priority> determines the queue position, but the selection strategy applied by the spooling system needs not be based solely on this criterion. <no. of copies> copies of the file are subsequently output on <device name>, which may be a specification of a group of devices, as described earlier. If <format no.> is a positive integer, and a standard format carrying this number is already defined to the print spooling system (CREATE-FORMAT), the file will be printed according to this format. Format no. 0 is the default format for the device. Format no. -1 means that the following parameters define the format: The first two parameters define the page size. The <message> will be given on the console terminal when the file is ready for printing. If <stop for each file> is 'yes', the printer will stop before each print-out is started. The last two parameters should only be specified when the operator has given SET-FORMAT with corresponding values on the given device. Otherwise, the file will not be let through.

>3 ==> User defined structure

<organization>
1 ==> serial
2 ==>sequential
3 ==>relative sequential
4 ==> keyed sequential
5 ==> random

<record length> The length of fixed-length records: not used for variable length records, except for relative organization, in which case it indicates the maximum record size.

"Relative" organization is similar to sequential, but allows records to be accessed by an integer key - i.e. a key of 1 refers to the first record, a key of 2 refers to the second, etc.

When a relative file is created, all records are, in concept, set empty. This implies that it is not possible to insert a new record between two existing records, and therefore all writing is in fact a kind of rewriting. For this reason, sequential files, which do permit insertion, cannot be access relatively. Limited relative access can, however, be achieved using the POSITION-FILE monitor call.

Relative access is not permitted for files organized serially.

### 3.3.11.12. SET-DATA-PACKING

(<file name>, <frame packing>, <frames/group>, <group packing>, <bytes per frame>, <oflo>, <=status>)

Illegal on unstructured files.

<file name> the file name of the data file. It must be empty.

<frame packing> The frame packing density: the maximum part of the frame, expressed as a percentage, that may be used when the file is in write mode.

Default was 0.75.

<frames/group> The number of frames per group

<group packing> The maximum number of frames in a group which may be used when the file is in write mode.

Default was 75.

<bytes per frame> The number bytes per frame. This will be rounded up to the least upper bound which is an integral power of 2.

Default was 2048 (1 page).

it on a printer, thus achieving useful parallelism and user program independence of printer speed. Second, it maintains a spooling queue of all files waiting to be output, thus relieving users from having to wait for output devices to become free. Third, it contains facilities for print formatting, multiple copies, and flexible print routing.

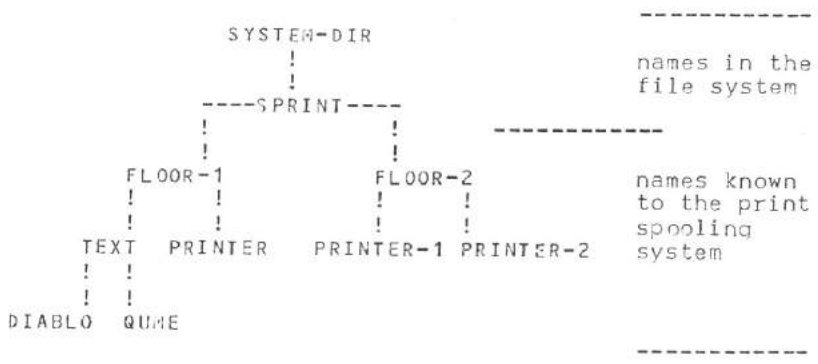
The simplest way to use the spooling system is to send output to the file named LINE-PRINTER (or other predefined names), either by copying an already existing file to it or by letting a program perform output to this special file directly. When the copying is finished, or when the LINE-PRINTER file is closed, the print will appear on an output device defined as LINE-PRINTER.

To avoid copying, an existing file may be put into the spooling queue directly by an append command.

## 7.1. Print devices

All print devices, i.e. devices connected to the print spooling system, are logically placed in a file directory. The name of this directory is installation dependent. The print device directory may contain several levels of directories with the bottom-level files representing the print devices.

Example: Let us assume that the print device directory is called SPRINT and that it resides in a directory named SYSTEM-DIR, i.e. its name is SYSTEM-DIR.SPRINT. Our system has three line printers, one on 1st floor and two on 2nd floor. There are also a Diablo and a Qume typewriter on 1st floor. We choose the names of the print devices to reflect this geographical structure:




The full name of printer 1 on 2nd floor is now:

SYSTEM-DIR.SPRINT.FLOOR-2.PRINTER-1

and of the Diablo:

SYSTEM-DIR.SPRINT.FLOOR-1.TEXT.DIABLO

#### 3.3.11.15. RESERVE-PAGES

(<object name>, <respages>, <maxpages>, <=status>)

Change pages reserved and/or maximum pages for the file or directory specified by <object name>.

Add <respages> to pages reserved for <object name>. The pages reserved are added to pages used for the parent directory. <respages> may be negative. Add <maxpages> (may be negative) to maximum number of pages for <object name>. Write access to parent directory is required.

#### 3.3.11.16. TRANSFER-PAGES

(<from object name>, <to object name>, <number of pages>, <=status>)

Transfer <number of pages> (must be positive) from <from object> (file or directory) to <to object> in the same parent directory. Write access to <from object> is required.

#### 3.3.11.17. GET-PAGES

(<object name>, <=respages>, <=maxpages>, <=usedpages>, <=pagesize>, <=status>)

Returns pages reserved, maximum number of pages, and pages actually used for <object name>. The page size in bytes on the partition containing the object is also returned.

#### 3.3.11.18. GET-PAGE-SIZE

(<partition name>, <=pagesize>, <=status>)

Return the page size in bytes on specified partition.

### 3.3.12. Monitor calls acting on open files

#### 3.3.12.1. CLOSE

(<object index>, <special action>, <=status>)

The file(s) connected to <file no.> will be closed.

<file no.>=-1 has the special meaning:
close all files not permanently opened.

<file no.>=-2 has the special meaning:
close all files

<special action flag> This parameter may be used to override the normal actions taken by close-file according to the allocation attributes.

queue until they are released by FREE-BATCH. Before such a hold command is given, the "wait for operator" flags will have no effect.

The "wait for operator" flag may be set on individual jobs by issuing a HOLD-BATCH command on a specific file. Again, only when a HOLD-BATCH with empty file name string has been issued, will the job really remain in the queue.

### 6.2.3. FREE-BATCH

<file name>,<=status>

This is the opposite to HOLD-BATCH. If the file name is an empty string, the "wait for operator" flags will no longer have any effect. If a particular file is specified, it will cause the "wait for operator" flag to be cleared on this job. Any suitable free process(es) will start processing the job(s) thus released.

### 6.2.4. SET-BATCH-LOG

<job in/out of queue> <job started> <job terminated>,<=status>

The parameters (yes or no) determine which events should be logged on the console terminal.

### 6.2.5. CREATE-BATCH-PROCESS

<program name> <=process no.>

A new batch process is created, and its process number is returned. The batch program <program name> can contain information on the job selection strategy applied by this particular process. This program may also prompt for installation dependent parameters.

### 6.2.6. DELETE-BATCH-PROCESS

<process no.>,<=status>

The batch process will be deleted when its current job (if any) is finished.

### 6.2.7. SET-BATCH-PROCESS-PRIORITY

<value> <process no.>,<=status>

Batch jobs with lower user-specified priority than <value> will not be run at all by the process. If <process no.> is zero, the hold criterion is defined. This criterion determines which jobs should automatically have their "wait for operator" flag set when they are appended. The <value> now usually signifies the biggest (in terms of processor time) job allowed to pass without the flag being set.

### 3.3.12.14. SET-RECORD-CURRENT

(<object index>, <internal address>, <=status>)

| | |
|---|---|
| <object index> | object index of opened data file |
| <internal address> | internal record address generated by FIND-RECORD or GET-RECORD-DESCRIPTION |
| <=status> | return status |

## 3.3.13. Access of partition system tables (directories etc.)

The file system is implemented as a hierarchy of separate modules (levels). The file system usually seen from user programs consists of the services available on the highest level (level 3). When the user executes a monitor call to level 3, this module is performing the appropriate function by monitor calls to the lower (more primitive) levels. The interaction between the different levels are illustrated in the figure below. The functions performed by the levels are roughly:

Level 0 performs physical device access. For some devices this level will contain buffering (disc. cache) and software to optimize the use of the hardware (minimize head movements on disc).

Level 1 is only used for the partition format. This level controls the allocation of storage space (maintains the bit table) and maintains the index hierarchy for all kinds of files (including directories and control files).

Level 2 is the name look-up manager. It is only used when objects are accessed by name (not object index) and when a change from one object to another is performed when more objects are opened simultaneously and concatenated (by * and/or + in the object name string). Level 2 is divided in part 2A and 2B. Level 2B searches for an entry in a directory on a partition corresponding to a single name (the bytes between two dots in a full name). Level 2A combines the single names to one or a set of full names. There is one level 2B process for every active partition, and one level 2A process for every active user program.

Level 3 separates access calls (READ-BLOCK/WRITE-BLOCK etc.) to different object types (partition files, channels, volumes files, terminal, etc.). New object types may be added to the system by adding an open routine and an access routine for the new object type on level 3. Level 3 contains all code and controlls all data necessary to structure partition files into records.

Directories may be opened as data files by specifying the directory name with type string 'DIRECTORY' as object name. Opening for read requires read access to the directory while opening for write is only allowed by privileged programs (system programs). Access method 0 should be specified to open the

where

<file name> is the batch job (macro file) to be started. The batch job is entered into the batch queue, and, if a suitable free batch process is found, it is started. If the macro file requires parameters, these may be appended to the file name string, separated by periods. (Example: The macro file MY.FILE should be run with the integer parameter 123 and the name parameter MY.SCRATCH. The <file name> should now read: MY.FILE.123.MY.SCRATCH.)

<output file> determines the job output file. As a default, the output will be routed to the device defined as LINE-PRINTER for the submitting terminal.

<priority> is a number, high value indicating high priority. High priority jobs will generally be run before low priority jobs, in that this priority determines the queue sequence. The selection strategy may, however, take other items, such as job size (max. time) and user class, into consideration as well. Batch processes can be reserved for jobs beyond a certain priority level.

<max. time> gives the maximum allowable processor time in minutes. Jobs exceeding this limit will be aborted.

<max. pages> gives the maximum number of pages allowed on the output device, if applicable. If this is a print spooling device, and the limit is exceeded, the job might have executed further than the print indicates due to the buffering nature of the spooling system.

<wait for operator> (yes or no) can cause the batch job to remain in the batch queue until it is released by an operator.

<message on termination> (yes or no) will tell the batch system to send a message through the Message System when the job is finished.

<urgent message on termination> (yes or no) will let the user be notified immediately that a message is waiting, if he/she is logged in.

Most parameters may be dropped by using the semicolon facility of the command processor:

APPEND-BATCH <file name> ;

Will give default values to the rest of the parameters. The default values are installation dependent.

### 6.1.5. APPEND-REMOTE-BATCH

<remote destination name> <...>

The batch job is sent to a remote system. The parameter list <...> is the same as for APPEND-BATCH.

The basic information-entity recognized by the Interprocess communication system is a message, a group of 8-bit bytes (octets) specified by the user. Some of the protocols will though recognize records, collections of messages, each terminated with a special mark. In addition to these information-entities the Interprocess communication system will also transfer control-entities, similar to the messages but specially marked, as to give control-information to the executing program, - the ownership of any Sintran IV object except for the areas - and the reference to ports and areas, - and "expedited" messages, which passes through the ordinary flow of information and may cause the receiving process to be signalled.

To initiate communication with another process, a reference to a port, fit for the communication, must be obtained from that process. This may be done in four ways: - The remote process may send the reference to such a port to a port on the first process already known to it. - The first process may inquire for a suited port to a user-written process, specially made to transfer references to wanted ports between a cluster of cooperating processes. - The first process may know the low-level name of the wanted port and inquire for a reference to it to the local low-level Name-process (one for each kernel). The latter possibility is restricted to a small group of high priority processes - and is mentioned here for the sake of completeness only. - Ultimately perhaps the most usual way; - the port may be named in the File-system name-space, and a reference may be obtained by reopening the named object in the scope of the first process.

### 3.4.2. Port types and protocols supported by the Interprocess communication system.

The "protocols" supported by the Interprocess communication system may be divided into rudimentary and higher level ones. Rudimentary protocols are the "Read-after-write"-protocol and the "Broadcast"-protocol, the higher level protocols are the "Single-receive"-, "Half-duplex-stream"- and "Full-duplex-stream"- protocols.

