

SINTRAN III

Utilities Manual

ND-60.151.01

NOTICE

The information in this document is subject to change without notice. Norsk Data A.S. assumes no responsibility for any errors that may appear in this document. Norsk Data A.S. assumes no responsibility for the use or reliability of its software on equipment that is not furnished or supported by Norsk Data A.S.

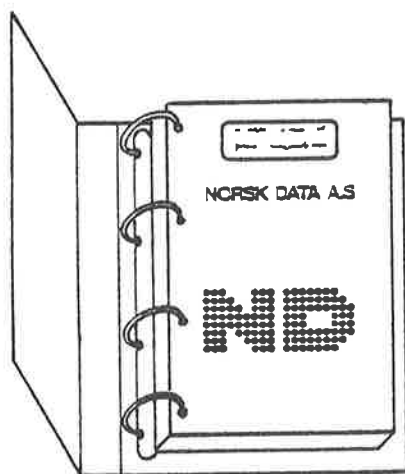
The information described in this document is protected by copyright. It may not be photocopied, reproduced or translated without the prior consent of Norsk Data A.S.

Copyright © 1982 by Norsk Data A.S.

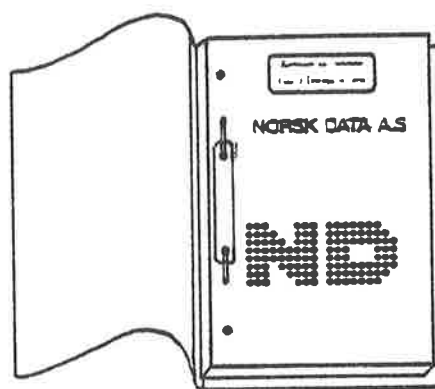
This manual is in loose leaf form for ease of updating. Old pages may be removed and new pages easily inserted if the manual is revised.

The loose leaf form also allows you to place the manual in a ring binder (A) for greater protection and convenience of use. Ring binders with 4 rings corresponding to the holes in the manual may be ordered in two widths, 30 mm and 40 mm. Use the order form below.

The manual may also be placed in a plastic cover (B). This cover is more suitable for manuals of less than 100 pages than for large manuals. Plastic covers may also be ordered below.



A Ring Binder



B Plastic Cover

Please send your order to the local ND office or (in Norway) to:

Documentation Department
Norsk Data A.S
P.O. Box 4, Lindeberg gård
Oslo 10

ORDER FORM

I would like to order

..... Ring Binders, 30 mm, at nkr 20,- per binder

..... Ring Binders, 40 mm, at nkr 25,- per binder

..... Plastic Covers at nkr 10,- per cover

Name

Company

Address

City

SINTRAN III Utilities Manual
Pbl. No. ND-60.151.01 Rev. B



Manuals can be updated in two ways, new versions and revisions. New versions consist of a complete new manual which replaces the old manual. New versions incorporate all revisions since the previous version. Revisions consist of one or more single pages to be merged into the manual by the user, each revised page being listed on the new printing record sent out with the revision. The old printing record should be replaced by the new one.

New versions and revisions are announced in the ND Bulletin and can be ordered as described below.

The reader's comments form at the back of this manual can be used both to report errors in the manual and to give an evaluation of the manual. Both detailed and general comments are welcome.

These forms, together with all types of inquiry and requests for documentation should be sent to the local ND office or (in Norway) to:

Documentation Department
Norsk Data A.S
P.O. Box 4, Lindeberg gård
Oslo 10

SINTRAN III/VS

INTRODUCTORY

ND-60.125
Sintr.III
Introduct.

USER'S
GUIDES

ND-60.134
Communic.
Guide

ND-60.132
Timeshar./
Batch Guide

ND-60.133
Real Time
Guide

REFERENCE

ND-60.151
Sintr.III
Utilities

ND-60.128
Sintr.III
Ref. Man.

ND-60.051
Real Time
Loader

OPERATOR/
SUPERVISOR

ND-30.001
NORD 10/50
Oper.Guide

ND-30.003
Sintr.III
Sys.Sup.Gu.

ND-60.110
Postmortem
Investegat.

ND-60.062
Sintr.III
Sys.Docum.

ND-60.122
File Sys.
Sys.Docum.

ND-60.072
RT-Loader
Sys.Docum.

ND-60.081
Nordnet
Sys.Docum.

ND-60.112
Sintr.III
Data Fields

INTERNAL SYSTEM DOCUMENTATION

SINTRAN III/RT

ND-60.082
Sin.III/RT
Ref. Man.

PREFACE

THE PRODUCTS

This manual describes products which run under the SINTRAN III operating systems

SINTRAN III/VS	ND—10048
SINTRAN III/VSE	ND—10174
SINTRAN III/VS-500	ND—10175

The products described are:

GPM	ND—10124
PERFORM	ND—10022
BACKUP—SYSTEM	ND—10337
LOOK—FILE	ND—10005
	ND—10044
FILE EXTRACT UTILITY	ND—10044

THE READER

This manual is written for users of SINTRAN III who want to use any of the subsystems listed above.

PREREQUISITES

Familiarity with SINTRAN III is an advantage.

THE MANUAL

This manual describes some utility subsystems of SINTRAN III.

RELATED MANUALS

Related manuals giving basic information about SINTRAN III are

SINTRAN III Introduction	ND—60.125
SINTRAN III Timesharing Batch Guide	ND—60.132

Other SINTRAN III manuals are shown on the preceding diagram.

TABLE OF CONTENTS

+ + +

<i>Section:</i>		<i>Page:</i>
1	INTRODUCTION	1—1
2	THE PERFORM SYSTEM	2—1
2.1	Introduction	2—1
2.2	PERFORM Commands	2—3
2.3	Example of a PERFORM Macro	2—4
2.4	Interactive Prompts During Macro Execution	2—5
2.5	Output Listing File Control	2—5
2.6	Mode File Submission Control	2—5
2.7	Extended Parameter Submission	2—6
2.8	Limitations, Restrictions and Defaults	2—6
2.9	Standard PERFORM Macros	2—7
3	GENERAL-PURPOSE MACRO GENERATOR — GPM	3—1
3.1	Introduction	3—1
3.2	GPM Syntax and Evaluation Rules	3—2
3.3	System Macros	3—3
3.4	Macro Evaluation	3—5
3.5	Conditional Macros	3—7
3.6	Recursive Macros	3—8
3.7	The GPM Library	3—9
3.8	GPM Under SINTRAN III	3—16
3.9	GPM Applications — Some Ideas	3—17
3.9.1	GPM and Semigraphic Display	3—17
3.9.2	System Generation Using GPM	3—18
3.10	Combined Use of PERFORM AND GPM	3—28
4	THE MAIL SYSTEM	4—1
4.1	Introduction	4—1
4.2	General Format	4—1
4.3	Subcommands	4—2
5	BACK-UP SYSTEM	5—1
5.1	Introduction	5—1
5.2	Simple Use of the BACKUP-SYSTEM	5—3
5.3	Commands	5—3
5.4	Commands — Detailed Description	5—7
5.5	Label Formats on Magnetic Tape VOLUMES	5—15
6	LOOK-FILE	6—1
6.1	Introduction	6—1
6.2	Commands — Summary	6—1
6.3	Commands — General Rules	6—2
6.4	Commands — Detailed Description	6—3

<i>Section:</i>	<i>Page:</i>
7	NORD FILE EXTRACT UTILITY COMMAND7—1
7.1	Introduction7—1
7.1.1	Purpose7—2
7.2	Command Structure7—3
7.2.1	Input File7—3
7.2.1.1	Mode File Save Option7—4
7.2.1.2	Limited Automatic Command Input.....7—4
7.2.1.3	Fixed Record Length Input File Option7—5
7.2.1.4	Indexed Access via KEY File7—5
7.2.2	Output File7—6
7.2.2.1	Output File Append Option7—6
7.2.2.2	File Split Option7—6
7.2.2.3	Output File Organization Change (X Option)7—7
7.2.3	Extract Selection specifications7—8
7.2.3.1	Numeric Field Evaluation7—9
7.2.3.2	Text Field Evaluation7—10
7.2.3.3	Text String Search7—11
7.2.3.4	Limited Text String Search7—12
7.2.3.5	Logical Operands7—13
7.2.3.6	Parentheses Nesting7—14
7.2.3.7	Input File Record Intervals7—15
7.2.3.8	Show First Input File Record Option7—16
7.2.3.9	Command Line Continuation Option7—16
7.2.4	Output Specifications7—17
7.2.4.1	Input Record Subsets Specification7—18
7.2.4.2	Output Record Constants7—19
7.2.4.3	Input Record Number Inclusion7—19
7.2.4.4	Output Record Number Inclusion7—20
7.2.4.5	Random Key Inclusion Option7—21
7.2.4.6	Terminal Output Wait Option7—22
7.2.4.7	Line Printer/Terminal Output Heading Option.....7—23
7.2.4.8	Line Printer/Terminal Page Numbering Option7—24
7.2.4.9	Predefined Heading as Extract Command Line ...7—25
7.2.4.10	Predefined Heading as Position Mask7—25
7.2.4.11	Split File Copy Option.....7—25
7.2.4.12	Show First Input File Record Option7—26
7.2.4.13	Command Line Continuation Option7—26
7.2.4.14	Skip Output Record Trailing Spaces7—26
7.3	Run-Time Status Messages7—27

1 INTRODUCTION

This manual collects together information previously published in the GPM manual ND-60.109 and SINTRAN III Reference Manual ND-60.128 and some new material.

The subsystems described are not necessary for simple use of SINTRAN III but may be of considerable use for particular tasks. The manual contains the information users need to efficiently use the particular subsystem relevant to their task.

2 THE PERFORM SYSTEM

2.1 INTRODUCTION

During the development of any software system, it is often necessary to re-compile the programs, load and test them. This repetitive work can be done more easily by using MODE files. However, MODE files have two limitations:

- 1) each MODE file requires a *separate* file
- 2) the parameters are *fixed* within the MODE file.

This program takes parameters from the command line and inserts them into a copy of the specified file and then runs it as a MODE file. This file contains "copies" of MODE file "images" that would have been on individual files.

Thus, the PERFORM program both saves file space and offers the user a great deal of flexibility as to the type of data that can be modified. It is NOT intended to replace the functions offered by the standard EDITORS.

PERFORM is called thus:

```
@PERFORM /macro file/, /macro name/, /optional paras/, P1, ... Pn
```

macro file = file :MCRO as described below
default: PERFORM-LIB:MCRO

macro name = 1-16 character macro identification.
default: FTN

optional paras = < output file.
default: TERMINAL.
a file name may be specified in quotes but the < *must* precede the first quote sign. Default type is :SYMB.

= > Mode file submission control para,
> C, = create don't run,
> R, = create run,
> Bn, = create submit to BATCH,
> default = > R

* MODE file name,

If a MODE file other than the default mode file MACRO'n:MODE is to be used, this file name may be given as an optional parameter preceded by a * (asterisk). The type of this MODE file will be :MODE.

P1,,, Pn = user macro replacement parameter(s). n = 1-20.

The macro file consists of a number of macro descriptions separated by B, E delimiters. The PERFORM program scans the macro file for the requested macro name, and if found creates a MODE file with the replacement or default parameters.

If a MODE file other than the default mode file MACRO'n:MODE is to be used, this file name may be given as an optional parameter preceded by a * (asterisk). The type of this MODE file will be :MODE.

P1,,, Pn = user macro replacement parameter(s). n = 1-20.

The macro file consists of a number of macro descriptions separated by B, E delimiters. The PERFORM program scans the macro file for the requested macro name, and if found creates a MODE file with the replacement or default parameters.

2.2 *PERFORM COMMANDS*

Macro is defined by the following simple syntax:

```

^B, macro name:
--- macro prompt and default commands ---
;
--- SINTRAN commands and data lines ---

```

^E,

Macro prompt and default commands:

^B, macro name; to indicate the beginning of a new macro definition.

^P,n, prompt string; to specify the prompts to be given where the parameter is not supplied.

^F,n, prompt string; as for ^P, however user response is assumed to be a SINTRAN mass storage file name and PEFORM will attempt expansion of abbreviated name.

^D,n,default string; to specify the default value to be used if no value given to a prompt (n).

^L, information string; string is displayed on Terminal.

^C, comment string; string is ignored.

^; to delimit the declarations from the MODE file data.

^E; to indicate the END of a macro definition.

macro name = a 1 to 16 character identifier.

n = a numeric value 1-20 signifying a positional parameter expected on the @PERFORM command.

The ^ sign shown above can be any character other than A-Z, 0-9 or blank. The up arrow (^) is recommended as this follows the NOTIS and GPM syntax. The ^ character is taken from the *first* character on the :MCRO file, and must be the same character throughout the file.

Macro parameter submission placement is indicated by the use of the backlash (\) followed by n (1-20).

The semicolon (;) is required to terminate macro commands.

2.3

EXAMPLE OF USING A PERFORM MACRO

For example, to compile, load and dump a FORTRAN program, type the following command:

```
@PERFORM FTN FTNDUMP ABC 30000
```

Where:

FTNDUMP is the macro name that will be searched for on the file FTN:MCRO.
ABC is the FTN source program (ND editor file).
30000 is the desired UPPER LIMIT parameter for the loader.

FTN:MCRO (note the file type) is a file created by an ND editor as follows:

```
^B, PREVMACRO;
^----- etc. -----          other parameters as required
^;
^E;
^B, FTNDUMP;                   Begin macro definition
^L, Macro to compile and dump a FTN
  program;
^P, 1, source file name?;
^P, 2, Upper limit?;
^D, 2, 177777;
^;                               end parameter specifications
@DELETE-FILE \1:PROG
@FTN
COM \1,,TEMP
EX
@NRL
UPPER-LIMIT \2
L TEMP
DUMP ''\1''
EX
^E;                               End macro definition
^B, NEXTMACRO;
^----- etc. -----          other macro definitions (no limit)
^;
^E;
```

Example of use of default upper limit:

```
@PERFORM FTN FTNDUMP ABC,,
```

Resulting MACROn:MODE file:

```
@DELETE-FILE ABC:PROG
@FTN
COM ABC,,TEMP
EX
@NRL
UPPER-LIMIT 177777
L TEMP
DUMP ''ABC''
EX
```


2.4 *INTERACTIVE PROMPTS DURING MACRO EXECUTION*

PERFORM may be called by @PERFORM. Often the user wishes to know if the PERFORM-LIB:MCRO file being used actually contains the particular macro requested. Therefore, under "AUTO PROMPT" mode the user may respond to the "macro name:" prompt with a "?" character. PERFORM will then list all the macros defined on the currently attached :MCRO file.

Example:

```
:MCRO file name  : FREDs-MACROS
Macro name      : ?
```

Macros available in file FREDs-MACROS

```
FTN
COBRUN
Macro name      : COBRUN
```

2.5 *OUTPUT LISTING FILE CONTROL*

By default, the output from the execution of the mode file goes to the terminal, in the same way as in the @MODE command. Optionally, it may be sent to a file by specifying the file name, preceded by "less than" (<). This must appear after the macro name and before any parameters for the procedure.

Example:

```
@PERFORM FTN FTNDUMP <SINK ABC,,
```

2.6 *MODE FILE SUBMISSION CONTROL*

The @PERFORM user may direct the usage of the created :MODE file by using the optional "greater than" (>) directive:

```
> CREATE, = create MODE file but do not execute.
> RUN, = create MODE file and execute.
> BATCHn, = create MODE file and submit to BATCH number n (n = 1-9).
```

These directives may be abbreviated as only the first character after the > is checked, also the last in the case of the > BATCHn directive.

Default is > RUN.

PERFORM writes the actual mode file to be executed to a file called MACROi:MODE, where i is 1 to 9. The file is created if not already in existence. If the file is in use at another terminal or in a batch job, another file with a greater value of i is used automatically. It will be created if necessary. Note that if a > CREATE mode file is built it is the user's responsibility to make sure that later calls to @PERFORM does not destroy that file.

2.7 *EXTENDED PARAMETER SUBMISSION*

Any parameter (Pn) may be replaced by a file name, preceded by a square bracketed ([). The file should be a normal QED or PED file and should contain a list of values for the parameter, one per line. The procedure will then be executed repeatedly, taking successive values for the parameter from the file, if the file LIST contains:

```
ABC
DEF
GHI
```

and one gives the command

```
@PERFORM FTN FTNDUMP [ LIST, ,
```

the files ABC, DEF and GHI will be compiled in turn.

2.8 *LIMITATIONS, RESTRICTIONS AND DEFAULTS*

The macro name must be unique, in any case the first occurrence is taken.

The macro name should not be abbreviated, but if abbreviated that abbreviation will be searched for.

The macro cannot be nested, nor invoke other macros.

The first two parameters must be present either as actual parameters or empty parameters separated by commas.

The optional parameters (< and >) may also be entered if the macro name is being prompted for by PERFORM.

Use the F command rather than the P command if SINTRAN files are to be created by your Macro. The F command will attempt to find the *full SINTRAN file name*. If successful that name will be submitted to your macro. Default type is SYMB.

2.9 *STANDARD PERFORM MACRO'S*

The following macro's are supplied as standard with SINTRAN III operating system:

Name	Function
FTN	compile a Fortran program.
FTNRUN	compile, load and execute a Fortran program.
COBOL	compile a Cobol program.
COBRUN	compile, load and execute a Cobol program.
COBDEBUG	compile, load and execute a Cobol program under control of the Symbolic Debugger.
PLANC	compile a Planc program.
PLRUN	compile, load and execute a Planc program.
PASCAL	compile a PASCAL program.
PASRUN	compile, load and execute a Pascal program.
FTNRUN	compile, load and execute a Fortran program.
BASIC	compile a Basic program.
BASRUN	compile, load and execute a Basic program.
CREDIR	create a directory and a user on a formatted diskette.

3. GENERAL-PURPOSE MACRO GENERATOR - GPM

3.1 *INTRODUCTION*

In the Computer Journal, October 1965, C. Strachey described a macrogenerator called GPM (General-Purpose Macrogenerator). GPM was originally planned to help write a compiler for the language CPL. The idea was to write the whole compiler as a set of macro calls.

In this way, one got a machine-independent compiler. By redefining the macros, a compiler for another machine could be produced, and by rewriting GPM, one could generate the compiler on another machine other than the target machine.

GPM is referenced in most of the literature dealing with macro-processors.

Input to GPM is a character string, in which macro calls may occur. GPM copies the input character unmodified to the output string, with the exception of the macro-calls which yield their values instead.

GPM pays no attention to what type of symbolic input it receives, as long as no confusion arises concerning the GPM control characters. The GPM-version on the NORD computers expects (and produces) characters with even parity. It may be called as a SINTRAN III subsystem. Program size is 1,5k, while the rest of the virtual memory is used for run-time stack.

Most persons reading this manual for the first time know macros only from simple assembler macro-options. They should immediately be aware of the fact that in GPM macro-calls may not only occur in the source-code string, but also in a macro-call's name-string, parameter-strings and in the value-strings found in the macro definition-list. They should also keep in mind that the effect of a macro call may be of two kinds:

- 1) Substitution. A character string is substituted for the call.
- 2) Macro-(re)definition. New macros may be defined and old ones redefined.

3.2 GPM SYNTAX AND EVALUATION RULES

A GPM macro call looks like this:

↑ NAME, PAR1, PAR2, -----, PARn;

It consists of a macro name and a list of the actual parameters, each separated by a comma. The macro-call starts with ↑ and ends with a semicolon. The name- and parameter- strings may themselves contain macro calls.

Six characters have special meaning in GPM:

- ↑ Precedes macro calls
- ;
- ,
- \ Denotes formal parameter, and is followed by the parameter number in the set 0-9, A-Z. Occurs in macro definitions and the resulting macro bodies
- < Start quote. Should always match a >. Evaluation of a character-string enclosed in < > yields the same string without < >. Thus, by quoting, strings are prevented from being changed by GPM- evaluation
- > End quote. (An unmatched > outside macro calls terminates GPM)

The input string is scanned from left to right and copied to the output string until a macro call is encountered. The macro call is evaluated as follows:

- a) The macro name and its arguments are evaluated from left to right. They are all evaluated *once*. This process may involve evaluation of other macro calls so that the whole process of evaluating is a recursive one. Macro-definitions made during this process are so-called *temporary* definitions.
- b) When the argument list is complete (: when the name- and parameter strings have been evaluated) the macro-definition list is searched for a match with the evaluated name-string. The scanning stops with the first entry with the correct name, so that the most recent definition is used.
- c) The string corresponding to the macro name (macro's value, "body") is now scanned in the same way as the original input string, except that occurrences of \1, \2 --- etc. are replaced by exact copies of the corresponding actual parameter (the corresponding evaluated parameter-string). \ 0 means the macro name. If an argument asked for is not supplied, the string NIL is taken as actual parameter.
- d) On reaching the end of the defining string, the argument list (macro name- and actual arguments) are lost. Any macro-definitions added to the definition- list in course of macro name- and parameter evaluation are lost (temporary definitions).
- e) Scanning of the input string is resumed.