To prevent interaction between ports obeying different protocols, different port types are defined, so that two communicating ports must be of the same type. The port-types are nonconnected port, connected port, single-receive port, half duplex stream port and full-duplex stream port. The protocols are invoked on the ports in the same order of sequence, except for the "Read-after-write"-protocol, which is also available to connected ports, - as both the rudimentary protocols are rather mechanisms than real protocols (one operation instead of an endless sequence of interacting operations).

On the two first port types, direct messages may also be sent to known or connected ports respectively, - in addition to the communication ruled by the mentioned protocols.

#### 5.4.2.2. FIX-CONTIGUOUS

<segment no.> <=physical memory address> <=status>
A contiguous area in physical memory will be found, and the address returned.

#### 5.4.2.3. FIX-ABSOLUTE

<segment no.> <physical memory address> <=status>
The segment will be placed on a definite place in physical memory.

#### 5.4.2.4. UNFIX

<segment no.> <=status>
The pages can be swapped out.

#### 5.4.2.5. WRITE-BACK

<segment no.> <=status>
All pages which have been written into will be copied back to the disk.

#### 5.4.2.6. FORGET-SEGMENT

<segment no.> <=status>
All pages will be released, without writing back.

#### 5.4.2.7. SET-SEGMENT-LIMITS

<segment no.>, <lower bound>, <upper bound>,<=status>
Defines minimum and maximum amount of physical memory used for this segment.

#### 5.4.2.8. SET-PROCESS-LIMITS

<task name>, <lower bound>, <upper bound>,<=status>
Defines minimum and maximum amount of physical memory used for this task.

### 5.4.3. Privileged instructions

The partner, having got the turn to send, may decide whether it shall continue receiving or "take the turn" - and send. It chooses to continue receiving just by continuing, and to send by starting to send. In the former case, a system generated, empty logical record is sent to the port that originally had the turn, thus returning the turn to it. In the latter case, the turn is really changed.

Initially the port that makes the connect has got the turn to send, the other one should be listening.

### 3.4.3. Interprocess communication system commands.

The commands listed are divided into three functionally different groups; - the port opening and connect commands, which partly are subcommands under the OPEN/ CLOSE OBJECT commands, - the data transfer commands, which may be specified directly or used by higher level READ/ WRITE commands, - and the port maintenance commands, which may be addressed through a special object maintenance system (READ/ WRITE OBJECT PROFILE), common to all Sintran IV objects or partly through direct commands.

#### 3.4.3.1. Standard parameters.

Some standard parameters are used in all or mostly all commands. These are:

- <OBJECT INDEX> : The open object number of the port or area of interest. It only identifies a port locally, within a kernel, and will be produced by the Interprocess communication system on OPEN. The object index is an unsigned 16 bit integer, but only a system-defined subset of these will make sense in a specific kernel.

- <=STATUS> : Result of operation. The status consists of two unsigned 16 bit integers, which are both zero if the operation was terminated correctly. Did an error occure, will the first word of the status identify the subsystem where the error occured, while the second word identifies the level within the Interprocess communication system and the actual error.

- <TIMEOUT> : The time a program is willing to pend before it gives up the interaction specified on the port. The timeoutparameter is a signed 32 bit integer, and is described more thoroughly under the general I/O-system of Sintran IV above.

- <DATA ADDRESS> : Reference to the buffer containing or to contain data under transfer. This

1: I/O completion. If an I/O operation has been started with zero timeout (no-wait), the process will be restarted when operation is finished (only from a higher interrupt-level than the monitor-level) .

2: Started by RESTART-PROCESS.

3: Timeout in HOLD

4: Terminal-interrupt

### 5.3.20. HOLD

<time> <time unit> <=restart reason> <=status>
Execution is stopped until the process is restarted or the time is out.

The meaning of <time unit> :
(see SET-EXECUTION-TIME )

The meaning of <restart reason> :
(see WAIT)

### 5.3.21. EXIT-PROCESS

<=status>
The execution of the process is stopped and reserved resources are released. The process can be started again by the command START-PROCESS.

### 5.3.22. RESTART-PROCESS

<object index> <=status>
The given process will be restarted after a HOLD/WAIT. Restart reason will be "Started by RESTART-PROCESS" .

### 5.3.23. PHYSICAL-PROCESSES

<processor name> <=output> <=status>
A list of all processes on the given processor will be returned, together with the most important status information.

### 5.3.24. PHYSICAL-PROCESS-STATUS

<processor name> <object index> <=output> <=status>
Detailed status of the process will be returned.

kernel.

<NAME-TABLE INDEX> : Selects adequate group of names. The parameter is an unsigned 16 bit integer.

<GROUP INDEX> : Selects adequate name within the group. The parameter is an unsigned 16 bit integer.

### 3.4.3.2.2. GIVE-LOW-LEVEL-NAME.

<object index>, <name-table index>, <group index>, <=name-table index>, <=group index>, <=status>

The function creates, changes or deletes names due to the combination of input-parameters. Underneath, 1 indicates an input-parameter given, while zero indicates that the corresponding input-parameter is not given. The order of the indicators is the same as that of the parameters:

000 Not allowed.

001 Delete the corresponding local name.

010 Create (preoccupy) a group name, or (if already present) delete the corresponding group of names.

011 Delete the corresponding global name.

100 Obtain a new local name and attach it to the referenced object.

101 Create this local name and attach it to the referenced object.

110 Obtain a new global name and attach it to the referenced object.

111 Create this global name and attach it to the referenced object.

### 3.4.3.2.3. OPEN-AREA.

<data address>, <length>, <access>, <=area index>, <=length>, <=status>

The function creates an object, describing access to some specific part of the user address area of the process. The reference to this object may be transfered to other processes, thus allowing them to access the specified part as given in the original access-parameter. The "area" must be within the limits of a physical segment. The area is accessible after it has been ACCEPTED by the receiving process and until it is closed by that process. If the corresponding physical segment does not exist any more, an error-status is given when access to the area is attempted.

a process in the specified processor.

### 5.3.8. SET-SYSTEM

<system name>,<=status>
Succeeding CREATE-PROCESS and NEW-PROCESS commands will generate a process in the specified system.The first time control is given to the new process,the user will have to go through the LOGIN procedure of the foreign system.User type-ins may alternatively be given in the <command string> parameter of CREATE-PROCESS. This ,of course , also includes additional parameters required by a SYSTEM-LOGIN-PROGRAM or USER-LOGIN-PROGRAM.
Note that processor names are only known within the local system where the processors are physically present.

### 5.3.9. DETACH-PROCESS

<process name>,<output file>,<=status>
The process will immediately be detached from the terminal.It will retain its user name and file access rights.Command responses ( ie error messages) will be routed to <output file>.

### 5.3.10. ATTACH-PROCESS

<process name>,<= object index>,<=status>
The specified process is attached to terminal.It will be regarded as a child of the current process,so that this process may give control to it and delete it.If the user logs out without detaching the process,an implicit detach will occur.

### 5.3.11. START-PROCESS

<object index> <=status>
The process will be started if enabled for start (not restarted after a HOLD/WAIT).

### 5.3.12. SET-EXECUTION-TIME

<object index> <time> <time unit> <=status>
The process will be started when the given time is out.

The meaning of <time unit>:

1: milliseconds
2: seconds
3: minutes
4: hours

When control returned from a receive-command indicates that an object has been transfered, this command may be issued to accept it. If another object has been received (due to a receive-command!) in the mean-time, the former is lost.

<=LENGTH> : The length of an area, if the TYPE says so.

3.4.3.2.7. CLOSE.

<object index>, <special action flag>, <=status>

The function closes the referenced object. If the object is a port with connections to others, the connections are closed. If the port was named, either low or high level, or if some logging-function was active, including that port, the appropriate system-process is updated.

<SPECIAL ACTION FLAG>: Specifies which actions should be taken due to the close-command. Actions should be compared with those for closing opened files (chapter 3.3.12.12). The parameter is an unsigned 16 bit integer.

0 : Normal close. Disconnect is initiated on all connections and all incomming messages are ignored. Return is given after the last received disconnect confirmation.

3 : Close and delete. Same as normal close for ports.

4 : Close without special checkpoint action. Disconnect is initiated on all connections and sent - but not yet received - messages are removed. Then the port is removed without awaiting the connected port to experience the disconnects.

5 : Fast normal close. As 0, but all sent - but not received - messages are removed.

6 : Initiate close. Disconnect is initiated on all connections and the port is closed for further sending. If no connections are up, and the input- and output-queues are empty, the port is closed. Otherwise the port will be closed when the last transmission is ended, or the last disconnect-message is

### 5.2.3. RAISE-EXCEPTION

<exception object index>

An exception event is raised for this user defined object.

### 5.2.4. SET-TRAP-ADDRESS

<address>

An address to a trap record is defined. The record consists of the address of the handler program, the register block at the trapped point and some return information.

### 5.2.5. GET-EXCEPTION

<=exception object index>

This call can be used inside a trap program to identify the exception.

### 5.2.6. TRAP-EXIT

The trapped program will continue.

## 5.3. Processes

A Process is the basic independent active unit in the system.To the novice user, a process is equivalent to an active program,but a process does not terminate when the program running on it terminates.Instead,it may be used to execute a new program.A terminal that is logged in, will always have at least one process servicing it.At any one time,only one process may have control of the command stream from the terminal.This process is termed current.A process may create new processes.These are termed children processes,and the creating process is termed the parent process.

A processes may be deleted in two different ways,either implicitly,by LOGOUT, or explicitly,by a special command.Only unnamed processes are deleted by LOGOUT (A process may be given a name when it is created ). Whenever a process is deleted,the accounting system is informed about resources used.

Processes are objects defined in the object naming system,and are known to each other by an object index. It is also possible to give a process a symbolic name.The object naming system rules with directories etc of course also apply to process names.It is also possible to transfer the capability of a processes from one process to another.The object index identifying it,will generally be different in the process transfering and the process transferred to.

<=REMOTE PORT INDEX>: Specifies a temporary port reference to the remote port.

#### 3.4.3.2.10. DISCONNECT.

<port index>, <remote port index>, <=status>

The function breakes up an established connection.

When the disconnect is received in "the other end", an ACK-message is automatically returned, thus confirming the initiator that the disconnect has taken place.

### 3.4.3.3. Data transfer commands.

#### 3.4.3.3.1. SEND.

<port index>, <control word>, <data address>, <length>, <remote port index>, <timeout>, <=length>, <=message reference>, <=status>

This is one of the two central commands for transfering information through Interprocess communication process ports. A block describing the information to be transfered, - either by referencing it or by actually containing it, - is created, transfered to the destination process and linked into the input-queue of the destination port. Thus the Interprocess communication system implements a serial organization, and may be looked upon as an abstraction of a serially organized file.

Return is given either when a block containing the actual information has been linked into the receivers input-queue or when the referenced information entity has been RECEIVED (see below!), - in both cases when the space on the sender's user domain may be used once again, and the information is made the available to the receiver; - or due to timeout. In the latter case, any transfer that has been initiated will continue, but special commands must be issued to obtain the final parameters.

Note that only a limited number SENDs and RECEIVEs may be issued per port, initially one of each.

<CONTROL WORD> : Contains three flags (boolean values), coded in the rightmost part of a 16 bit word, listed as they are placed from the right:
- Expedited-flag.
- Broadcast-flag.
- Turn-flag
- Control-data-flag.
- Bounce-flag.
- Nonsecure-flag.

### 5.1.7. CREATE-SEGMENT

<open object number> <=object-index for segment> <=status>

A segment referring to an object which must have been opened beforehand by an OPEN-OBJECT will be created.<object-index for segment> is a reference to the created segment.

### 5.1.8. SET-SEGMENT

<capab.-index for segm.> <object-index for dom.> <access>, <object-index for segment> <=status>

It is possible to define a segment in current domain or in a domain which is referred to from current domain. If <object-index for domain> is equal to -1, current domain is used. The <capab.-index for segm.> defines the segment number, placing the segment in the virtual address space. The segment must be opened beforehand by a normal OPEN-OBJECT and the segment must have been created by a CREATE-SEGMENT to get the <object-index for segment>. The <access> defines the access to the segment

The segment access can be :

bit 0 : fetch capability allowed
1 : data capability allowed
2 : transfer-capability for segment allowed
3 : copy-capability for segment allowed
13: shared segment
14: parameter-access permitted
15: write permitted

### 5.1.9. SET-INDIRECT-SEGMENT

<capab.-index for segm.> <object-index for domain> , <ref. for segm. capab.> <ref. for domain capab.> <=status>

An indirect reference for inter-domain calls is established. The two first parameters define where the reference should be placed (calling domain), and the two last parameters define the contents of the reference (called domain). If <object-index for domain> is equal to -1, current domain is used.