3.3 SYSTEM MACROS

GPM contains a number of system macros. These are, in reality, calls of system procedures, but the syntax of these calls is the same as that of the macro-calls and so are the evaluation rules. The system macros are:

DEF defines user's macros. It takes two arguments: The name and the value ("body") of the new macro. Formal parameters occurring in the "body" must always be quoted. The latest definition of a macro is the valid one.
 Format: \uparrow DEF, macro-name, macro-body;
 Example:
 \uparrow DEF, A, <B\1>; defines macro A to have B\1 (B and the first parameter) as its value. For instance, \uparrow A,5; yields the value B5.

Consider the definition of A in the following two examples:

- 1) \uparrow DEF, B, C; \uparrow DEF, A, \uparrow B;; \uparrow DEF, B, D; \uparrow A;
- 2) \uparrow DEF, B, C; \uparrow DEF, A, < \uparrow B>; \uparrow DEF, B, D; \uparrow A;

Each example consists of three definitions and a call of macro A. What is the result in these two cases? The only difference between 1) and 2) is the quotes in the definition of A.

- 1) defines A equal to the value of B, which is C.
 Hence: \uparrow A; yields C.
- 2) defines A equal to \uparrow B;. \uparrow A; is therefore equivalent to \uparrow B; which yields D. (Latest definition of B is valid!)
 Hence: \uparrow A; yields D.

Definitions made during parameter-evaluation are *temporary* definitions. These definitions are lost when the macro possessing the parameters has been evaluated. Earlier definitions of the same macros will then be reinstated.

Example:
 \uparrow DEF, A, B; \uparrow A, \uparrow DEF, A, C;; \uparrow A;

Temporary definition.

This string yields CB. Explanation:
 \uparrow DEF,A,B; defines A to have value B.
 \uparrow A, \uparrow DEF, A, C;; calls macro A, defining A temporarily to have value C. The call of A, therefore yields C, and the temporary definition is lost.
 \uparrow A; therefore yields B since the old definition has been reinstated.

VAL gives the value ("body") of the macro given as parameter. By means of VAL, macro-definitions may be inspected.
 Format: \uparrow VAL, macro-name;
 Example:
 Suppose macro A has been defined by \uparrow DEF,A, <B\1>;
 Then \uparrow VAL,A; yields B\1.

UPDATE	<p>updates macro definitions. Works in the same way as DEF. The new value must not be longer than the old value.</p> <p>Format: ↑UPDATE, macro-name, macro-body;</p> <p>Example:</p> <p>Suppose A has been defined equal to B\1.</p> <p>The call ↑UPDATE,A,<C\1>;</p> <p>defines A equal to C\1.</p>
BAR	<p>performs binary arithmetic. Takes three arguments. The first must be +, -, *, / or R, which means add, subtract, multiply, divide and remainder, respectively. The second and third arguments are two binary numbers.</p> <p>Format: ↑BAR, operator, binary number, binary number;</p>
DECBIN	<p>performs decimal-to-binary conversion.</p> <p>Format: ↑DECBIN, decimal number;</p>
BINDEC	<p>performs binary-to-decimal conversion.</p> <p>Format: ↑BINDEC, binary number;</p> <p>Example:</p> <p>↑DEF, SUM, <↑BINDEC, ↑BAR, +, ↑DECBIN, \1; , ↑DECBIN, \2; ; >;</p> <p>defines a macro SUM which yields the decimal sum of its two parameters. For instance, ↑SUM,5,3; yields 8.</p>
OCTBIN	<p>performs octal-to-binary conversion.</p> <p>Format: ↑OCTBIN, octal number;</p> <p>Example: ↑DEF, CTR, <↑BAR, -, \1, ↑OCTBIN, 100; ; >;</p> <p>defines a macro that yields control characters.</p> <p>For instance, ↑CTR,A; yields A^c.</p>
BINOCT	<p>performs binary-to-octal conversion.</p> <p>Format: ↑BINOCT, binary number;</p>
HD	<p>gives the first character of its argument ("head").</p> <p>Format: ↑HD, string;</p> <p>Example: ↑HD, ABC; yields A.</p>
TL	<p>gives all but the first character of its argument ("tail").</p> <p>Format: ↑TL, string;</p> <p>Example: ↑TL, ABC; yields BC.</p>

In the present GPM version, two additional system-macros have been made:

ICRMOD	<p>which makes GPM ignore the characters "carriage return" and "line feed" in its <i>input</i> string. They may, however be used internally and be output.</p>
CRMOD	<p>turns off the mode set by ICRMOD.</p>

3.4 MACRO EVALUATION

According to rules a-e in Section 3.2, GPM works as follows:

Initially GPM is in *copying mode*.

When a macro-call $\uparrow N, P_1, P_2, \dots, P_K;$ is encountered, GPM enters the *parameter-evaluation mode*.

The string N is evaluated to p_0 .

The string P_1 is evaluated to p_1 .

The string P_2 is evaluated to p_2 .

⋮

The string P_K is evaluated to p_R .

GPM now searches for the latest definition of p_0 in its macro-definition list. When found, GPM enters the *macro-expansion mode* (or the *macro-definition mode*, if p_0 is equal to DEF or UPDATE). GPM now reads and evaluates the macro body of p_0 . When encountering a formal parameter marker $\backslash m$, GPM enters the *parameter-substitution mode* and replaces $\backslash m$ by p_m . The resulting string (the evaluated body with the actual parameters substituted for the formal ones) replaces the call $\uparrow N, P_1, P_2, \dots, P_K;$ in the output string.

The macro-evaluation procedure is illustrated by this example:

Suppose the following macros are defined.

```

↑DEF, $, <ENE\1>;
↑DEF, ' ' DIRTY_DICK' ', 1;
↑DEF, #, <\2<LIC_>↑$, \1; _\0\3>;

```

We want to find the value of:

```

↑#, MY, <PUB>, ↑' ' DIRTY_DICK' ' ; ;

```

We start to evaluate the name and parameters.

evaluates to # which is the macro-name.

MY evaluates to MY which is the parameter no. 1

<PUB> evaluates to PUB which is the parameter no. 2

↑' ' DIRTY_DICK' ' ; evaluates to 1 which is the parameter no. 3

The latest definition of # is $\backslash 2 < LIC_ > \uparrow \$, \backslash 1; _ \backslash 0 \backslash 3$

$\backslash 2$ evaluates to PUB

$< LIC_ >$ evaluates to LIC_

$\uparrow \$, \backslash 1;$ is equivalent to $\uparrow \$, MY;$ which evaluates to ENEMY

_ evaluates to _

$\backslash 0$ evaluates to the evaluated macro-name #

$\backslash 3$ evaluates to 1

So the value of our macro-call is the string

PUBLIC ENEMY #1

A further example:

A well-known GPM example is the successor macro. When called with a number 0-9 it gives the next number. For instance, $\uparrow\text{SUC},3;\rightarrow 4$ $\uparrow\text{SUC},4;\rightarrow 5$ etc. Of course this can be achieved in arithmetical ways, but the SUC-macro accomplishes it in a way that makes it theoretically interesting.

SUC is defined as follows:

$\uparrow\text{DEF}, \text{SUC}, <\uparrow 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \uparrow\text{DEF}, 1, <\backslash>\backslash 1; ;>;$

We see that a call of SUC is equivalent to a call of a macro whose name is 1. The macro 1 is called with its first parameter=2, the second parameter=3, the third parameter=4, etc. A temporary definition of 1 defines it to have a value equal to one of its actual parameters. The parameter-number is equal to the actual parameter of SUC. Therefore, a call $\uparrow\text{SUC},3;$ defines macro 1 to be equal to its third actual parameter which is 4. Macro 1 is called, and yields 4 which is also the value of $\uparrow\text{SUC},3;$

3.5 *CONDITIONAL MACROS*

This chapter and the next one which deals with recursive macros, will describe the rather complicated methods used for defining such macros. They may be bypassed by readers who are not especially interested.

The definition of a conditional macro is given below:

```
↑DEF, COND, <↑\1, ↑DEF, \1, C; ↑DEF, A, B; ;>;
```

The macro COND gives B or C, depending on its argument. The only argument that gives B, is A, i.e.

```
↑COND, A;           yields B
↑COND, anything else; yields C
```

Explanation:

Suppose COND is called with argument=A. The macro-body with argument=A inserted, will look like ↑A,↑DEF,A,C;↑DEF,A,B;; This is a call of macro A which is defined twice in its own argument. (These are *temporary* definitions.) Since these definitions are made before searching the definition-list for the value of macro A, this works perfectly well. Since the last definition of A defines it equal to B, the call of A yields B which is also the value of COND. Therefore:

```
↑COND, A; →B.
```

Suppose COND is called with argument=X. The macro-body with argument X inserted, gives:

```
↑X, ↑DEF, X, C; ↑DEF, A, B; ;
```

This shows a call of macro X, which is defined once in its own parameter. The value is C, which is also the value of COND. Therefore:

```
↑COND, X; →C.
```

Note that the temporary definitions cannot be confused with any other definitions of X or A since the temporary definitions will be lost when COND has been evaluated.

Proper understanding of this conditional macro is necessary in order to understand how recursive macros with finite call-sequences work.

3.6

RECURSIVE MACROS

$\uparrow \text{DEF}, A, \langle B \uparrow A; \rangle;$

This is the simplest example of a recursive macro. One call of A yields an infinite stream of B-characters. (The evaluation will of course cease when GPM runs short of stack-space).

More interesting, however, are the recursive macros that allow a finite number of recursive calls. Before discussing them, we take a short review of the conditional macro COND, discussed in Chapter 4.

$\uparrow \text{DEF}, \text{COND}, \langle \uparrow \backslash 1, \uparrow \text{DEF}, \backslash 1, C; \uparrow \text{DEF}, A, B; \rangle;$

Covers the "general case" Covers the "special case"

Tells whether
"general case" or "special case"

Suppose we want to write a recursive macro with finite call-sequence. There must obviously be some kind of "condition" involved, in order to stop the recursive evaluation.

The "general case" results in an operation between a value and a recursive call, while the "special case" involves no recursive call since we now want to stop. What tells us the current "case"? Usually a counter, since we often want to give the number of recursive calls.

A recursive macro RECUR may, therefore, have a structure like this:

$\uparrow \text{DEF}, \text{RECUR},$
 $\langle \uparrow \text{counter}, \uparrow \text{DEF}, \text{counter}, \langle \text{value } X \text{ op } \uparrow \text{RECUR}, \text{counter}-1; \rangle; \uparrow \text{DEF}, 0, \text{value } Y; \rangle;$

"Current case" "General case" "Special case"

Where op denotes any operation wanted.

Suppose we want to construct a recursive macro FAC which computes the n'th factorial.

$\uparrow \text{FAC}, n; \rightarrow \text{The value of } 1.2.3. \dots n = n!$

Suppose that macros computing products and differences have been defined earlier and that their names are PROD and DIF. (For instance: $\uparrow \text{PROD}, 2, 3; \rightarrow 6$ and $\uparrow \text{DIF}, 8, 3; \rightarrow 5$).

We first concentrate on the "general case".

We observe that $n! = n.(n-1)!$

or, in macro language, where n is the 1st parameter of FAC:

$\uparrow \text{PROD}, \backslash 1, \uparrow \text{FAC}, \uparrow \text{DIF}, \backslash 1, 1; ; ;$

This leads us to the temporary definition that covers the "general case":

$\uparrow \text{DEF}, \backslash 1, \langle \uparrow \text{PROD}, > \backslash 1 <, \uparrow \text{FAC}, \uparrow \text{DIF}, > \backslash 1 <, 1; ; ; \rangle;$

Note that the 1st parameter must be "unquoted" since it is a parameter of FAC, not of the counter.

The "special case" is very simple.

Since $\uparrow \text{FAC}, 0; \rightarrow 0! = 1$ the temporary definition that covers the special case simply is $\uparrow \text{DEF}, 0, 1;$

Now we may write the complete definition of FAC:

$\uparrow \text{DEF}, \text{FAC}, < \uparrow \backslash 1, \uparrow \text{DEF}, \backslash 1, < \uparrow \text{PROD}, > \backslash 1 <, \uparrow \text{FAC}, \uparrow \text{DIF}, > \backslash 1 <, 1; ; >; \uparrow \text{DEF}, 0, 1; ; >;$

$\underbrace{\hspace{10em}}$
 "General case"
 expressing that $n! = n \cdot (n-1)!$

$\underbrace{\hspace{10em}}$
 "Special case"
 expressing that $0! = 1$

n

Here is another example which is important, since it allows us to generalize the "recursive call" property.

We want to make a recursive macro DO so that $\uparrow \text{DO}, A, n;$ is equivalent to n calls of the parameterless macro A.

DO may be defined as follows:

$\uparrow \text{DEF}, \text{DO}, < \uparrow \backslash 2, \uparrow \text{DEF}, \backslash 2 < \uparrow \backslash 1 <; \uparrow \text{DO}, > \backslash 1 <, \uparrow \text{DIF}, > \backslash 2 <, 1; ; >; \uparrow \text{DEF}, 1, < >; ; >;$

$\uparrow \text{DO}, A, 5;$ gives the same value as $\uparrow A; \uparrow A; \uparrow A; \uparrow A; \uparrow A;$

That a macro is parameterless does not necessarily mean that its value is constant, since it may call and redefine other macros.

3.7

THE GPM LIBRARY

This GPM library consists mainly of definitions of macros performing arithmetical or logical functions. It also contains generalized, recursive macros and conditional macros. The arithmetical functions may either be *decimal* or *octal*. When necessary to distinguish between them, the macro-name for the octal operation begins with &.

Example:

The macro SUM yields the decimal sum of its two parameters, while &SUM yields the octal sum. The arithmetical macros may further be divided into two classes, the "verbs" and the "nouns". A "verb" has only side-effects. That means it affects the macro-definitions, but leaves no value. A "noun" has no side-effect but yields a value.

Examples:

ADD is a "verb", SUM is a "noun".

$\uparrow \text{ADD}, J, 3;$ adds 3 to the value of "macro J" (which is updated) but the ADD-macro leaves no value. $\uparrow \text{SUM}, 3, 5;$ yields 8 as its value but it has no side-effects.

If you are unfamiliar with macro languages, please keep the following in mind:

The effect of a macro-call may be of two kinds:

- 1) *Substitution.*
A character string (which may be empty) is substituted for the macro call.
- 2) *Definition*
Macros may be defined or redefined. Nothing is substituted due to definition alone.

Both kinds of effects may arise from one macro-call.

↑VARIABLE, name, initial value (optional);

Six digits are allocated (for the value) and the variable is updated to its initial value (to 0 if no value specified).

Example:

↑VARIABLE, PER; ↑PER; 6→0

↑VARIABLE, OLA, 14; ↑OLA; →14

Since six digits are allocated, octal or decimal integer values may be assigned to a variable by an UPDATE-call.

↑INCREMENT, variable;

Increments the specified variable and is equivalent to ↑ADD, variable, 1;

Example:

↑VARIABLE, PER, 5;

↑PER; →5

↑INCREMENT, PER;

↑PER; →6

↑&INCREMENT, variable;

Octal increment of the specified variable and is equivalent to ↑&ADD, variable, 1;

↑DECREMENT, variable;

Decimal decrement of the variable.

Equivalent to ↑SUB, variable, 1;

↑&DECREMENT, variable;

Octal decrement of the variable.

Equivalent to ↑&SUB, variable, 1;

↑ADD, variable, number;

Decimal addition. Adds the number to the variable, but yields no value.

↑&ADD, variable, number;

Octal addition.

↑SUB, variable, number;

Decimal subtraction.

↑&SUB, variable, number;

Octal subtraction.

↑MPY, variable, number;

Decimal multiplication.

↑&MPY, variable, number;

Octal multiplication.

- ↑DIV, variable, number;
Decimal division.
- ↑&DIV, variable, number;
Octal division.
- ↑SUM, number, number;
Yields the decimal sum of the two numbers.
- ↑&SUM, number, number;
Yields the octal sum of the two numbers.
- ↑DIFFERENCE, number, number;
Yields the decimal difference between the two numbers.
- ↑&DIFFERENCE, number, number;
Yields the octal difference between the two numbers.
- ↑PRODUCT, number, number;
Yields the decimal product of the two numbers.
- ↑&PRODUCT, number, number;
Yields the octal product of the two numbers.
- ↑QUOTIENT, number, number;
Yields the decimal quotient of the two numbers.
- ↑"IENT, number, number;
Yields the octal quotient of the two numbers.
- ↑REMAINDER, number, number;
Yields the decimal remainder of the two numbers (concerning division).
- ↑&REMAINDER, number, number;
Yields the octal remainder.
- ↑POWER, number, exponent;
Yields a^n where a is the first parameter and n the second. $n \geq 0$.
- ↑SIGN, number;
Yields the sign (+ or —) of the decimal number.
- ↑&SIGN, number;
Yields the sign (+ or —) of the octal number.
- ↑DEC, number;
Converts from octal to decimal number.
- ↑OCT, number;
Converts from decimal to octal number.
- ↑CTR, letter;
Yields the corresponding control-character.
(↑CTR, A; → A^o).

- ↑CHARACTER, octal, number;
Yields the corresponding character.
Example:
↑CHARACTER, 76; → >
- ↑ESC;
Yields an escape-character (33_a).
- ↑CRLF;
Yields "carriage return"/"line-feed".
- ↑EQUAL, String 1, String 2, String 3, String 4;
If String 1 is equal to String 2, the result is String 3. If unequal, the result is String 4.
- ↑LESS-THAN, Number 1, Number 2, String 1, String 2;
If Number 1 is less than Number 2, the result is String 1. If not, the result is String 2.
- ↑&LESS-THAN, Number 1, Number 2, String 1, String 2;
LESS-THAN macro for octal numbers.
- ↑OR, String 1, String 2, String 3, String 4;
If 1st or 2nd parameter or both are non-empty, the value will be the 3rd parameter. Else the 4th parameter.
- ↑AND, String 1, String 2, String 3, String 4;
If both 1st and 2nd parameter are non-empty, the value will be the 3rd parameter. Else the 4th parameter.
- ↑XOR, String 1, String 2, String 3, String 4;
If 1st or 2nd parameter, but not both, is non-empty, the value will be the 3rd parameter. Else the 4th parameter.
- ↑NUMCH, String;
Yields the decimal number of characters in the string. The string should contain no GPM control characters.
- ↑ERRAB, cause;
Yields the following:
@CC _ *** SYSTEM_GENERATION_ ABORTED ***
@CC_CAUSE: _Cause
Esc Esc
- ↑%, comment;
Yields nothing. May be used for comments.
- ↑MAKE2, number;
Yields the number by giving at least two digits.
Example:
↑MAKE2, 5; yields 05
- ↑BITMASK, number;
Yields the bitmask corresponding to the decimal bitnumber [0-15].
(Example: ↑BITMASK, 8; → 400)

↑MASK, length, bitnumber;

Yields the bitmask. The length is given by the first decimal parameter, and the rightmost bit is given by the second, decimal parameter [0-15].

(Example: ↑MASK, 2, 1; →6)

↑LSHIFT, octal number, octal number of shifts;

Yields an octal number which is the first parameter left-shifted the number of times given by the second parameter. The number of shifts must be in the interval [0-17₈].

↑RSHIFT, octal number, octal number of shifts;

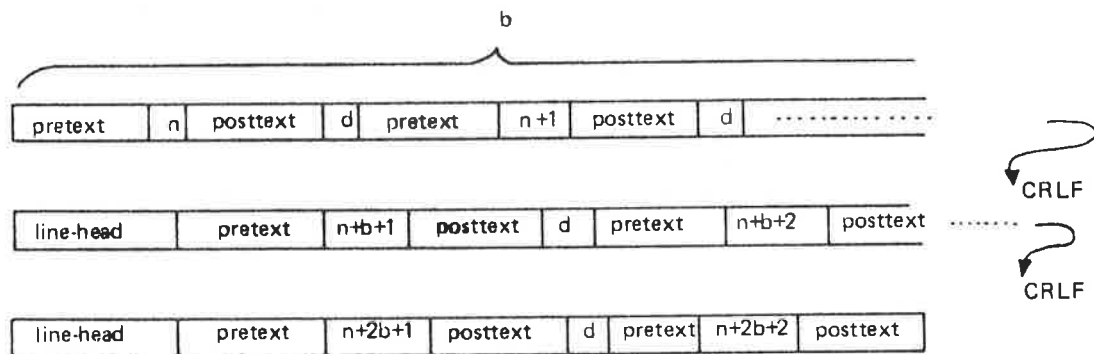
Yields the octal number right-shifted with sign extension.