### 5.1.10. TRANSFER-CAPABILITY

<capab.-index for 'from segm.'> , <object-index for 'from domain'> <capab.-index for 'to segment'> , <object-index for 'to domain'> <access> <=status>

The capability will be moved/copied, provided that transfer-access has been specified in SET-SEGMENT. The <access> is the segment access of the 'to segment'. The allowed values of the <access> is described in SET-SEGMENT. Only one of the accesses transfer/copy can be specified. The segment access of the 'from segment' decides the operation to perform (copy/transfer). This

the number of messages sent! The parameter is a momentary value, and more of the addressed ports may still receive the information to be transfered. When the transfer is finished (see TERMINATE-SEND), the total number of receivers is returned as =LENGTH. Due to errors, this number may be less than all connected ports.

<=MESSAGE REFERENCE>: If just a reference to the information to be sent was really transfered, and the command was timed out, the transfer will still continue. This parameter returns an identification of that transfer, and should be used both when attempting to abort the transfer, and when reading it's status (see commands below!). The message reference is a signed 16-bit integer.

3.4.3.3.2. RECEIVE.

<port index>, <data address>, <length>, <timeout>, <=control word>, <=length1>, <=length2>, <=remote port index/ =request index>, <=status>

This is the other of the two central commands for transfering information through Interprocess communication system ports. The block describing the information to be transfered, - either by referencing it or by actually containing it, - is selected, and the information is copied, either from the message-buffer or from the original buffer of the sender, to the specified buffer in the receiver user domain. If no message is present when the receive is issued, a request for input is entered into the input-queue of that port, and the process may wait for the transfer until it is timed out. On a possible transfer, data will then be copied directly between the to user domains. - Answers to QUESTIONs (See below!) (READ-AFTER-WRITE) are always copied back to the buffer in the user-domain, specified to receive the "answer".

Return is given either when an entire message-block has been read, or when the specified buffer on the user domain has been filled up, - in case of non-stream ports; - when an entire logical record has been read, or when the specified buffer on the user domain has been filled up, - in case of stream ports; - or when timed out before any transfer has been started.

If another message than the first one in the input-queue should be read, this message should be transfered to this place by means of the MOVE-command. The latter command is not available for use on stream-ports.

The message will appear on the terminal. A return message can then be typed.

# . User programs

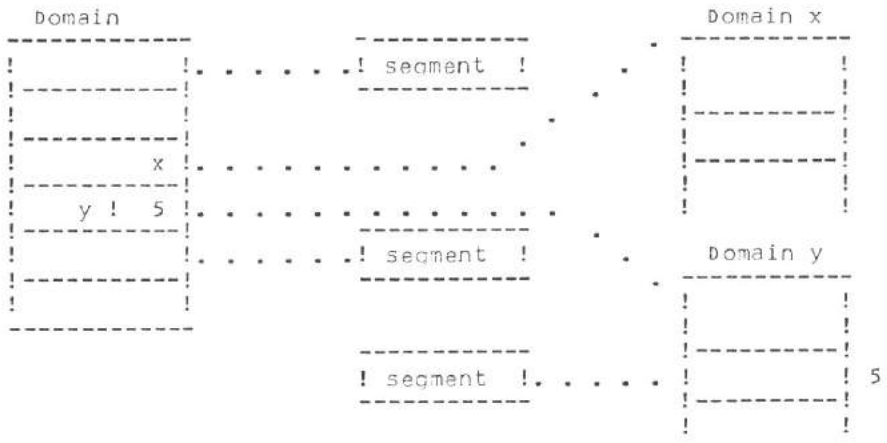
## 5.1. User addressing space

In a process there can be several addressing spaces, domains. A domain consists of all that a process is allowed to access at a certain point. At least one domain is dedicated to system functions, and one or more can be used for user programs. The domains are protected from each other, and communication is performed by means of monitor calls and common segments.

A domain consists of the instruction addressing space and the data addressing space. For NORD-500 the two separate spaces are provided by the instruction set. For NORD-100 the data space is used for addressing with the alternative page table.
For NORD-100 and NORD-500, each addressing space is divided in up to 32 segments. For NORD-100, each segment is a multiple of 2 k words blocks.
Each segment is a file. A loader puts program units into segments and combine segments to domains.

### 5.1.1. Operations on domains

Normally the Loader is used to set up the domains. However, the user can have explicit control by using a set of commands.
The domain consists of two parts, each with 32 segments:
0-31 Instruction segments
0-31 Data segments
A segment can be replaced by an indirect reference to some other domain. It can contain just the reference to the domain, or it can in addition refer to a specific segment in the other domain. The last case is used for monitor calls to the other domain.



object to be expelled, and results in a NAK-message to the sender.

<OBJECT INDEX> : Identifies the object which ownership or reference should be transfered.

<RIGHTS MASK> : The rights of an object determines the mobility of it. The object may just exist within one process, or it may be allowed to be sent away to some other process. There may exist a lot of copies of it or just one single copy. For areas, as long as just one single copy exists, this may be retractable to the originator process (Commanding TERMINATE-SEND on that area!). And finally, temporary objects may be declared, which disappears when they are no longer needed. The first four possibilities defines different degrees of mobility, with multiple copies as the highest degree of freedom, and an owned, single copy as the lowest. Retractability may be added to any single copy of an area, to assert that it - when lent away - will be returned to the first owner when it is closed, or when the first owner wants it back. The "rights" are defined in the three lowest bits of a 16 bit word:

- The transferability bit.
- The indivisibility bit.
- The retractability bit.
- The temporal bit.

The rights given initially is the most extensive set of rights allowed for that object-type, e.g. a port reference will be transferable and may be copied to any other process.

The RIGHTS MASK specified in this statement, may reduce these rights, when the appropriate bits are set, but never increase them.

3.4.3.3.4. QUESTION.

<port index>, <control word>, <data address>, <length>, <remote port index>, <return data address>, <return length>, <timeout>, <=length>, <=message reference>, <=status>

partition. This parameter may also be a file name. In that case the contents of this file will be physically copied into the destination device.

#### 3.5.2.8. PARTITION-STATISTICS

(<partition name>, <=status>)

Statistics information about the specified partition(s) is written to the command output object.

### 3.5.3. Utility commands to display and change the contents of partition system tables

#### 3.5.3.1. OBJECT-ENTRY (<dir.name> <entry no.> <=status>)

The contents of the specified object will be written to the command output file in a readable form. The <entry no.> for an entry may be found by the OBJECTS or OBJECT-STATISTICS command.

#### 3.5.3.2. CHANGE-OBJECT-ENTRY (<dir.name> <entry no.>)

This is an interactive command where the contents of the entry may be displayed and changed in octal format.

#### 3.5.3.3. PARTITION-ENTRY (<partition name> <=status>)

The contents of the specified partition entry will be written to the command file in a readable format.

#### 3.5.3.4. CHANGE-PARTITION-ENTRY (<partition name>)

This is an interactive command where the contents of the partition entry may be displayed and changed in octal format.

#### 3.5.3.5. BIT-FILE (<partition name> <=status>)

The contents of the bit file will be written to the command output file in octal format.

3.4.3.3.7. WRITE-BLOCK.

<object index>, <block no.>, <data address>, <length>, <=length>, <=status>

As this command is also described elswere in this manual, it should here be emphasized it's function when the object index designates an owned port.

No receiver address may be specified, thus the command may just be issued on connected ports without fan-out. Only serial access to data is possible.

<BLOCK NO.> : Specifies the actions that may be performed upon the port. Other values than -1 and -2 are not allowed for ports.

-1 : Append this message to the stream of messages constituting the current record. If there is no stream-connection, then just append.

-2 : Append this message to the stream of messages constituting the current record and close it. If there is no stream-connection, the parameter is equivalent to -1.

<=LENGTH> : Number of bytes (octets) actually transfered.

3.4.3.3.8. READ-BLOCK.

<object index>, <block no.>, <data address>, <length>, <=length>, <=status>

As this command is also described elswere in this manual, it should here be emphasized it's function when the object index designates an owned port.

The command may be issued on any port. Only serial access to data is possible.

<BLOCK NO.> : Specifies the actions that may be performed upon the port. Other values than -1, -2 and -4 are not allowed for ports.

-1 : Read until 'End Of Record' is encountered or the receiver buffer is filled up. For non-stream ports a message constitutes a record.

-2 : Ignore the rest of this record and read from the beginning of

to a directory, the <backup generation> parameter has no significance. Default= an empty string.

<log file> One record will be written to this file for every file that is copied. This record will contain the following information: The Corresponding entry in the drive file, The number of pages copied, The time when the copying started, Destination volume/directory name; Eventual error status from the copying.
All entries containing an unnormal error status will be written to the command output stream, regardless of the value of the <log file> parameter. Default= no logging.

If BACKUP-FILES is used for backup purposes, copying back may be done by the COPY or another BACKUP-FILES command.

#### 3.5.1.8. DIRECTORY-STATISTICS

(<dir.name>, <=status>)

Statistics information about the specified directorie(s) is written to the command output object.

#### 3.5.1.9. OBJECTS-OPENED

(<=status>)

Name and object index of all objects opened from this process will be written to the command output object.

#### 3.5.1.10. DELETE-OBJECTS

(<object name>,<manual check>)
The specified object(s) will be deleted. If manual check is true, the names of the objects will be listed on the command output stream before they are deleted, and the user must confirm the deletion of each individual one.

#### 3.5.1.11. PARTITIONS-ENTERED

(<partition name>, <=status>)

List the partition(s) entered specified by <partition name> to command output object.

### 3.5.2. System supervisor utilities

When a RECEIVED is terminated, the stati for that RECEIVE will usually be returned to the participants through the parameter-list of their commands. If a command is timed out, it may not receive the stati any more, but it can still pick them up with this command.

The return parameters have the same explanation as those belonging to the RECEIVE-command.

<MESSAGE REFERENCE> : References on output the request for input to be investigated, on input the request that has now been terminated, if the RECEIVE was really finished.

If message reference used describes a transfer not yet ended, this will be aborted. For stream-ports the request for input and all requests newer than it will be removed.

### 3.4.3.3.11. ENTER-EVENT.

<object index>, <event combination>, <= status>

This commands should be used to specify another set of elements of a list of events which the process later may wait for. There may be built one list for each domain with a separate domain object table (see CREATE DOMAIN), and the same element may just be a member of one list at the time.

If an awaited event has occured only once on an object since a WAIT-FOR-EVENT-command is executed, the element is taken out of the list. Otherwise it will remain there until either all such events that has appeared are noticed through a WAIT-FOR-EVENT-command, or it is removed by another ENTER-EVENT-command with zero as "event combination" (see below!).

<EVENT COMBINATION> : Specifies a combinations of events which the process will later wait for to appear on the specified object. Events may be "end of transmission, input", "end of transmission, output" and "message arrived". If the list already contains a set of elements implying the object, the set is redefined due to the parameter. The parameter is kept in an unsigned 16-bit integer.

### 3.4.3.3.12. WAIT-FOR-EVENT.

<object index>, <event combination>, <timeout>, <=object index>, <=event combination>, <=status>

This command may be used to wait for the event set current to the actual domain, or for the event combination specified in the two first parameters.

3.5.1.3. APPEND

(<source object(s), <dest.object(s)>, <manual check>, <=status>)
Same as copy, but the source data is appended to the destination object(s) (concatenation).

3.5.1.4. BROADCAST-FILE

(<source object>,<dest.object(s)>,<=status>)
The <source object> is copied to all objects identified by <dest.object(s)>).

3.5.1.5. OBJECT-STATISTICS

(<object name>, <creation time>, <last access time>, <last write time>, <last read time>, <sort keys>, <output format>, <=status>)

The output from this command may be used as statistics information about the objects specified by the five first parameters. It may also be used as input to other utility commands (for instance BACKUP-FILES), where it identifies the object to be acted upon.

<object name> name of the object(s) to consider. Default= * (all objects in current environment).

<creation time> creation time of the object(s) to consider. Creation time is specified by > or < followed by a eight digit time specification yy mm dd hh. Default= >0.

<last access time> last access time of the object(s) to consider. Default= >0.

<last write time> last time opened for write for the object(s) to consider. Default= >0.

<last read time> last time for read for the objects to consider. Default= >0.

<sort keys> 0-9 sort keys may be specified, the most significant first. A key specification starts with the character 1 or 9. 1 means ascending sort sequence on the following key specification and 9 means descending sort sequence. The rest of the key specification may be an unambiguous abbreviation of one of the following key names:

NAME sort on full object name.
FILE sort on object name

<PARAMETER STRING> : This is a string of parameters to be changed or deleted. The parameters are of variable length, the string itself also being of arbitrary length. The parameter follow the proposal for "Standard type identification", to be published by ND, R&D.