↑RZSHIFT, octal number, octal number of shifts;

Yields the octal number right-shifted with zero end-input.

↑SEQUENCE, pretext, posttext, number of el., block-size, start-no., delim., line-head;

This macro gives a sequence of the following form:



where `n` is start-no., `b` is block-size and `d` is delimiter.

Example:

```
*)9EXT_↑SEQUENCE, RT, P, 11, 4, 2, _ , *)9EXT_ ;
yields
```

```
*)9EXT_ RT2P_ RT3P_ RT4P_ RT5P
*)9EXT_ RT6P_ RT7P_ RT8P_ RT9P
*)9EXT_ RT10P_ RT11P_ RT12P
```

Another example:

```
INTEGER ARRAY:=(↑SEQUENCE, A, , 7, 3, 0, <<<, >>>, ↑CTR, I; ; );
yields
INTEGER ARRAY ARR: =(A0, A1, A2,
    A3, A4, A5,
    A6);
```

Note that the comma must be triple-quoted in the macro call.

↑DO, macro-name, number;

This macro results in a number of calls of the *parameterless* macro given by the first parameter.

The number of calls is given by the second, decimal parameter which must be ≥ 0 .

(Example: ↑DO,A,3; is equivalent to ↑A;↑A;↑A;)

↑DO-LOOP, variable, start-value, step-length, limit, <body>;

This macro temporarily defines a parameterless macro which has *body* plus the proper updating of *variable* as its value. The macro is called the specified number of times. Default step-length is 1. The call of DO-LOOP leaves the variable incremented *beyond* the limit. The DO-LOOPS may be nested. GPM control-characters within *body* should be quoted.

Example:

```
↑VARIABLE, I;
↑VARIABLE, RESULT, 0;
↑DO-LOOP, I, 1, , 10, <↑ADD, RESULT, ↑I; ; >;
Computes the sum of the integers [1, 10].
The call ↑RESULT; now yields 55.
```

Example:

```
↑VARIABLE, I;
↑VARIABLE, J;
↑DO-LOOP, I, 1, , 3, <
↑I; . _ ↑DO-LOOP, J, 2, 3, 8, <↑J; >; ↑CRLF;
>;
```

yields the following result:

```
1. _ 258
2. _ 258
3. _ 258
```

Example:

```

↑VARIABLE, NUMBER_OF PROGRAMS, 3;
↑VARIABLE, SEGNO, 157;
↑VARIABLE, I;
↑DO-LOOP, I, 1,, ↑NUMBER_OF PROGRAMS;,<
CL-SEGM_↑SEGNO; ↑CRLF;
Y↑CRLF;
N-SEGM_↑SEGNO;,,,,,↑CRLF;
SET-L-A_↑SEGNO;, 100000↑CRLF;
LOAD_MAIN ↑I;:BRF,,,,↑CRLF;
END ↑CRLF;
↑&INCREMENT, SEGNO;>;

```

yields the following result:

```

CL-SEGM_157
Y
N-SEGM_157,,,,,
SET-L-A_157, 100000
LOAD_MAIN1:BRF,,,,
END
CL-SEGM_160
Y
N-SEGM_160
SET-L-A_160, 100000
LOAD_MAIN2:BRF,,,,
END
CL-SEGM_161
Y
N-SEGM_161
SET-L-A_161, 100000
LOAD_MAIN3:BRF,,,,
END

```

3.8 *GPM UNDER SINTRAN III*

The GPM subsystem under SINTRAN III operating system is called by writing GPM.

Example:

(Computer output underlined)

```
@GPM
CR/LF TO BE IGNORED ON INPUT? Y
OUTPUT FILE NAME: OFILE
INPUT FILE NAME: GPM-LIBRARY
INPUT FILE NAME: TERMINAL
>
END OF GPM
@
```

The mode set by the "CR/LF TO BE IGNORED ON INPUT?" - question may be changed by the use of the ICRMOD/CRMOD-Macros.

The GPM library must always be read in "ignore CR/LF" - mode.

The question INPUT FILE NAME: is written whenever the previous input file is exhausted (or none has been specified) or the EOF-byte (27_a = W_c) has been read. An unmatched > outside macro calls terminates GPM.

NOTE: It is strongly recommended that the file "GPM-LIBRARY" should be limited to "READ-ACCESS" only, by using the SET-FILE-ACCESS command. This will protect the file from accidentally being specified as "OUTPUT-FILE" and consequently losing its contents completely.

3.9 *GPM APPLICATIONS - SOME IDEAS*

GPM may of course be applied in a variety of ways ranging from semigraphic picture definitions to software system generation. It may also be used as a pre-processor of symbolic source code, applied prior to compiling/assembling. It is especially well suited for FORTRAN programs since no confusion arises concerning the GPM control characters ↑, < and >. For many programming languages, however, confusion may arise, and one way to avoid it is this: Substitute <↑> for all ↑ that do not denote macro calls. Substitute ↑CHARACTER, 74; for < and ↑CHARACTER, 76; for > if they are not meant as "quotes". Now GPM may process this source-code stream if the GPM-LIBRARY has been read (in order to define the CHARACTER-macro).

3.9.1 *GPM and Semigraphic Display*

GPM is an interesting tool for off-line building of static parts of pictures for semigraphic display (NORDCOM NCT, for instance). Output from GPM may go directly to the screen or to a file where the picture is saved.

The main advantages of using GPM are:

- Control information (concerning colour, for instance) is referenced by name.
- Line segments of variable length may be defined as macros. For instance, a horizontal line of length 46 starting in position (5,7) may be denoted ↑HL,5,7,46;
- Special symbols may be called by name. For instance, ↑TRAFO,12,9; means a transformer symbol in position (12,9).
- Some standard figures such as squares, triangles, etc. may be defined as macros. For instance, ↑SQUARE,10,2,8,16; may yield a square of height 8, length 16, with topmost, leftmost corner in (10,2).
- The user may define and name his own picture parts. The screen position may be parameter in the call.

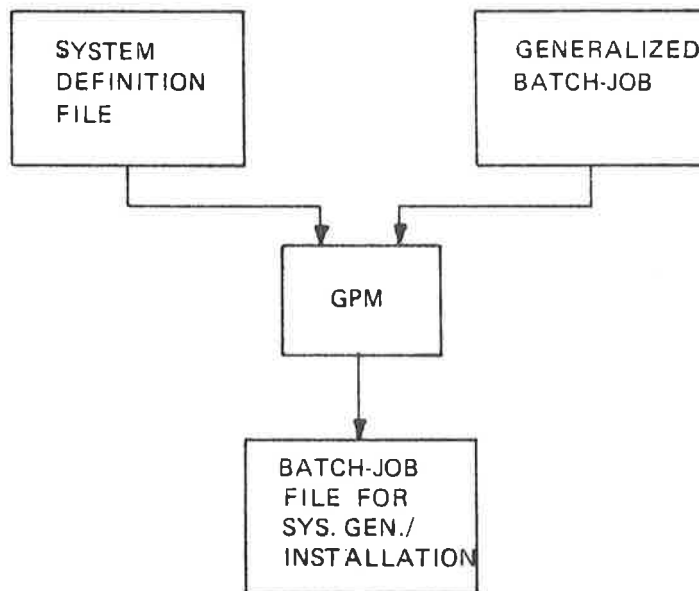
For further details, see the manual NORD PROCESS I/O, Software Guide, Section 10.2.3.

3.9.2 SYSTEM GENERATION USING GPM

GPM is well suited for production of mode or batch jobs for system generation and installation.

GPM then mainly operates as follows:

First GPM reads the "system definition" file, which consists mainly of DEF-macros defining the system parameters. Then GPM reads the "generalized batch-job" file which contains a mixture of ordinary batch commands and macros. From these files, GPM produces that particular batch-job that generates/installs the system given by the "system definition" file.



The most important properties offered by GPM for system generation are listed below:

- 1) Constants may be given symbolic names.
Example: Macro calls for segment-numbers in a MODE-file calling the RT-loader:

```

CL-SEGM_↑SEGNO;
Y
N-SEGM_↑SEGNO; , , , ,
SET-L-A_↑SEGNO; , ↑LOAD-ADDR;
LOAD_↑MAIN↑PROGNO; : BR , , , ,
END
  
```

- 2) Such constants may be modified during system generation. Suppose, in the example given above, that SEGNO and PROGNO have been declared by VARIABLE-macros. The END-command might then be replaced by:

```

END↑&INCREMENT, SEGNO; ↑INCREMENT, PROGNO;
  
```

thus performing octal increment of the segment number and modification of the input file name.

- 3) One macro-call may result in a number of calls in different contexts. It is self-evident that this is possible, since one macro-call may cause (re)definition of a group of other macros. For instance, a call `↑BRF-SYSTEM;` may cause assembling in BRF-mode to a BRF-file and a call of the loader, instead of assembling directly into memory.
- 4) The system parameters may be checked before system-generation if some relations must be fulfilled.

Example:

Suppose that a variable A always has to be greater or equal to variable B if the system is to be consistent.

This macro will check that condition:

```
↑LESS-THAN, ↑A;, ↑B;,
@CC A LESS THAN B! ↑CRLF;
@CC ***SYSTEM GENERATION ABORTED*** ↑CRLF;
↑ESC; ↑ESC;
,;
```

The error message aborts the MODE-file only if $A < B$.

- 5) Do-loops. A group of commands or statements may be repeated with different parameters. Many examples of this have been given previously in this manual.

As a conclusion of this manual, an example showing generalized source code is given.

Suppose you have made a reentrant subroutine SROUT which you want to call from a variable number of RT-programs. Each RT-program is allotted a data-field of 10_a locations for its local variables. In addition subroutine SROUT is called with the A-register pointing to the data-field and with the T-register holding the RT-program number.

For two RT-programs, the NPL source code for calling SROUT looks like this:

```
''' BRF
*)9BEG
*)9EXT SROUT RT1 RT2
'''

SYMBOL PRI=30
INTEGER ARRAY IA1(10)
INTEGER ARRAY IA2(10)
SUBR RPROG
*)9RT RT1 PRI
  '' IA1''; T:=1; CALL SROUT; *MON 0; )FILL
*)9RT RT2 PRI
  '' IA2''; T:=2; CALL SROUT; *MON 0; )FILL
RBUS
''' BRF
*)9END
*)9EOF
'''

*)LINE
@EOF
```

However, this source code may be generalized by calling some GPM-library macros.

The generalized source-code file looks like this:

```

↑CRMOD; *'' BRF
*)9BEG
*)9EXT SROUT ↑SEQUENCE, RT, , ↑NUPROG; , 8, 1, □, *)9EXT;
*''

SYMBOL PRI=30
↑VARIABLE, I; ↑DO-LOOP, I, 1, , ↑NUPROG; , <INTEGER ARRAY IA↑I; (10)
>: SUBR RPROG
↑DO-LOOP, I, 1, , ↑NUPROG; , <*)9RT RT↑I; PRI
    '' IA↑I; '' <;> T:=↑OCT, ↑I; ; <;> CALL SROUT<;> *MON 0<;> )FILL
>; RBUS
*'' BRF
*)9END
*)9EOF
*''

*)LINE
@EOF
>

```

Suppose you call this file GENERAL-SOURCE, and that you let a file called SYSGEN-PARAM hold the definition of the only system parameter, NUPROG, the number of RT-programs. (The definition of NUPROG may of course instead be inserted on top of the GENERAL-SOURCE file). Thus GPM may produce a source code system according to the definition of NUPROG:

```

@GPM
CR/LF TO BE IGNORED ON INPUT? Y
OUTPUT FILE NAME: SOURCE-CODE
INPUT FILE NAME: GPM-LIBRARY
INPUT FILE NAME: SYSGEN-PARAM
INPUT FILE NAME: GENERAL-SOURCE
END OF GPM
@

```


Suppose SYSGEN-PARAM contains †DEF,NUPROG,5;

The following SOURCE-CODE file will then be produced:

```

''' BRF
*)9BEG
*)9EXT SROUT RT1 RT2 RT3 RT4 RT5
'''

SYMBOL PRI=30
INTEGER ARRAY IA1(10)
INTEGER ARRAY IA2(10)
INTEGER ARRAY IA3(10)
INTEGER ARRAY IA4(10)
INTEGER ARRAY IA5(10)
SUBR RPROG
*)9RT RT1 PRI
    '' IA1'' ; T:=1; CALL SROUT; *MON 0; )FILL
*)9RT RT2 PRI
    '' IA2'' ; T:=2; CALL SROUT; *MON 0; )FILL
*)9RT RT3 PRI
    '' IA3'' ; T:=3; CALL SROUT; *MON 0; )FILL
*)9RT RT4 PRI
    '' IA4'' ; T:=4; CALL SROUT; *MON 0; )FILL
*)9RT RT5 PRI
    '' IA5'' ; T:=5; CALL SROUT; *MON 0; )FILL
RBUS
''' BRF
*)9END
*)9EOF
'''

*)LINE
@EOF

```

This source-file yields a system for five RT-programs calling SROUT.

Suppose you define NUPROG equal to 100 and run the system generation procedure. The result is:

```
*"BRF
*)9BEG
*)9EXT SROUT RT1 RT2 RT3 RT4 RT5 RT6 RT7 RT8
*)9EXT RT9 RT10 RT11 RT12 RT13 RT14 RT15 RT16
*)9EXT RT17 RT18 RT19 RT20 RT21 RT22 RT23 RT24
*)9EXT RT25 RT26 RT27 RT28 RT29 RT30 RT31 RT32
*)9EXT RT33 RT34 RT35 RT36 RT37 RT38 RT39 RT40
*)9EXT RT41 RT42 RT43 RT44 RT45 RT46 RT47 RT48
*)9EXT RT49 RT50 RT51 RT52 RT53 RT54 RT55 RT56
*)9EXT RT57 RT58 RT59 RT60 RT61 RT62 RT63 RT64
*)9EXT RT65 RT66 RT67 RT68 RT69 RT70 RT71 RT72
*)9EXT RT73 RT74 RT75 RT76 RT77 RT78 RT79 RT80
*)9EXT RT81 RT82 RT83 RT84 RT85 RT86 RT87 RT88
*)9EXT RT89 RT90 RT91 RT92 RT93 RT94 RT95 RT96
*)9EXT RT97 RT98 RT99 RT100
*"
SYMBOL PRI=30
INTEGER ARRAY IA1(10)
INTEGER ARRAY IA2(10)
INTEGER ARRAY IA3(10)
INTEGER ARRAY IA4(10)
INTEGER ARRAY IA5(10)
INTEGER ARRAY IA6(10)
INTEGER ARRAY IA7(10)
INTEGER ARRAY IA8(10)
INTEGER ARRAY IA9(10)
INTEGER ARRAY IA10(10)
INTEGER ARRAY IA11(10)
INTEGER ARRAY IA12(10)
INTEGER ARRAY IA13(10)
INTEGER ARRAY IA14(10)
INTEGER ARRAY IA15(10)
INTEGER ARRAY IA16(10)
INTEGER ARRAY IA17(10)
INTEGER ARRAY IA18(10)
INTEGER ARRAY IA19(10)
INTEGER ARRAY IA20(10)
INTEGER ARRAY IA21(10)
INTEGER ARRAY IA22(10)
INTEGER ARRAY IA23(10)
INTEGER ARRAY IA24(10)
INTEGER ARRAY IA25(10)
INTEGER ARRAY IA26(10)
INTEGER ARRAY IA27(10)
INTEGER ARRAY IA28(10)
INTEGER ARRAY IA29(10)
INTEGER ARRAY IA30(10)
INTEGER ARRAY IA31(10)
INTEGER ARRAY IA32(10)
INTEGER ARRAY IA33(10)
INTEGER ARRAY IA34(10)
INTEGER ARRAY IA35(10)
INTEGER ARRAY IA36(10)
INTEGER ARRAY IA37(10)
```

INTEGER ARRAY IA38(10)
INTEGER ARRAY IA39(10)
INTEGER ARRAY IA40(10)
INTEGER ARRAY IA41(10)
INTEGER ARRAY IA42(10)
INTEGER ARRAY IA43(10)
INTEGER ARRAY IA44(10)
INTEGER ARRAY IA45(10)
INTEGER ARRAY IA46(10)
INTEGER ARRAY IA47(10)
INTEGER ARRAY IA48(10)
INTEGER ARRAY IA49(10)
INTEGER ARRAY IA50(10)
INTEGER ARRAY IA51(10)
INTEGER ARRAY IA52(10)
INTEGER ARRAY IA53(10)
INTEGER ARRAY IA54(10)
INTEGER ARRAY IA55(10)
INTEGER ARRAY IA56(10)
INTEGER ARRAY IA57(10)
INTEGER ARRAY IA58(10)
INTEGER ARRAY IA59(10)
INTEGER ARRAY IA60(10)
INTEGER ARRAY IA61(10)
INTEGER ARRAY IA62(10)
INTEGER ARRAY IA63(10)
INTEGER ARRAY IA64(10)
INTEGER ARRAY IA65(10)
INTEGER ARRAY IA66(10)
INTEGER ARRAY IA67(10)
INTEGER ARRAY IA68(10)
INTEGER ARRAY IA69(10)
INTEGER ARRAY IA70(10)
INTEGER ARRAY IA71(10)
INTEGER ARRAY IA72(10)
INTEGER ARRAY IA73(10)
INTEGER ARRAY IA74(10)
INTEGER ARRAY IA75(10)
INTEGER ARRAY IA76(10)
INTEGER ARRAY IA77(10)
INTEGER ARRAY IA78(10)
INTEGER ARRAY IA79(10)
INTEGER ARRAY IA80(10)
INTEGER ARRAY IA81(10)
INTEGER ARRAY IA82(10)
INTEGER ARRAY IA83(10)
INTEGER ARRAY IA84(10)
INTEGER ARRAY IA85(10)
INTEGER ARRAY IA86(10)
INTEGER ARRAY IA87(10)
INTEGER ARRAY IA88(10)
INTEGER ARRAY IA89(10)
INTEGER ARRAY IA90(10)
INTEGER ARRAY IA91(10)

```

INTEGER ARRAY IA92(10)
INTEGER ARRAY IA93(10)
INTEGER ARRAY IA94(10)
INTEGER ARRAY IA95(10)
INTEGER ARRAY IA96(10)
INTEGER ARRAY IA97(10)
INTEGER ARRAY IA98(10)
INTEGER ARRAY IA99(10)
INTEGER ARRAY IA100(10)
SUBR RPROG
*)9RT RT1 PRI
    "IA1"; T:=1; CALL SROUT; *MON 0; )FILL
*)9RT RT2 PRI
    "IA2"; T:=2; CALL SROUT; *MON 0; )FILL
*)9RT RT3 PRI
    "IA3"; T:=3; CALL SROUT; *MON 0; )FILL
*)9RT RT4 PRI
    "IA4"; T:=4; CALL SROUT; *MON 0; )FILL
*)9RT RT5 PRI
    "IA5"; T:=5; CALL SROUT; *MON 0; )FILL
*)9RT RT6 PRI
    "IA6"; T:=6; CALL SROUT; *MON 0; )FILL
*)9RT RT7 PRI
    "IA7"; T:=7; CALL SROUT; *MON 0; )FILL
*)9RT RT8 PRI
    "IA8"; T:=10; CALL SROUT; *MON 0; )FILL
*)9RT RT9 PRI
    "IA9"; T:=11; CALL SROUT; *MON 0; )FILL
*)9RT RT10 PRI
    "IA10"; T:=12; CALL SROUT; *MON 0; )FILL
*)9RT RT11 PRI
    "IA11"; T:=13; CALL SROUT; *MON 0; )FILL
*)9RT RT12 PRI
    "IA12"; T:=14; CALL SROUT; *MON 0; )FILL
*)9RT RT13 PRI
    "IA13"; T:=15; CALL SROUT; *MON 0; )FILL
*)9RT RT14 PRI
    "IA14"; T:=16; CALL SROUT; *MON 0; )FILL
*)9RT RT15 PRI
    "IA15"; T:=17; CALL SROUT; *MON 0; )FILL
*)9RT RT16 PRI
    "IA16"; T:=20; CALL SROUT; *MON 0; )FILL
*)9RT RT17 PRI
    "IA17"; T:=21; CALL SROUT; *MON 0; )FILL
*)9RT RT18 PRI
    "IA18"; T:=22; CALL SROUT; *MON 0; )FILL
*)9RT RT19 PRI
    "IA19"; T:=23; CALL SROUT; *MON 0; )FILL
*)9RT RT20 PRI
    "IA20"; T:=24; CALL SROUT; *MON 0; )FILL
*)9RT RT21 PRI
    "IA21"; T:=25; CALL SROUT; *MON 0; )FILL
*)9RT RT22 PRI
    "IA22"; T:=26; CALL SROUT; *MON 0; )FILL