<=PARAMETER STRING> : This parameter returns stati for all unsuccessful updates, coded as in the input-parameter.

3.4.3.4.2. READ-PORT-PROFILE.

<object index>, <parameter string>, <=parameter string>, <=status>

This function may be used to obtain some of the information residing in the port profiles or in any of their queues. Some information is not available to the reading process, as the security of the entire system depends on it.

Examples of coding will be given in later issues of this manual.

Some commands are used so frequently, that special commands are given. These are listed below.

3.4.3.4.3. SELECT-MESSAGE.

<port index>, <access>, <key>, <=element reference>, <=status>

This function may be used to select any message of the input- or output- queues of a port.

<ACCESS> : May select the input-queue, or the output-queue.

<KEY> : Holds parameters that identify the element of the port-profile (most usually a message) searched for. Any part of the element profile may be used for identification. The parameter is coded as the parameter string of the READ WRITE-PORT-PROFILE commands.

<=ELEMENT REFERENCE>: Elements that may be selected are ports (object index = 0), connections (port reference) or messages.

assosiated with a resource lock may be used to prevent deadlocks. It is only allowed to reserve resource locks with a higher hierarchical number than the highest one previously reserved by this process. For resource locks with hierarchical number zero no checking will be done.

### 3.4.4.4. Trapobject

A trapobject, like system extensions, is allowing for functional extensions to the object naming and/or i/o system without any internal changes in the system.

When a trapobject is opened or accessed, a software trap is generated giving control to a trap handling routine. The trap handling routine will have access to the parameters in the call and may interpret them in whatever way it finds appropriate.

#### 3.4.4.4.1. CREATE-TRAP-OBJECT

(<object name>, <trap domain>, <=status>)

Create a trap object.
<trap domain> specifies the domain where the trap handling routine resides.

### 3.4.4.5. Area

An area is a buffer in the domain of one process that may be accessed from another process. Thus, an area is used for interprocess communication. Access of a "remote area" is done by READ-BLOCK and WRITE-BLOCK with addressing as on unstructured files.

### 3.4.4.6. OPEN-AREA.

<data address>, <length>, <access>, <=object index>, <=length>, <=status>

The function creates an object, describing access to some specific part of the user address area of the process. The reference to this object may be transfered to other processes, thus allowing them to access the specified part as given in the original access-parameter. The "area" must be within the limits of a physical segment. The area is accessible after it has been REOPENED by the receiving process and until it is closed by that process. If the corresponding physical segment does not exist any more, an error-status is given when access to the area is attempted.

The usual READ/- WRITE-BLOCK commands of the file-system are utilized for access of a referenced area, and using these command high-speed, site-independent transfer between process may be obtained.

<AREA REFERENCE> : Denotes the start of the area to be

is inserted in the front of or last in the input-queue, dependant of it's priority (expedited or not). The 'sender' process will be updated, and the message will be definitely out of the scope of the first process. In this case, the function corresponds to the XFWD-function of the XMSG.

<MESSAGE REF. 1> : Defines the message to be moved.

>=0: Designates a message previously selected with a SELECT-MESSAGE command.

-1 : Designates the CURRENT MESSAGE of that port.

-2 : Designates the PREVIOUS MESSAGE of that port.

<MESSAGE REF. 2> : Defines where in the input-queue of the 'new port' the message should be inserted. The parameter has no effect, if the 'receiver port' is not owned by the process issuing the command.

>=0: The message is inserted into the queue after the message described. This message has previously been selected with a SELECT-MESSAGE command.

-1 : The message is inserted after the CURRENT MESSAGE of the 'new port'.

-2 : The message is inserted into the front end of the queue.

-3 : The message is appended to the back end of the queue.

3.4.3.4.6. RESET-PORT.

<port index>, <access>, <status>

The function resets the input-part, output-part, or both of a port, due to the access-parameter. In the two former cases, only all elements of the appropriate queues are removed, and connections and waiting-states affecting the direction are cleared, - while in the latter case, all information on that port is erased, and system-/ process-default-values are reinserted where appropriate.

is inserted in the front of or last in the input-queue, dependant of it's priority (expedited or not). The 'sender' process will be updated, and the message will be definitely out of the scope of the first process. In this case, the function corresponds to the XFWD-function of the XMSG.

<MESSAGE REF. 1> : Defines the message to be moved.

>=0: Designates a message previously selected with a SELECT-MESSAGE command.

-1 : Designates the CURRENT MESSAGE of that port.

-2 : Designates the PREVIOUS MESSAGE of that port.

<MESSAGE REF. 2> : Defines where in the input-queue of the 'new port' the message should be inserted. The parameter has no effect, if the 'receiver port' is not owned by the process issuing the command.

>=0: The message is inserted into the queue after the message described. This message has previously been selected with a SELECT-MESSAGE command.

-1 : The message is inserted after the CURRENT MESSAGE of the 'new port'.

-2 : The message is inserted into the front end of the queue.

-3 : The message is appended to the back end of the queue.

### 3.4.3.4.6. RESET-PORT.

<port index>, <access>, <status>

The function resets the input-part, output-part, or both of a port, due to the access-parameter. In the two former cases, only all elements of the appropriate queues are removed, and connections and waiting-states affecting the direction are cleared, - while in the latter case, all information on that port is erased, and system-/ process-default-values are reinserted where appropriate.

assosiated with a resource lock may be used to prevent deadlocks. It is only allowed to reserve resource locks with a higher hierarchical number than the highest one previously reserved by this process. For resource locks with hierarchical number zero no checking will be done.

### 3.4.4.4. Trapobject

A trapobject, like system extensions, is allowing for functional extensions to the object naming and/or i/o system without any internal changes in the system.

When a trapobject is opened or accessed, a software trap is generated giving control to a trap handling routine. The trap handling routine will have access to the parameters in the call and may interpret them in whatever way it finds appropriate.

#### 3.4.4.4.1. CREATE-TRAP-OBJECT

(<object name>, <trap domain>, <=status>)

Create a trap object.
<trap domain> specifies the domain where the trap handling routine resides.

### 3.4.4.5. Area

An area is a buffer in the domain of one process that may be accessed from another process. Thus, an area is used for interprocess communication. Access of a "remote area" is done by READ-BLOCK and WRITE-BLOCK with addressing as on unstructured files.

### 3.4.4.6. OPEN-AREA

<data address>, <length>, <access>, <=object index>, <=length>, <=status>

The function creates an object, describing access to some specific part of the user address area of the process. The reference to this object may be transfered to other processes, thus allowing them to access the specified part as given in the original access-parameter. The "area" must be within the limits of a physical segment. The area is accessible after it has been REOPENED by the receiving process and until it is closed by that process. If the corresponding physical segment does not exist any more, an error-status is given when access to the area is attempted.

The usual READ/- WRITE-BLOCK commands of the file-system are utilized for access of a referenced area, and using these command high-speed, site-independent transfer between process may be obtained.

<AREA REFERENCE> : Denotes the start of the area to be

<PARAMETER STRING> : This is a string of parameters to be changed or deleted. The parameters are of variable length, the string itself also being of arbitrary length. The parameter follow the proposal for "Standard type identification", to be published by ND, R&D.

<=PARAMETER STRING> : This parameter returns stati for all unsuccessful updates, coded as in the input-parameter.

3.4.3.4.2. READ-PORT-PROFILE.

<object index>, <parameter string>, <=parameter string>, <=status>

This function may be used to obtain some of the information residing in the port profiles or in any of their queues. Some information is not available to the reading process, as the security of the entire system depends on it.

Examples of coding will be given in later issues of this manual.

Some commands are used so frequently, that special commands are given. These are listed below.

3.4.3.4.3. SELECT-MESSAGE.

<port index>, <access>, <key>, <=element reference>, <=status>

This function may be used to select any message of the input- or output- queues of a port.

<ACCESS> : May select the input-queue, or the output-queue.

<KEY> : Holds parameters that identify the element of the port-profile (most usually a message) searched for. Any part of the element profile may be used for identification. The parameter is coded as the parameter string of the READ WRITE-PORT-PROFILE commands.

<=ELEMENT REFERENCE>: Elements that may be selected are ports (object index = 0), connections (port reference) or messages.

### 3.5.1.3. APPEND

(<source object(s), <dest.object(s)>, <manual check>, <=status>)
Same as copy, but the source data is appended to the destination object(s) (concatenation).

### 3.5.1.4. BROADCAST-FILE

(<source object>,<dest.object(s)>,<=status>)
The <source object> is copied to all objects identified by <dest.object(s)>).

### 3.5.1.5. OBJECT-STATISTICS

(<object name>, <creation time>, <last access time>, <last write time>, <last read time>, <sort keys>, <output format>, <=status>)

The output from this command may be used as statistics information about the objects specified by the five first parameters. It may also be used as input to other utility commands (for instance BACKUP-FILES), where it identifies the object to be acted upon.

<object name> name of the object(s) to consider. Default= * (all objects in current environment).

<creation time> creation time of the object(s) to consider. Creation time is specified by > or < followed by a eight digit time specification yy mm dd hh. Default= >0.

<last access time> last access time of the object(s) to consider. Default= >0.

<last write time> last time opened for write for the object(s) to consider. Default= >0.

<last read time> last time for read for the objects to consider. Default= >0.

<sort keys> 0-9 sort keys may be specified, the most significant first. A key specification starts with the character 1 or 9. 1 means ascending sort sequence on the following key specification and 9 means descending sort sequence. The rest of the key specification may be an unambiguous abbreviation of one of the following key names:

NAME sort on full object name.
FILE sort on object name

When a RECEIVED is terminated, the stati for that RECEIVE will usually be returned to the participants through the parameter-list of their commands. If a command is timed out, it may not receive the stati any more, but it can still pick them up with this command.

The return parameters have the same explanation as those belonging to the RECEIVE-command.

<MESSAGE REFERENCE> : References on output the request for input to be investigated, on input the request that has now been terminated, if the RECEIVE was really finished.

If message reference used describes a transfer not yet ended, this will be aborted. For stream-ports the request for input and all requests newer than it will be removed.

### 3.4.3.3.11. ENTER-EVENT.

<object index>, <event combination>, <= status>

This commands should be used to specify another set of elements of a list of events which the process later may wait for. There may be built one list for each domain with a separate domain object table (see CREATE DOMAIN), and the same element may just be a member of one list at the time.

If an awaited event has occured only once on an object since a WAIT-FOR-EVENT-command is executed, the element is taken out of the list. Otherwise it will remain there until either all such events that has appeared are noticed through a WAIT-FOR-EVENT-command, or it is removed by another ENTER-EVENT-command with zero as "event combination" (see below!).

<EVENT COMBINATION> : Specifies a combinations of events which the process will later wait for to appear on the specified object. Events may be "end of transmission, input", "end of transmission, output" and "message arrived". If the list already contains a set of elements implying the object, the set is redefined due to the parameter. The parameter is kept in an unsigned 16-bit integer.

### 3.4.3.3.12. WAIT-FOR-EVENT.

<object index>, <event combination>, <timeout>, <=object index>, <=event combination>, <=status>

This command may be used to wait for the event set current to the actual domain, or for the event combination specified in the two first parameters.

to a directory, the <backup generation> parameter has no significance. Default= an empty string.

<log file> One record will be written to this file for every file that is copied. This record will contain the following information: The corresponding entry in the drive file, The number of pages copied, The time when the copying started, Destination volume/directory name; Eventual error status from the copying.
All entries containing an unnormal error status will be written to the command output stream, regardless of the value of the <log file> parameter. Default= no logging.

If BACKUP-FILES is used for backup purposes, copying back may be done by the COPY or another BACKUP-FILES command.

### 3.5.1.8. DIRECTORY-STATISTICS

(<dir.name>, <=status>)

Statistics information about the specified directorie(s) is written to the command output object.

### 3.5.1.9. OBJECTS-OPENED

(<=status>)

Name and object index of all objects opened from this process will be written to the command output object.

### 3.5.1.10. DELETE-OBJECTS

(<object name>,<manual check>)
The specified object(s) will be deleted. If manual check is true, the names of the objects will be listed on the command output stream before they are deleted, and the user must confirm the deletion of each individual one.

### 3.5.1.11. PARTITIONS-ENTERED

(<partition name>, <=status>)

List the partition(s) entered specified by <partition name> to command output object.

## 3.5.2. System supervisor utilities

3.4.3.3.7. WRITE-BLOCK.

<object index>, <block no.>, <data address>, <length>, <=length>, <=status>

As this command is also described elswere in this manual, it should here be emphasized it's function when the object index designates an owned port.

No receiver address may be specified, thus the command may just be issued on connected ports without fan-out. Only serial access to data is possible.

<BLOCK NO.> : Specifies the actions that may be performed upon the port. Other values than -1 and -2 are not allowed for ports.

-1 : Append this message to the stream of messages constituting the current record. If there is no stream-connection, then just append.

-2 : Append this message to the stream of messages constituting the current record and close it. If there is no stream-connection, the parameter is equivalent to -1.

<=LENGTH> : Number of bytes (octets) actually transfered.

3.4.3.3.8. READ-BLOCK.

<object index>, <block no.>, <data address>, <length>, <=length>, <=status>

As this command is also described elswere in this manual, it should here be emphasized it's function when the object index designates an owned port.

The command may be issued on any port. Only serial access to data is possible.

<BLOCK NO.> : Specifies the actions that may be performed upon the port. Other values than -1, -2 and -4 are not allowed for ports.

-1 : Read until 'End Of Record' is encountered or the receiver buffer is filled up. For non-stream ports a message constitutes a record.

-2 : Ignore the rest of this record and read from the beginning of

partition. This parameter may also be a file name. In that case the contents of this file will be physically copied into the destination device.

#### 3.5.2.8. PARTITION-STATISTICS

(<partition name>, <=status>)

Statistics information about the specified partition(s) is written to the command output object.

### 3.5.3. Utility commands to display and change the contents of partition system tables

#### 3.5.3.1. OBJECT-ENTRY (<dir.name> <entry no.> <=status>)

The contents of the specified object will be written to the command output file in a readable form. The <entry no.> for an entry may be found by the OBJECTS or OBJECT-STATISTICS command.

#### 3.5.3.2. CHANGE-OBJECT-ENTRY (<dir.name> <entry no.>)

This is an interactive command where the contents of the entry may be displayed and changed in octal format.

#### 3.5.3.3. PARTITION-ENTRY (<partition name> <=status>)

The contents of the specified partition entry will be written to the command file in a readable format.

#### 3.5.3.4. CHANGE-PARTITION-ENTRY (<partition name>)

This is an interactive command where the contents of the partition entry may be displayed and changed in octal format.

#### 3.5.3.5. BIT-FILE (<partition name> <=status>)

The contents of the bit file will be written to the command output file in octal format.

object to be expelled, and results in a NAK-message to the sender.

<OBJECT INDEX> : Identifies the object which ownership or reference should be transfered.

<RIGHTS MASK> : The rights of an object determines the mobility of it. The object may just exist within one process, or it may be allowed to be sent away to some other process. There may exist a lot of copies of it or just one single copy. For areas, as long as just one single copy exists, this may be retractable to the originator process (Commanding TERMINATE-SEND on that area!). And finally, temporary objects may be declared, which disappears when they are no longer needed. The first four possibilities defines different degrees of mobility, with multiple copies as the highest degree of freedom, and an owned, single copy as the lowest. Retractability may be added to any single copy of an area, to assert that it - when lent away - will be returned to the first owner when it is closed, or when the first owner wants it back. The "rights" are defined in the three lowest bits of a 16 bit word:

- The transferability bit.
- The indivisibility bit.
- The retractability bit.
- The temporal bit.

The rights given initially is the most extensive set of rights allowed for that object-type, e.g. a port reference will be transferable and may be copied to any other process.

The RIGHTS MASK specified in this statement, may reduce these rights, when the appropriate bits are set, but never increase them.

3.4.3.3.4. QUESTION.

<port index>, <control word>, <data address>, <length>, <remote port index>, <return data address>, <return length>, <timeout>, <=length>, <=message reference>, <=status>

The message will appear on the terminal. A return message can then be typed.

# 5. User programs

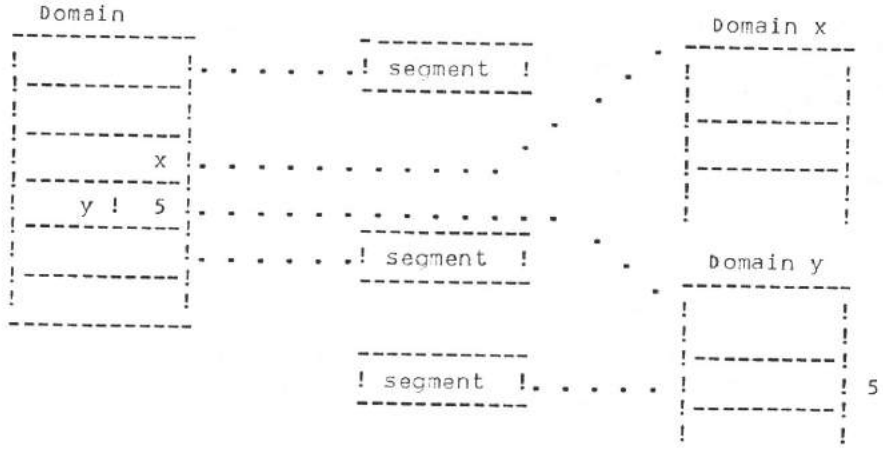
## 5.1. User addressing space

In a process there can be several addressing spaces, domains. A domain consists of all that a process is allowed to access at a certain point. At least one domain is dedicated to system functions, and one or more can be used for user programs. The domains are protected from each other, and communication is performed by means of monitor calls and common segments.

A domain consists of the instruction addressing space and the data addressing space. For NORD-500 the two separate spaces are provided by the instruction set. For NORD-100 the data space is used for addressing with the alternative page table.
For NORD-100 and NORD-500, each addressing space is divided in up to 32 segments. For NORD-100, each segment is a multiple of 2 k words blocks.
Each segment is a file. A loader puts program units into segments and combine segments to domains.

### 5.1.1. Operations on domains

Normally the Loader is used to set up the domains. However, the user can have explicit control by using a set of commands.
The domain consists of two parts, each with 32 segments:
0-31 Instruction segments
0-31 Data segments
A segment can be replaced by an indirect reference to some other domain. It can contain just the reference to the domain, or it can in addition refer to a specific segment in the other domain. The last case is used for monitor calls to the other domain.



the number of messages sent! The parameter is a momentary value, and more of the addressed ports may still receive the information to be transfered. When the transfer is finished (see TERMINATE-SEND), the total number of receivers is returned as =LENGTH. Due to errors, this number may be less than all connected ports.

<=MESSAGE REFERENCE>: If just a reference to the information to be sent was really transfered, and the command was timed out, the transfer will still continue. This parameter returns an identification of that transfer, and should be used both when attempting to abort the transfer, and when reading it's status (see commands below!). The message reference is a signed 16-bit integer.

3.4.3.3.2. RECEIVE.

<port index>, <data address>, <length>, <timeout>, <=control word>, <=length1>, <=length2>, <=remote port index/ =request index>, <=status>

This is the other of the two central commands for transfering information through Interprocess communication system ports. The block describing the information to be transfered, - either by referencing it or by actually containing it, - is selected, and the information is copied, either from the message-buffer or from the original buffer of the sender, to the specified buffer in the receiver user domain. If no message is present when the receive is issued, a request for input is entered into the input-queue of that port, and the process may wait for the transfer until it is timed out. On a possible transfer, data will then be copied directly between the to user domains. - Answers to QUESTIONs (See below!) (READ-AFTER-WRITE) are always copied back to the buffer in the user-domain, specified to receive the "answer".

Return is given either when an entire message-block has been read, or when the specified buffer on the user domain has been filled up, - in case of non-stream ports; - when an entire logical record has been read, or when the specified buffer on the user domain has been filled up, - in case of stream ports; - or when timed out before any transfer has been started.

If another message than the first one in the input-queue should be read, this message should be transfered to this place by means of the MOVE-command. The latter command is not available for use on stream-ports.

### 5.1.7. CREATE-SEGMENT

<open object number> <=object-index for segment> <=status>

A segment referring to an object which must have been opened beforehand by an OPEN-OBJECT will be created. <object-index for segment> is a reference to the created segment.

### 5.1.8. SET-SEGMENT

<capab.-index for segm.> <object-index for dom.> <access>, <object-index for segment> <=status>

It is possible to define a segment in current domain or in a domain which is referred to from current domain. If <object-index for domain> is equal to -1, current domain is used. The <capab.-index for segm.> defines the segment number, placing the segment in the virtual address space. The segment must be opened beforehand by a normal OPEN-OBJECT and the segment must have been created by a CREATE-SEGMENT to get the <object-index for segment>. The <access> defines the access to the segment

The segment access can be :

bit 0 : fetch capability allowed
1 : data capability allowed
2 : transfer-capability for segment allowed
3 : copy-capability for segment allowed
13: shared segment
14: parameter-access permitted
15: write permitted

### 5.1.9. SET-INDIRECT-SEGMENT

<capab.-index for segm.> <object-index for domain> , <ref. for segm. capab.> <ref. for domain capab.> <=status>

An indirect reference for inter-domain calls is established. The two first parameters define where the reference should be placed (calling domain), and the two last parameters define the contents of the reference (called domain). If <object-index for domain> is equal to -1, current domain is used.

### 5.1.10. TRANSFER-CAPABILITY

<capab.-index for 'from segm.'> , <object-index for 'from domain'> <capab.-index for 'to segment'> , <object-index for 'to domain'> <access> <=status>

The capability will be moved/copied, provided that transfer-access has been specified in SET-SEGMENT. The <access> is the segment access of the 'to segment'. The allowed values of the <access> is described in SET-SEGMENT. Only one of the accesses transfer/copy can be specified. The segment access of the 'from segment' decides the operation to perform (copy/transfer). This

<=REMOTE PORT INDEX>: Specifies a temporary port reference to the remote port.

#### 3.4.3.2.10. DISCONNECT.

<port index>, <remote port index>, <=status>

The function breakes up an established connection.

When the disconnect is received in "the other end", an ACK-message is automatically returned, thus confirming the initiator that the disconnect has taken place.

### 3.4.3.3. Data transfer commands.

#### 3.4.3.3.1. SEND.

<port index>, <control word>, <data address>, <length>, <remote port index>, <timeout>, <=length>, <=message reference>, <=status>

This is one of the two central commands for transfering information through Interprocess communication process ports. A block describing the information to be transfered, - either by referencing it or by actually containing it, - is created, transfered to the destination process and linked into the input-queue of the destination port. Thus the Interprocess communication system implements a serial organization, and may be looked upon as an abstraction of a serially organized file.

Return is given either when a block containing the actual information has been linked into the receivers input-queue or when the referenced information entity has been RECEIVED (see below!), - in both cases when the space on the sender's user domain may be used once again, and the information is made the available to the receiver; - or due to timeout. In the latter case, any transfer that has been initiated will continue, but special commands must be issued to obtain the final parameters.

Note that only a limited number SENDs and RECEIVEs may be issued per port, initially one of each.

<CONTROL WORD> : Contains three flags (boolean values), coded in the rightmost part of a 16 bit word, listed as they are placed from the right:
- Expedited-flag.
- Broadcast-flag.
- Turn-flag
- Control-data-flag.
- Bounce-flag.
- Nonsecure-flag.

### 5.2.3. RAISE-EXCEPTION

<exception object index>

An exception event is raised for this user defined object.

### 5.2.4. SET-TRAP-ADDRESS

<address>

An address to a trap record is defined. The record consists of the address of the handler program, the register block at the trapped point and some return information.

### 5.2.5. GET-EXCEPTION

<=exception object index>

This call can be used inside a trap program to identify the exception.

### 5.2.6. TRAP-EXIT

The trapped program will continue.

## 5.3. Processes

A Process is the basic independent active unit in the system.To the novice user, a process is equivalent to an active program,but a process does not terminate when the program running on it terminates.Instead,it may be used to execute a new program.A terminal that is logged in, will always have at least one process servicing it.At any one time,only one process may have control of the command stream from the terminal.This process is termed current.A process may create new processes.These are termed children processes,and the creating process is termed the parent process.

A processes may be deleted in two different ways,either implicitly,by LOGOUT, or explicitly,by a special command.Only unnamed processes are deleted by LOGOUT (A process may be given a name when it is created ). Whenever a process is deleted,the accounting system is informed about resources used.

Processes are objects defined in the object naming system,and are known to each other by an object index. It is also possible to give a process a symbolic name.The object naming system rules with directories etc of course also apply to process names.It is also possible to transfer the capability of a processes from one process to another.The object index identifying it,will generally be different in the process transfering and the process transferred to.

When control returned from a receive-command indicates that an object has been transfered, this command may be issued to accept it. If another object has been received (due to a receive-command!) in the mean-time, the former is lost.

<=LENGTH> : The length of an area, if the TYPE says so.

3.4.3.2.7. CLOSE.

<object index>, <special action flag>, <=status>

The function closes the referenced object. If the object is a port with connections to others, the connections are closed. If the port was named, either low or high level, or if some logging-function was active, including that port, the appropriate system-process is updated.