```

```

*)9RT RT23 PRI
    "IA23"; T:=27; CALL SROUT; *MON 0; )FILL
*)9RT RT24 PRI
    "IA24"; T:=30; CALL SROUT; *MON 0; )FILL
*)9RT RT25 PRI
    "IA25"; T:=31; CALL SROUT; *MON 0; )FILL
*)9RT RT26 PRI
    "IA26"; T:=32; CALL SROUT; *MON 0; )FILL
*)9RT RT27 PRI
    "IA27"; T:=33; CALL SROUT; *MON 0; )FILL
*)9RT RT28 PRI
    "IA28"; T:=34; CALL SROUT; *MON 0; )FILL
*)9RT RT29 PRI
    "IA29"; T:=35; CALL SROUT; *MON 0; )FILL
*)9RT RT30 PRI
    "IA30"; T:=36; CALL SROUT; *MON 0; )FILL
*)9RT RT31 PRI
    "IA31"; T:=37; CALL SROUT; *MON 0; )FILL
*)9RT RT32 PRI
    "IA32"; T:=40; CALL SROUT; *MON 0; )FILL
*)9RT RT33 PRI
    "IA33"; T:=41; CALL SROUT; *MON 0; )FILL
*)9RT RT34 PRI
    "IA34"; T:=42; CALL SROUT; *MON 0; )FILL
*)9RT RT35 PRI
    "IA35"; T:=43; CALL SROUT; *MON 0; )FILL
*)9RT RT36 PRI
    "IA36"; T:=44; CALL SROUT; *MON 0; )FILL
*)9RT RT37 PRI
    "IA37"; T:=45; CALL SROUT; *MON 0; )FILL
*)9RT RT38 PRI
    "IA38"; T:=46; CALL SROUT; *MON 0; )FILL
*)9RT RT39 PRI
    "IA39"; T:=47; CALL SROUT; *MON 0; )FILL
*)9RT RT40 PRI
    "IA40"; T:=50; CALL SROUT; *MON 0; )FILL
*)9RT RT41 PRI
    "IA41"; T:=51; CALL SROUT; *MON 0; )FILL
*)9RT RT42 PRI
    "IA42"; T:=52; CALL SROUT; *MON 0; )FILL
*)9RT RT43 PRI
    "IA43"; T:=53; CALL SROUT; *MON 0; )FILL
*)9RT RT44 PRI
    "IA44"; T:=54; CALL SROUT; *MON 0; )FILL
*)9RT RT45 PRI
    "IA45"; T:=55; CALL SROUT; *MON 0; )FILL
*)9RT RT46 PRI
    "IA46"; T:=56; CALL SROUT; *MON 0; )FILL
*)9RT RT47 PRI
    "IA47"; T:=57; CALL SROUT; *MON 0; )FILL
*)9RT RT48 PRI
    "IA48"; T:=60; CALL SROUT; *MON 0; )FILL
*)9RT RT49 PRI
    "IA49"; T:=61; CALL SROUT; *MON 0; )FILL

```

```

*)9RT RT50 PRI
    "IA50": T:=62: CALL SROUT: *MON 0: )FILL
*)9RT RT51 PRI
    "IA51": T:=63: CALL SROUT: *MON 0: )FILL
*)9RT RT52 PRI
    "IA52": T:=64: CALL SROUT: *MON 0: )FILL
*)9RT RT53 PRI
    "IA53": T:=65: CALL SROUT: *MON 0: )FILL
*)9RT RT54 PRI
    "IA54": T:=66: CALL SROUT: *MON 0: )FILL
*)9RT RT55 PRI
    "IA55": T:=67: CALL SROUT: *MON 0: )FILL
*)9RT RT56 PRI
    "IA56": T:=70: CALL SROUT: *MON 0: )FILL
*)9RT RT57 PRI
    "IA57": T:=71: CALL SROUT: *MON 0: )FILL
*)9RT RT58 PRI
    "IA58": T:=72: CALL SROUT: *MON 0: )FILL
*)9RT RT59 PRI
    "IA59": T:=73: CALL SROUT: *MON 0: )FILL
*)9RT RT60 PRI
    "IA60": T:=74: CALL SROUT: *MON 0: )FILL
*)9RT RT61 PRI
    "IA61": T:=75: CALL SROUT: *MON 0: )FILL
*)9RT RT62 PRI
    "IA62": T:=76: CALL SROUT: *MON 0: )FILL
*)9RT RT63 PRI
    "IA63": T:=77: CALL SROUT: *MON 0: )FILL
*)9RT RT64 PRI
    "IA64": T:=100: CALL SROUT: *MON 0: )FILL
*)9RT RT65 PRI
    "IA65": T:=101: CALL SROUT: *MON 0: )FILL
*)9RT RT66 PRI
    "IA66": T:=102: CALL SROUT: *MON 0: )FILL
*)9RT RT67 PRI
    "IA67": T:=103: CALL SROUT: *MON 0: )FILL
*)9RT RT68 PRI
    "IA68": T:=104: CALL SROUT: *MON 0: )FILL
*)9RT RT69 PRI
    "IA69": T:=105: CALL SROUT: *MON 0: )FILL
*)9RT RT70 PRI
    "IA70": T:=106: CALL SROUT: *MON 0: )FILL
*)9RT RT71 PRI
    "IA71": T:=107: CALL SROUT: *MON 0: )FILL
*)9RT RT72 PRI
    "IA72": T:=110: CALL SROUT: *MON 0: )FILL
*)9RT RT73 PRI
    "IA73": T:=111: CALL SROUT: *MON 0: )FILL
*)9RT RT74 PRI
    "IA74": T:=112: CALL SROUT: *MON 0: )FILL
*)9RT RT75 PRI
    "IA75": T:=113: CALL SROUT: *MON 0: )FILL
*)9RT RT76 PRI
    "IA76": T:=114: CALL SROUT: *MON 0: )FILL

```

```

*)9RT RT77 PRI
    "IA77": T:=115; CALL SROUT; *MON 0; )FILL
*)9RT RT78 PRI
    "IA78": T:=116; CALL SROUT; *MON 0; )FILL
*)9RT RT79 PRI
    "IA79": T:=117; CALL SROUT; *MON 0; )FILL
*)9RT RT80 PRI
    "IA80": T:=120; CALL SROUT; *MON 0; )FILL
*)9RT RT81 PRI
    "IA81": T:=121; CALL SROUT; *MON 0; )FILL
*)9RT RT82 PRI
    "IA82": T:=122; CALL SROUT; *MON 0; )FILL
*)9RT RT83 PRI
    "IA83": T:=123; CALL SROUT; *MON 0; )FILL
*)9RT RT84 PRI
    "IA84": T:=124; CALL SROUT; *MON 0; )FILL
*)9RT RT85 PRI
    "IA85": T:=125; CALL SROUT; *MON 0; )FILL
*)9RT RT86 PRI
    "IA86": T:=126; CALL SROUT; *MON 0; )FILL
*)9RT RT87 PRI
    "IA87": T:=127; CALL SROUT; *MON 0; )FILL
*)9RT RT88 PRI
    "IA88": T:=130; CALL SROUT; *MON 0; )FILL
*)9RT RT89 PRI
    "IA89": T:=131; CALL SROUT; *MON 0; )FILL
*)9RT RT90 PRI
    "IA90": T:=132; CALL SROUT; *MON 0; )FILL
*)9RT RT91 PRI
    "IA91": T:=133; CALL SROUT; *MON 0; )FILL
*)9RT RT92 PRI
    "IA92": T:=134; CALL SROUT; *MON 0; )FILL
*)9RT RT93 PRI
    "IA93": T:=135; CALL SROUT; *MON 0; )FILL
*)9RT RT94 PRI
    "IA94": T:=136; CALL SROUT; *MON 0; )FILL
*)9RT RT95 PRI
    "IA95": T:=137; CALL SROUT; *MON 0; )FILL
*)9RT RT96 PRI
    "IA96": T:=140; CALL SROUT; *MON 0; )FILL
*)9RT RT97 PRI
    "IA97": T:=141; CALL SROUT; *MON 0; )FILL
*)9RT RT98 PRI
    "IA98": T:=142; CALL SROUT; *MON 0; )FILL
*)9RT RT99 PRI
    "IA99": T:=143; CALL SROUT; *MON 0; )FILL
*)9RT RT100 PRI
    "IA100": T:=144; CALL SROUT; *MON 0; )FILL
RBUS
*"BRF
*)SEND
*)SEOF
*"
*)LINE

```

3.10 *COMBINED USE OF PERFORM AND GPM*

While GPM is very flexible, allowing the competent user a great variety of transformations, it has the following restrictions:

- It is not possible to enter parameter values interactively into GPM.
- When editing a GPM macro file, there is some risk of errors such as misspelling macro names or making macro calls with incorrect syntax.
- Such errors can cause a considerable number of error messages making it difficult to find the real problem.

The PERFORM subsystem on the other hand is a simple facility for substituting 'MODE file' variables into general purpose MODE files, eg. names of files for compilation or loading. While PERFORM does not have very extensive macro facilities, it is very convenient to be able to enter parameter values interactively.

The combined use of PERFORM and GPM takes advantage of the strengths of both systems, namely interactive input of parameters and accurate substitution into a GPM macro with its powerful transformation facilities. However, the user should be careful when mixing the macros of the two systems; in particular it is advised that a character different from the up arrow character (^) is used for the PERFORM macro's in order not to confuse them with GPM macros.

The following is an example of the combined use of PERFORM and GPM. The steps of this job are:

1. Get parameter values interactively or substitute default values.
2. Use the editor to write some GPM macros to a file.
3. Call GPM to create several Fortran Source files.
4. Call GPM to create MODE files to compile, then load the programs ready for execution.
5. Execute the MODE files which have just been created.