<SPECIAL ACTION FLAG>: Specifies which actions should be taken due to the close-command. Actions should be compared with those for closing opened files (chapter 3.3.12.12). The parameter is an unsigned 16 bit integer.

0 : Normal close. Disconnect is initiated on all connections and all incomming messages are ignored. Return is given after the last received disconnect confirmation.

3 : Close and delete. Same as normal close for ports.

4 : Close without special checkpoint action. Disconnect is initiated on all connections and sent - but not yet received - messages are removed. Then the port is removed without awaiting the connected port to experience the disconnects.

5 : Fast normal close. As 0, but all sent - but not received - messages are removed.

6 : Initiate close. Disconnect is initiated on all connections and the port is closed for further sending. If no connections are up, and the input- and output-queues are empty, the port is closed. Otherwise the port will be closed when the last transmission is ended, or the last disconnect-message is

a process in the specified processor.

### 5.3.8. SET-SYSTEM

<system name>,<=status>
Succeeding CREATE-PROCESS and NEW-PROCESS commands will generate a process in the specified system.The first time control is given to the new process,the user will have to go through the LOGIN procedure of the foreign system.User type-ins may alternatively be given in the <command string> parameter of CREATE-PROCESS. This ,of course , also includes additional parameters required by a SYSTEM-LOGIN-PROGRAM or USER-LOGIN-PROGRAM.
Note that processor names are only known within the local system where the processors are physically present.

### 5.3.9. DETACH-PROCESS

<process name>,<output file>,<=status>
The process will immediately be detached from the terminal.It will retain its user name and file access rights.Command responses ( ie error messages) will be routed to <output file>.

### 5.3.10. ATTACH-PROCESS

<process name>,<= object index>,<=status>
The specified process is attached to terminal.It will be regarded as a child of the current process,so that this process may give control to it and delete it.If the user logs out without detaching the process,an implicit detach will occur.

### 5.3.11. START-PROCESS

<object index> <=status>
The process will be started if enabled for start (not restarted after a HOLD/WAIT).

### 5.3.12. SET-EXECUTION-TIME

<object index> <time> <time unit> <=status>
The process will be started when the given time is out.

The meaning of <time unit>:

1: milliseconds
2: seconds
3: minutes
4: hours

kernel.

<NAME-TABLE INDEX> : Selects adequate group of names. The parameter is an unsigned 16 bit integer.

<GROUP INDEX> : Selects adequate name within the group. The parameter is an unsigned 16 bit integer.

3.4.3.2.2. GIVE-LOW-LEVEL-NAME.

<object index>, <name-table index>, <group index>, <=name-table index>, <=group index>, <=status>

The function creates, changes or deletes names due to the combination of input-parameters. Underneath, 1 indicates an input-parameter given, while zero indicates that the corresponding input-parameter is not given. The order of the indicators is the same as that of the parameters:

000 Not allowed.

001 Delete the corresponding local name.

010 Create (preoccupy) a group name, or (if already present) delete the corresponding group of names.

011 Delete the corresponding global name.

100 Obtain a new local name and attach it to the referenced object.

101 Create this local name and attach it to the referenced object.

110 Obtain a new global name and attach it to the referenced object.

111 Create this global name and attach it to the referenced object.

3.4.3.2.3. OPEN-AREA.

<data address>, <length>, <access>, <=area index>, <=length>, <=status>

The function creates an object, describing access to some specific part of the user address area of the process. The reference to this object may be transfered to other processes, thus allowing them to access the specified part as given in the original access-parameter. The "area" must be within the limits of a physical segment. The area is accessible after it has been ACCEPTED by the receiving process and until it is closed by that process. If the corresponding physical segment does not exist any more, an error-status is given when access to the area is attempted.

1: I/O completion. If an I/O operation has been started with zero timeout (no-wait), the process will be restarted when operation is finished (only from a higher interrupt-level than the monitor-level) .

2: Started by RESTART-PROCESS.

3: Timeout in HOLD

4: Terminal-interrupt

### 5.3.20. HOLD

<time> <time unit> <=restart reason> <=status>
Execution is stopped until the process is restarted or the time is out.

The meaning of <time unit> :
(see SET-EXECUTION-TIME )

The meaning of <restart reason> :
(see WAIT)

### 5.3.21. EXIT-PROCESS

<=status>
The execution of the process is stopped and reserved resources are released. The process can be started again by the command START-PROCESS.

### 5.3.22. RESTART-PROCESS

<object index> <=status>
The given process will be restarted after a HOLD/WAIT. Restart reason will be "Started by RESTART-PROCESS" .

### 5.3.23. PHYSICAL-PROCESSES

<processor name> <=output> <=status>
A list of all processes on the given processor will be returned, together with the most important status information.

### 5.3.24. PHYSICAL-PROCESS-STATUS

<processor name> <object index> <=output> <=status>
Detailed status of the process will be returned.

The partner, having got the turn to send, may decide whether it shall continue receiving or "take the turn" - and send. It chooses to continue receiving just by continuing, and to send by starting to send. In the former case, a system generated, empty logical record is sent to the port that originally had the turn, thus returning the turn to it. In the latter case, the turn is really changed.

Initially the port that makes the connect has got the turn to send, the other one should be listening.

### 3.4.5. Interprocess communication system commands.

The commands listed are divided into three functionally different groups; - the port opening and connect commands, which partly are subcommands under the OPEN/ CLOSE OBJECT commands, - the data transfer commands, which may be specified directly or used by higher level READ/ WRITE commands, - and the port maintenance commands, which may be addressed through a special object maintenance system (READ/ WRITE OBJECT PROFILE), common to all Sintran IV objects or partly through direct commands.

#### 3.4.5.1. Standard parameters.

Some standard parameters are used in all or mostly all commands. These are:

- <OBJECT INDEX> : The open object number of the port or area of interest. It only identifies a port locally, within a kernel, and will be produced by the Interprocess communication system on OPEN. The object index is an unsigned 16 bit integer, but only a system-defined subset of these will make sense in a specific kernel.

- <=STATUS> : Result of operation. The status consists of two unsigned 16 bit integers, which are both zero if the operation was terminated correctly. Did an error occure, will the first word of the status identify the subsystem where the error occured, while the second word identifies the level within the Interprocess communication system and the actual error.

- <TIMEOUT> : The time a program is willing to pend before it gives up the interaction specified on the port. The timeoutparameter is a signed 32 bit integer, and is described more thoroughly under the general I/O-system of Sintran IV above.

- <DATA ADDRESS> : Reference to the buffer containing or to contain data under transfer. This

#### 5.4.2.2. FIX-CONTIGUOUS

<segment no.> <=physical memory address> <=status>
A contiguous area in physical memory will be found, and the address returned.

#### 5.4.2.3. FIX-ABSOLUTE

<segment no.> <physical memory address> <=status>
The segment will be placed on a definite place in physical memory.

#### 5.4.2.4. UNFIX

<segment no.> <=status>
The pages can be swapped out.

#### 5.4.2.5. WRITE-BACK

<segment no.> <=status>
All pages which have been written into will be copied back to the disk.

#### 5.4.2.6. FORGET-SEGMENT

<segment no.> <=status>
All pages will be released, without writing back.

#### 5.4.2.7. SET-SEGMENT-LIMITS

<segment no.>, <lower bound>, <upper bound>,<=status>
Defines minimum and maximum amount of physical memory used for this segment.

#### 5.4.2.8. SET-PROCESS-LIMITS

<task name>, <lower bound>, <upper bound>,<=status>
Defines minimum and maximum amount of physical memory used for this task.

### 5.4.3. Privileged instructions

The basic information-entity recognized by the Interprocess communication system is a message, a group of 8-bit bytes (octets) specified by the user. Some of the protocols will though recognize records, collections of messages, each terminated with a special mark. In addition to these information-entities the Interprocess communication system will also transfer control-entities, similar to the messages but specially marked, as to give control-information to the executing program, - the ownership of any Sintran IV object except for the areas - and the reference to ports and areas, - and "expedited" messages, which passes through the ordinary flow of information and may cause the receiving process to be signalled.

To initiate communication with another process, a reference to a port, fit for the communication, must be obtained from that process. This may be done in four ways: - The remote process may send the reference to such a port to a port on the first process already known to it. - The first process may inquire for a suited port to a user-written process, specially made to transfer references to wanted ports between a cluster of cooperating processes. - The first process may know the low-level name of the wanted port and inquire for a reference to it to the local low-level Name-process (one for each kernel). The latter possibility is restricted to a small group of high priority processes - and is mentioned here for the sake of completeness only. - Ultimately perhaps the most usual way; - the port may be named in the File-system name-space, and a reference may be obtained by reopening the named object in the scope of the first process.

### 3.4.2. Port types and protocols supported by the Interprocess communication system.

The "protocols" supported by the Interprocess communication system may be divided into rudimentary and higher level ones. Rudimentary protocols are the "Read-after-write"-protocol and the "Broadcast"-protocol, the higher level protocols are the "Single-receive"-, "Half-duplex-stream"- and "Full-duplex-stream"- protocols.

To prevent interaction between ports obeying different protocols, different port types are defined, so that two communicating ports must be of the same type. The port-types are nonconnected port, connected port, single-receive port, half duplex stream port and full-duplex stream port. The protocols are invoked on the ports in the same order of sequence, except for the "Read-after-write"-protocol, which is also available to connected ports, - as both the rudimentary protocols are rather mechanisms than real protocols (one operation instead of an endless sequence of interacting operations).

On the two first port types, direct messages may also be sent to known or connected ports respectively, - in addition to the communication ruled by the mentioned protocols.

where

<file name> is the batch job (macro file) to be started. The batch job is entered into the batch queue, and, if a suitable free batch process is found, it is started. If the macro file requires parameters, these may be appended to the file name string, separated by periods. (Example: The macro file MY.FILE should be run with the integer parameter 123 and the name parameter MY.SCRATCH. The <file name> should now read: MY.FILE.123.MY.SCRATCH.)

<output file> determines the job output file. As a default, the output will be routed to the device defined as LINE-PRINTER for the submitting terminal.

<priority> is a number, high value indicating high priority. High priority jobs will generally be run before low priority jobs, in that this priority determines the queue sequence. The selection strategy may, however, take other items, such as job size (max. time) and user class, into consideration as well. Batch processes can be reserved for jobs beyond a certain priority level.

<max. time> gives the maximum allowable processor time in minutes. Jobs exceeding this limit will be aborted.

<max. pages> gives the maximum number of pages allowed on the output device, if applicable. If this is a print spooling device, and the limit is exceeded, the job might have executed further than the print indicates due to the buffering nature of the spooling system.

<wait for operator> (yes or no) can cause the batch job to remain in the batch queue until it is released by an operator.

<message on termination> (yes or no) will tell the batch system to send a message through the Message System when the job is finished.

<urgent message on termination> (yes or no) will let the user be notified immediately that a message is waiting, if he/she is logged in.

Most parameters may be dropped by using the semicolon facility of the command processor:

```
APPEND-BATCH <file name> ;
```

Will give default values to the rest of the parameters. The default values are installation dependent.

### 6.1.3. APPEND-REMOTE-BATCH

```
<remote destination name> <...>
```

The batch job is sent to a remote system. The parameter list <...> is the same as for APPEND-BATCH.

3.3.12.14. SET-RECORD-CURRENT

(<object index>, <internal address>, <=status>)

<object index> object index of opened data file

<internal address> internal record address generated by FIND-RECORD or GET-RECORD-DESCRIPTION

<=status> return status

## 3.3.13. Access of partition system tables (directories etc.)

The file system is implemented as a hierarchy of separate modules (levels). The file system usually seen from user programs consists of the services available on the highest level (level 3). When the user executes a monitor call to level 3, this module is performing the appropriate function by monitor calls to the lower (more primitive) levels. The interaction between the different levels are illustrated in the figure below. The functions performed by the levels are roughly:

Level 0 performs physical device access. For some devices this level will contain buffering (disc. cache) and software to optimize the use of the hardware (minimize head movements on disc).

Level 1 is only used for the partition format. This level controls the allocation of storage space (maintains the bit table) and maintains the index hierarchy for all kinds of files (including directories and control files).

Level 2 is the name look-up manager. It is only used when objects are accessed by name (not object index) and when a change from one object to another is performed when more objects are opened simultaneously and concatenated (by * and/or + in the object name string). Level 2 is divided in part 2A and 2B. Level 2B searches for an entry in a directory on a partition corresponding to a single name (the bytes between two dots in a full name). Level 2A combines the single names to one or a set of full names. There is one level 2B process for every active partition, and one level 2A process for every active user program.

Level 3 separates access calls (READ-BLOCK/WRITE-BLOCK etc.) to different object types (partition files, channels, volumes files, terminal, etc.). New object types may be added to the system by adding an open routine and an access routine for the new object type on level 3. Level 3 contains all code and controlls all data necessary to structure partition files into records.

Directories may be opened as data files by specifying the directory name with type string 'DIRECTORY' as object name. Opening for read requires read access to the directory while opening for write is only allowed by privileged programs (system programs). Access method 0 should be specified to open the

queue until they are released by FREE-BATCH. Before such a hold command is given, the "wait for operator" flags will have no effect.

The "wait for operator" flag may be set on individual jobs by issuing a HOLD-BATCH command on a specific file. Again, only when a HOLD-BATCH with empty file name string has been issued, will the job really remain in the queue.

### 6.2.3. FREE-BATCH

<file name>,<=status>

This is the opposite to HOLD-BATCH. If the file name is an empty string, the "wait for operator" flags will no longer have any effect. If a particular file is specified, it will cause the "wait for operator" flag to be cleared on this job. Any suitable free process(es) will start processing the job(s) thus released.

### 6.2.4. SET-BATCH-LOG

<job in/out of queue> <job started> <job terminated>,<=status>

The parameters (yes or no) determine which events should be logged on the console terminal.

### 6.2.5. CREATE-BATCH-PROCESS

<program name> <=process no.>

A new batch process is created, and its process number is returned. The batch program <program name> can contain information on the job selection strategy applied by this particular process. This program may also prompt for installation dependent parameters.

### 6.2.6. DELETE-BATCH-PROCESS

<process no.>,<=status>

The batch process will be deleted when its current job (if any) is finished.

### 6.2.7. SET-BATCH-PROCESS-PRIORITY

<value> <process no.>,<=status>

Batch jobs with lower user-specified priority than <value> will not be run at all by the process. If <process no.> is zero, the hold criterion is defined. This criterion determines which jobs should automatically have their "wait for operator" flag set when they are appended. The <value> now usually signifies the biggest (in terms of processor time) job allowed to pass without the flag being set.

<=key length>  length of the key

<=bytes moved>  number of bytes moved

<=majority>  The current majority

<=status>  return code

3.3.12.10. FIND-RECORD

(<object index>, <key majority>, <key value>, <position>, <(=)rec address>, <=rec length>, <=status>)

<object index>  = 0 ==>the current key majority
> 0 ==> the key majority to be used. The majority is set current before the key is accessed

<key value>  a string containing the key value to be searched for

<position>  = 0 ==> position the file at the record with the greatest key less than or equal to <key value>
= 1 ==> position the file at the record with the smallest key greater than or equal to <key value>

<(=)rec address>  a read write parameter, containing an internal record address. If it is zero on input, it will contain the internal address of the data record found on output. If it is non zero on input, it will be assumed to contain an internal address generated by a previous instance of this monitor call, and the file will be positioned accordingly. The parameters <key value> and <position> will be used to check that the wanted record is still in the same position. If this is not true, <(=)rec address> will be ignored and the record is found according to key value.

<=rec length>  length of record

<=status>  the presence of the search key is indicated in this field.

3.3.12.11. DEFINE-RECORD

(<object index>, <position>, <record length>, <=status>)

<object index>  reference to open file returned from OPEN-OBJECT

<position>  = -1 ==> new record to be inserted before the current record
= 0 ==> update current record

it on a printer, thus achieving useful parallelism and user program independence of printer speed. Second, it maintains a spooling queue of all files waiting to be output, thus relieving users from having to wait for output devices to become free. Third, it contains facilities for print formatting, multiple copies, and flexible print routing.

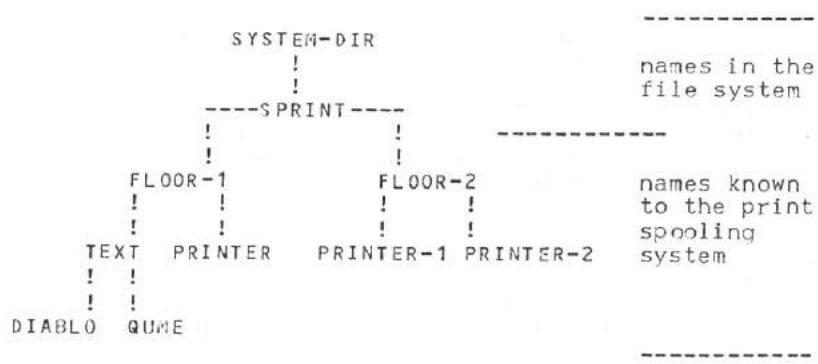
The simplest way to use the spooling system is to send output to the file named LINE-PRINTER (or other predefined names), either by copying an already existing file to it or by letting a program perform output to this special file directly. When the copying is finished, or when the LINE-PRINTER file is closed, the print will appear on an output device defined as LINE-PRINTER.

To avoid copying, an existing file may be put into the spooling queue directly by an append command.

## 7.1. Print devices

All print devices, i.e. devices connected to the print spooling system, are logically placed in a file directory. The name of this directory is installation dependent. The print device directory may contain several levels of directories with the bottom-level files representing the print devices.

Example: Let us assume that the print device directory is called SPRINT and that it resides in a directory named SYSTEM-DIR, i.e. its name is SYSTEM-DIR.SPRINT. Our system has three line printers, one on 1st floor and two on 2nd floor. There are also a Diablo and a Qume typewriter on 1st floor. We choose the names of the print devices to reflect this geographical structure:




The full name of printer 1 on 2nd floor is now:

SYSTEM-DIR.SPRINT.FLOOR-2.PRINTER-1

and of the Diablo:

SYSTEM-DIR.SPRINT.FLOOR-1.TEXT.DIABLO

without an intervening call to DEFINE-RECORD, the length of the record may not be altered.

If a record is to be inserted into a sequential fiel, a new current record must first be created by issuing a call to DEFINE-RECORD with <position> >< zero.

If a record is to be inserted into a keyed file, a new current record must first be created by issuing a call to DEFINE-KEY-RECORD.

If the record being written has been made current by a call to DEFINE-RECORD with <record length> > 0, processing may be faster, and an attempt to exceed the specified record length will not be actioned and an error will be generated.

### 3.3.12.5. SET-BLOCK-SIZE

(<object index>, <bytes pr. block>, <=status>)

This monitor call is used to set the block size of an opened file. The block size is used only for addressing of the file (together with the <block no.> parameter>, and has nothing to do with the file structure. It is assumed that the most common addressing unit within an unstructured file or record is one byte. Therefore the default block size will be one byte. The block size will not survive a CLOSE.

### 3.3.12.6. TRUNCATE-FILE

(<object index>, <block no.>, <=status>)

The file is truncated after the specified block. Only allowed for access method Direct on record file, and on unstructured files.

### 3.3.12.7. GET-RECORD-DESCRIPTION

(<object index>, <=rec. address>, <=block size>, <=block no.>, <=rec. length>, <=majority>, <=status>)

<object index> reference returned form OPEN-OBJECT

<=rec address> internal record address

<=block size> the number of bytes per block as set by SET-BLOCK-SIZE.

<=block no.> relative block address within current record.

<=rec length> length of record for record structured files, else length of file, in bytes.

<=majority> for keyed organization, this will contain the current majority.

<=status> return code.

## 7.4. Formats

When a file is output on a print device, it is printed according to a standard format pertaining to the device. To be able to alter the output format according to particular user needs, a format description may accompany files to be printed. Also, a format restriction may be set on a device, forcing files to comply with hardware device settings.

The user may specify such items as page size (logical, printed page size, not paper sheet size) and a message to be printed on the console terminal before the print is started. These wishes are normally met and no print delay caused. The user may further specify values regarding hardware printer settings. If these values do not comply with the current settings, as specified by the operator, the print will remain in the spooling queue until the proper changes are performed by the operator.

The spooling system will contain a number of standard formats which the user can call upon when needed. In addition to standard page sizes and printer settings, these may indicate special functions like "convert to upper case only", "drop spooling header and trailer", "apply device-dependent text-formatting", etc.

## 7.5. Public commands

### 7.5.1. Appending files

As we have seen, spooling files may be created and entered in the spooling queue by outputting to a print device. Already existing files may be appended to the queue by calling:

### 7.5.2. APPEND-PRINT

<file name>, <device name>, <no. of copies>, <priority>, <format no.>, <characters per line>, <lines per page>, <message>, <stop for each file>, <paper type>, <character set>,<=status>

The file <file name> will be appended to the spooling queue. <priority> determines the queue position, but the selection strategy applied by the spooling system needs not be based solely on this criterion. <no. of copies> copies of the file are subsequently output on <device name>, which may be a specification of a group of devices, as described earlier. If <format no.> is a positive integer, and a standard format carrying this number is already defined to the print spooling system (CREATE-FORMAT), the file will be printed according to this format. Format no. 0 is the default format for the device. Format no. -1 means that the following parameters define the format: The first two parameters define the page size. The <message> will be given on the console terminal when the file is ready for printing. If <stop for each file> is 'yes', the printer will stop before each print-out is started. The last two parameters should only be specified when the operator has given SET-FORMAT with corresponding values on the given device. Otherwise, the file will not be let through.

-4 ==>read from the current byte pointer. The byte pointer is updated by the number of bytes read. If the end of the record is encountered, the record pointer is positioned backward, and the byte pointer set to zero.

-5 ==> position the record pointer backward, set the byte pointer to zero and read the current record. The byte pointer is updated by the number of bytes read.

-6 ==> position file pointer backwards, set the record pointer to the last record in the file, set the byte pointer to zero and read the current record. The byte pointer is updated by the number of bytes read.

<memory buffer> memory address of where to place data read

<number of bytes> number of bytes to read

<=bytes moved> number of bytes actually transferred

<=status> return code

Reverse accessing is not permitted for files organized serially.

### 3.3.12.4. WRITE-BLOCK

(<object index>, <block no.>, <memory buffer>, <buffer length>, <=bytes moved>, <=status>)

<object index> reference from OPEN-OBJECT

<block no.> The block pointer is updated by this value before writing begins. It always points within the current record, and may take the following values:

>= 0 ==> the byte pointer is set <block number>*<block size> bytes from the start of the current record. The byte pointer gives the start of the destination area to which the data is then written.

= -1 ==> The current value of the byte pointer gives the start of the destination area to which the data is then written

= -2 ==> The byte pointer gives the start of the destination area to which the data then is written. After the data is written, position the record pointer forward, and set the byte pointer to zero.

= -3 ==> The data is written to the

### 7.6.4. START-PRINT

<file name>, <device name>,<=status>

A file previously stopped by calling STOP-PRINT or by specifying 'yes' at <stop for each file> (APPEND-PRINT), will be resumed.

### 7.6.5. BACKSPACE-PRINT

<file name>, <device name>, <no. of pages>,<=status>

The print will be backspaced a number of pages. If backspace past the start of the file is specified, the print is restarted completely, including any spooling header.

### 7.6.6. FORWARD-SPACE-PRINT

<file name>, <device name>, <no. of pages>,<=status>

A number of pages will be skipped.

### 7.6.7. PRINT-STATUS

<file name>,<=status>

The status of the file (position in the queue, amount left to print, format, etc.) will be returned.

### 7.6.8. PRINT-QUEUE

<device name>,<=status>

The print queue for the device will be listed. The device may specify a group of devices, as explained earlier. All files that might be output on any of the devices in the group are displayed. Unspecified device gives the whole spooling queue.

### 7.6.9. PRINT-SYSTEM-STATUS

An overall status of the print spooling system will be returned.

### 7.6.10. FORMATS

A list of the available standard formats will be returned.

## 7.7. Privileged commands

The following commands are available to operator privileged users only:

#### 3.3.11.15. RESERVE-PAGES

(<object name>, <respages>, <maxpages>, <=status>)

Change pages reserved and/or maximum pages for the file or directory specified by <object name>.

Add <respages> to pages reserved for <object name>. The pages reserved are added to pages used for the parent directory. <respages> may be negative. Add <maxpages> (may be negative) to maximum number of pages for <object name>. Write access to parent directory is required.

#### 3.3.11.16. TRANSFER-PAGES

(<from object name>, <to object name>, <number of pages>, <=status>)

Transfer <number of pages> (must be positive) from <from object> (file or directory) to <to object> in the same parent directory. Write access to <from object> is required.

#### 3.3.11.17. GET-PAGES

(<object name>, <=respages>, <=maxpages>, <=usedpages>, <=pagesize>, <=status>)

Returns pages reserved, maximum number of pages, and pages actually used for <object name>. The page size in bytes on the partition containing the object is also returned.

#### 3.3.11.18. GET-PAGE-SIZE

(<partition name>, <=pagesize>, <=status>)

Return the page size in bytes on specified partition.

### 3.3.12. Monitor calls acting on open files

#### 3.3.12.1. CLOSE

(<object index>, <special action>, <=status>)

The file(s) connected to <file no.> will be closed.

<file no.>=-1 has the special meaning:
close all files not permanently opened.

<file no.>=-2 has the special meaning:
close all files

<special action flag> This parameter may be used to override the normal actions taken by close-file according to the allocation attributes.

A new standard format is created. The message to appear on the console terminal can be used for describing the paper and hardware setting on the printer, if these are easily selectable and an operator is usually present.

### 7.7.7. DELETE-FORMAT

<format no.>,<=status>

The format is deleted.

### 7.7.8. SET-FORMAT

<format no.>, <device name>, <=status>

From now on, the device will be reserved for files of a particular format. Not all fields in format <format no.> need be specified; format compliance is only enforced for the fields specified. However, if format parameters regarding printer hardware settings are unspecified, the user should also leave these fields unspecified, since exact match is always required here (cf. sect. 5).

## 7.8. Print access

The print access checks performed are installation dependent. It is always true, though, that for privileged commands, caller access to any file(s) concerned is not checked. Public commands are normally restricted to the appending user and operator privileged users. The append command does always check the calling user's access to the file. For the PRINT-QUEUE and PRINT-SYSTEM-STATUS commands, file and user names are replaced by empty spaces whenever the caller does not possess access rights or operator privilege.

If several instances of the same file are present in the spooling queue, the commands specifying an action to be performed on a <file name> will affect all matching files. (For MOVE-PRINT, the first occurrence of <before file> is assumed.) To enable individual file access, and also to make life easier for the operator, each file is given a unique tag value consisting of four alphanumeric bytes. Whenever a file name is to be specified, a tag value may be substituted, preceded by the '#' tag indicator (ex.: #12AB). The tag is returned from the APPEND-PRINT command, and may also be looked up by calling PRINT-STATUS or PRINT-QUEUE.

If only an incomplete (not bottom-level) device name is specified for a command signifying an operation to be performed on a device (BACKSPACE-, STOP-, SUSPEND-, etc.), all matching devices will be affected, provided the operation is meaningful and the caller has access to any print concerned.

>5 ==> User defined structure

<organization> 1 ==> serial
2 ==>sequential
3 ==>relative sequential
4 ==> keyed sequential
5 ==> random

<record length> The length of fixed-length records: not used for variable length records, except for relative organization, in which case it indicates the maximum record size.

"Relative" organization is similar to sequential, but allows records to be accessed by an integer key - i.e. a key of 1 refers to the first record, a key of 2 refers to the second, etc.

When a relative file is created, all records are, in concept, set empty. This implies that it is not possible to insert a new record between two existing records, and therefore all writing is in fact a kind of rewriting. For this reason, sequential files, which do permit insertion, cannot be access relatively. Limited relative access can, however, be achieved using the POSITION-FILE monitor call.

Relative access is not permitted for files organized serially.

### 3.3.11.12. SET-DATA-PACKING

(<file name>, <frame packing>, <frames/group>, <group packing>, <bytes per frame>, <oflo>, <=status>)

Illegal on unstructured files.

<file name> the file name of the data file. It must be empty.

<frame packing> The frame packing density: the maximum part of the frame, expressed as a percentage, that may be used when the file is in write mode.

Default was 0.75.

<frames/group> The number of frames per group

<group packing> The maximum number of frames in a group which may be used when the file is in write mode.

Default was 75.

<bytes per frame> The number bytes per frame. This will be rounded up to the least upper bound which is an integral power of 2.

Default was 2048 (1 page).

### 8.1.4. SAVE-SYSTEM

<file name>

The system will be stopped as by STOP-SYSTEM, and all the physical memory will be copied to a contiguous file. The system can be restarted by an offline bootstrap program.

### 8.1.5. INITIATE-TERMINALS

When a system is started, only one terminal in active. This command will prepare the other connected terminals for time-sharing.

### 8.1.6. INITIATE-NETWORK

Connections to other systems are established.

### 8.1.7. SET-INITIAL-COMMAND

<macro call>

This is a macro call to be executed when the system is restarted.

### 8.1.8. SET-AVAILABLE

It is possible to log in on the terminals.

### 8.1.9. SET-UNAVAILABLE

<text>

It is not possible to log in on terminals other than the console terminal. Instead, the message given by <text> will appear on the terminal. When all users have logged out, a message is returned.

### 8.1.10. CONTROL-PROCESSOR

<processor name>

This subsystem can control the processor completely, starting and stopping it, and running test programs.

## 8.2. System modifications

The system can be modified by the subsystem
SINTRAN-SERVICE
The modifications include

I/O-device-handling
Processor and memory usage tuning

Default: The system will find a free area for the file.

<number of pages> size of file

<partition name> must be specified if the file is to be placed in another partition than the parent directory. Default: Same partition as parent directory

<device name> may be specified instead of <partition name>

### 3.3.11.5. CREATE-INDEXED-FILE

(<file name>, <initial number of pages>, <min. last page>, <max. page>, <partition name>, <device name>, <=status>)

Create a dynamically expandable file with <initial number of pages> pages initially. If possible these pages will be allocated contiguously and no page index will be created before it is necessary because the logical pages of the file are not physically contiguous.

Unwritten logical pages higher than <min. last page> will be deleted on CLOSE if the file has been opened for WA (rewritten).

<max. page> specifies the maximum allowed logical page number in the file.

<partition name> and/or <device name> must be specified if the file is to be placed on another partition than its parent directory.

### 3.3.11.6. SET-FILE-ATTRIBUTES

(<file name>, <allocation>, <special>, <structure>, <initial number of pages>, <min. last page>, <max. page>, <=status>)

<allocation> may have the values: Permanent (1), Scratch (2), and Temporary (3).

<special> may have the values: Normal (1), Reentrant (2), Delayed update (3), Shared versions (4), Ringbuffer (5), and multiversion (6). Either all or none of the file of the files in a subdirectory may have the special attribute Shared versions or Multiversion.

<structure> may have the values: Unstructured (1) and Record structured (2). If a file is set to be recordstructured, its organization must be further specified by the SET-FILE-STRUCTURE call (section 3.3.11.11)

<initial number of pages>, <min. last page>, and <max. page> are as described for for CREATE-INDEXED-FILE (section 3.3.11.5).

### 8.3.2.1. INIT-ACCOUNTING

<desired> <max.>

<desired> is the desired decimal number of accounts. After the file has reached this limit the warning APPROACHING END OF ACCOUNTING FILE is written out on every log off. Default value is 500. <max.> is the maximum decimal number of accounts permitted. After the file has reached this limit the warning END OF ACCOUNTING FILE ENCOUNTERED is written out on every log off. No accounting is done after this number is reached. Default value is 600.

### 8.3.2.2. START-ACCOUNTING

This command starts the accounting, but does not initiate the accounting file.

### 8.3.2.3. STOP-ACCOUNTING

This command stops the accounting system. The accounting files is not affected.

### .3.2.4. ACCOUNTS

This is a subsystem to print the results of the accounting.

When a "delayed update" file is closed all unmodified pages are transferred to the new file and the old (unmodified) version of the modified pages are deleted. This applies for all pages concerning the file, both index pages and data pages. In this way a consistent version of the file is always guaranteed if the system should go down during the file update. Either all or none of the modifications made between open and close/checkpoint will be effective.

### 3.3.9.6. Shared file versions

If the directory immediately above the data file level (lowest level) is defined to be a "file version" directory, the file versions may be defined as shared. This means that the different file versions may share common data and index pages. While a file in a shared file version directory is open it acts like a "reentrant" and "delayed update" file. When it is closed, or a checkpoint is taken, neither the old nor the new version of updated pages are deleted. The new version of the updated pages plus the unchanged pages are set as the first file version, and the other versions are (unchanged) shifted one version up. The pages unique to the highest version, or the first one having reached its maximum version number, are deleted.

In this way the first version of the file will always represent the file state at the last checkpoint, while the higher versions will represent earlier checkpoints.

Shared files may be defined by SET-DEFAULT-FILE-ATTRIBUTES before they are created, or by SET-FILE-ATTRIBUTES to the files in a file version directory. Checkpoint is taken by CLOSE or CHECK-POINT.

## 3.3.10. File access methods

A file may be accessed by three access methods: Direct access, unstructured access and record access. Note that the access method is not a file attribute. It defines how a file is to be accessed and is in principle independent of the file type. However, some file types does not allow all access methods.

Direct access:
By this access method the data pages of a file is read or written as they are stored, including any control information and free space (deleted records). By this access method the user may decode and organize the control information in his own way.

Unstructured access:
By this access method the file is seen from the user as a contiguous unstructured string of bytes. All control information will be hidden from the user, and the boundaries between records will be invisible.

For unstructured files, direct access and unstructured access will be equivalent.

Record access:

#### 3.3.8.2.3. Keyed organization

Files with keyed organization have the same properties as sequential files, and in addition, have a prime key associated with each record. The order of the records in the file will be the order of the prime key values. The prime key must be unique, and can therefore be used to access any one record in the file. The user may change between sequential and keyed access at will.

As many secondary keys as necessary may be specified. Such keys need not be unique. Access via these keys will supply the records in the order of the chosen secondary key's value. If there are several instances of a record to one key value, the records will be supplied in the order in which they were written - i.e. the oldest record will be supplied first.

Prime keys may be variable in length, in which case they must occur first in the record, but secondary keys must have a fixed length, and a fixed position in the record, relative to the end of the prime key. This restriction is necessary at present as most programming languages do not provide suitable constructions to deal with variable length structures.

Keyed organization is illegal on volume files.

#### 3.3.8.2.4. Relative organization

A nascent relatively organized file consists of a number of empty records, each the maximum number of bytes long. These records are accessed via an integer value - the value 1 will refer to the first record, the value 2 the second, and so on.

Relative organization is illegal on volume files.

### 3.3.9. Special file attributes

These attributes, which are mutually exclusive, define some special properties of files used for special purposes.

#### 3.3.9.1. Normal files

This is the normal file having no special property. This is the only special attribute allowed on volume files.

#### 3.3.9.2. Ringbuffer files

A ringnumber file works as a ringbuffer. When the file reaches its maximum size, the writing program may enter waiting state (depending on the timeout set). When data is read from the file, the space it occupied is released. When the file is empty the reading program may enter waiting state (depending on the timeout set).

difference is that an indexed file dynamically expandable because one or more index levels will be automatically generated if necessary.

Directories will always be indexed.

An indexed file may also be logically discontiguous. That means that pages never written into may not be physically allocated. An attempt to read data from nonallocated areas ("holes") of a file will give an error return.

True indexed files are not allowed on volumes. However, to allow backup of partition files with holes on volume files, a degenerated version of indexed files are allowed even on volumes. This is done by storing a logical byte number together with each physical page on the volume. An indexed volume file may only be written with ascending logical addresses, and read in physical sequence.

### 3.3.7. File allocation attributes

These attributes, which are mutually exclusive, decide the lifetime of a file name and its data.

#### 3.3.7.1. Permanent files

This is the "normal" file. Lifetime of both file name and data pages are controlled by user commands.

#### 3.3.7.2. Scratch files

During a terminal session (or batch job) a user may create scratch files. These files are deleted when the user is logging off.

Scratch files are illegal on volumes.

#### 3.3.7.3. Temporary files

A temporary file is a "read once" file. That is, all pages in the file are deleted when it is closed after an open for read (not read/write).

Temporary files must be indexed files and are illegal on volumes.

#### 3.3.7.4. Nameless files

If an empty string is specified between double quotes ("") in an OPEN-OBJECT command, a nameless file will be created in current environment. This file disappear as soon as it is closed.

Nameless files are illegal on volumes.

The system checks that USPAGES<=MAXPAGES is always true. Note that MAXPAGES (and thus USPAGES) may be greater than RESPAGES. However, the pages exceeding RESPAGES will not be exclusively reserved for this directory/file, so the directory/file will have to compete for this pages with other directories/files in the same parent directory and therefore these pages can not be guaranteed to be available (overbooking).

Initially, when an empty partition is created, all pages are given to the top level directory. That is, RESPAGES=MAXPAGES= number of physical pages on partition, and USPAGES= number of pages initially used by the system (for bit file etc.). Pages may then be reserved for lower level files/directories by RESERVE-PAGES (section 3.3.11.14) and transferred between lower level files/directories in the same parent directory by TRANSFER-PAGES (section 3.3.11.15).