The PERFORM macro to do this job is:

```
%B,SERVICE;
%L,Macro to tailor the remote service system, device numbers;
%P,1,Logical device number of the internal device to be used;
%D,1,200B;
%P,2,Logical device number of the async modem;
%D,2,42;
%P,3,RT-program pair number;
%D,3,1;
%P,4,Segment number for input/output programs;
%D,4,167;
%;
@ QED                                edit some GPM macros
|
|  ↑DEF,INTDEV1,01;
|  ↑DEF,ASYNC,02;
|  ↑DEF,PROCNR,03;
|  ↑DEF,SEGNR,04;
|  Lc                                control L
W SLASK
F
@ GPM                                create first Fortran program
YSERVICE-REMOTE:SYMB
SLASK:SYMB
SERVICE-REMOTE:GPM
@ GPM                                create second Fortran program
YSERVICE-INPUT:SYMB
SLASK:SYMB
SERVICE-INPUT:GPM
@ GPM                                create third Fortran program
YSERVICE-OUTPUT:SYMB
SLASK:SYMB
SERVICE-OUTPUT:GPM
@ GPM                                create a mode file which will
YSERVICE-COMPILE:MODE                compile all the programs
SLASK:SYMB
SERVICE-COMPILE:GPM
@ GPM                                create a mode file which will
YSERVICE-RTLOAD:MODE                load all the programs ready for
SLASK:SYMB                            execution
SERVICE-RTLOAD:GPM
@ MODE SERVICE-COMPILE:MODE,,        execute the compilations
@ MODE SERVICE-RTLOAD:MODE,,        execute the program loading
%E;                                end of PERFORM macro !
```

The percent character (%) has been used to begin macro commands instead of the usual up arrow character (↑), to avoid confusion with the similar function required in the GPM macros.

In order to illustrate the use of GPM in this job the input to GPM to produce the source of the third Fortran program is:

```

^CRM0D;
C
C PROGRAM TO READ FROM ASYNC MODEM AND WRITE TO TERMINAL
C FOR REMOTE MAINTENANCE
C
      PROGRAM OUTPUT^PROCNR;,34
      INTEGER IST,RESRV,ICH,ERRCODE,ASYNC,TERMNO,IERR
      EXTERNAL INPUT^PROCNR;
C
      ASYNC = ^ASYNC;
      ID1   = ^INTDEV1;
C
      IST = RESRV ( ID1, 0, 0 )
      IF(IST .NE. 0) GO TO 9000
      TERMNO = INCH (ID1)
C
      IST = RESRV ( TERMNO, 1, 0 )
      IF(IST .NE. 0) GO TO 9000
      IST = RESRV ( ASYNC, 0, 0 )
      IF(IST .NE. 0) GO TO 9000
C
      IST = IOSET ( ASYNC, 0, 0, -1 )
      IF(IST .NE. 0) GO TO 9000
C
      CALL ECHOM ( ASYNC, -1, 0 )
      CALL BRKM  ( ASYNC, 0, 0 )
C
      CALL RT ( INPUT^PROCNR; )
      CALL RELES ( ID1, 0 )
C
      DO WHILE ( .TRUE. )
          ICH = INCH (ASYNC)
          IF(ERRCODE .NE. 0) GO TO 9000
          CALL OUTCH ( TERMNO, ICH )
          IF(ERRCODE .NE. 0) GO TO 9000
      END DO
C
9000  CONTINUE
      IF(ERRCODE .NE. 0) THEN
          IERR = ERRCODE
          WRITE (TERMNO,9100) IERR
          FORMAT(' ERROR IN OUTPUT PROGRAM, ERRCODE:',I6)
9100  ELSE IF (IST .NE. 0) THEN
          IERR = IST
          WRITE (TERMNO,9200) IERR
          FORMAT(' ERROR IN OUTPUT PROGRAM, STATUS:',I6)
9200  END IF
C
      END
EOF
>

```

If the above macro is used and the following values are input:

```
INTDEV   - 201B
ASYNC    - 42 (default)
PROCNR   - 2
SEGNR    - 201
```

then the Fortran source output from GPM is :

```
C
C PROGRAM TO READ FROM ASYNC MODEM AND WRITE TO TERMINAL
C FOR REMOTE MAINTENANCE
C
  PROGRAM OUTPUT2,34
  INTEGER IST,RESRV,ICH,ERRCODE,ASYNC,TERMNO,IERR
  EXTERNAL INPUT2
C
  ASYNC = 42
  ID1   = 201B
C
  IST = RESRV ( ID1, 0, 0 )
  IF(IST .NE. 0) GO TO 9000
  TERMNO = INCH (ID1)
C
  IST = RESRV ( TERMNO, 1, 0 )
  IF(IST .NE. 0) GO TO 9000
  IST = RESRV ( ASYNC, 0, 0 )
  IF(IST .NE. 0) GO TO 9000
C
  IST = IOSET ( ASYNC, 0, 0, -1 )
  IF(IST .NE. 0) GO TO 9000
C
  CALL ECHOM ( ASYNC, -1, 0 )
  CALL BRKM  ( ASYNC, 0, 0 )
C
  CALL RT ( INPUT2 )
  CALL RELES ( ID1, 0 )
C
  DO WHILE ( .TRUE. )
    ICH = INCH (ASYNC)
    IF(ERRCODE .NE. 0) GO TO 9000
    CALL OUTCH ( TERMNO, ICH )
    IF(ERRCODE .NE. 0) GO TO 9000
  END DO
C
9000 CONTINUE
  IF(ERRCODE .NE. 0) THEN
    IERR = ERRCODE
    WRITE (TERMNO,9100) IERR
9100   FORMAT(' ERROR IN OUTPUT PROGRAM, ERRCODE:',I6)
  ELSE IF (IST .NE. 0) THEN
    IERR = IST
    WRITE (TERMNO,9200) IERR
9200   FORMAT(' ERROR IN OUTPUT PROGRAM, STATUS:',I6)
  END IF
C
  END
EOF
```


4 THE MAIL SYSTEM

4.1 INTRODUCTION

The MAIL system is a facility for sending messages to any interactive user working under your operating system. It operates like a mailbox for users not currently logged on and will attempt to 'deliver' the messages accumulated at LOGON or LOGOFF. Messages may also be sent directly to a terminal device. Some MAIL commands are only available to the user SYSTEM for broadcast, start and stop of the entire MAIL system.

4.2 GENERAL FORMAT

@MAIL <output file>

Parameters:

<output file>

destination of the mail from the terminal user's mailbox. Only requested if the user has mail (DEF = TERMINAL).

Rules:

1. Permitted for all users but some subcommands are restricted as shown below.
2. Messages can be sent in two ways:
 - a) to a mailbox — the recipient is notified when logging in or out and collects mail by entering @MAIL.
 - b) as direct mail — the message is sent immediately.
3. A broadcast is mail to all users, through the mailbox or as direct mail. It can only be sent by user SYSTEM.
4. The mail system can only be used by one user at a time.

4.3 SUBCOMMANDS

1. For all users the following subcommands are available:

*EXIT — exit from the mail system.

*HELP — list all available subcommands.

*SEND-DIRECT-MESSAGE <logical device no.> — type message terminated by CTRL/L. The message is sent to the terminal with this <logical device no.>.

*SEND-MESSAGE <user name> — type message terminated by CTRL/L. The message is sent to the user's mailbox. \$ and ' are handled as for *BROADCAST below.

2. For user SYSTEM the following additional subcommands are available in addition to the ones above:

*BROADCAST — type message terminated by CTRL/L. It is put in the mailbox of all users. \$ is translated to CR, LF. Apostrophe (') is permitted.

*DELETE-BROADCAST <broadcast index> — the message is removed from all mailboxes. <broadcast index> can be found by *LIST-BROADCASTS.

*DELETE-MESSAGE <message no.> — the message is removed from the mailbox. The number can be found by *LIST-MESSAGES.

*DIRECT-BROADCAST — type message terminated by CTRL/L. The message is sent immediately to all terminals. \$ and ' are handled as for *BROADCAST.

*INITIALIZE <max. no. of messages> — this command must be given by user SYSTEM before the mail system can be used. It can be used to reset the mail system. The mail is collected in the file (SYSTEM)MAILBOX:DATA. The maximum length of a message is 512 characters.

*LIST-BROADCASTS <output file> — all broadcasts are listed with their broadcast number on the output file (DEF = TERMINAL).

*LIST-MESSAGES <output file> — as above, but messages are listed.

*RUN-MAIL-SYSTEM — restarts the mail system after SINTRAN start or after a *STOP-MAIL-SYSTEM command. The contents of the mailbox file are retained.

*STOP-MAIL-SYSTEM — the mail system is made unavailable; no mail is lost.

5 BACKUP SYSTEM

5.1 INTRODUCTION

The BACKUP-SYSTEM offers a variety of facilities for copying files, using the COPY-USERS-FILES command, to and from disc and tape media. The files may be copied for archive, backup or other purposes. To enable communication with other installations ANSI standard label format is available for magnetic tapes.

The old SINTRAN commands COPY-USERS-FILES, CREATE-VOLUME and LIST-VOLUME are now available as commands under the BACKUP-SYSTEM, with some extended and altered facilities. While there are new options available, every effort has been made to ensure compatibility and the ability to handle files produced under older versions of the SINTRAN III operating system (prior to the SINTRAN III/F version).

The following documentation is intended to give first an overview of the commands available in the BACKUP-SYSTEM and some of their more important options. The BACKUP-SYSTEM has also a detailed description of all its commands and their options, available interactively while using the system. The 'help' and question mark character (?) functions are available in all levels of dialogue to give descriptions of parameters for the command being used or information about the other commands which may be used.

The following is a list of all the commands and their parameters:

DESCRIBE-ALL-COMMANDS
 <OUTPUT-FILE>

COPY-USERS-FILES

 DESTINATION TYPE:

 DIRECTORY

 <DEST. DIRECTORY-NAME>

 <DEST. USER-NAME>

 VOLUME

 <DEST. VOLUME-NAME>

 <DEST. DEVICE-NAME>

 <DEST. UNIT-NUMBER>

 <DEST. FILE-GENERATION>

SOURCE TYPE:

DIRECTORY

<SOURCE DIRECTORY-NAME>
 <SOURCE USER-NAME>
 <SOURCE FILE-NAME>
 <MANUAL CHECK>

VOLUME

<SOURCE VOLUME-NAME>
 <SOURCE DEVICE-NAME>
 <SOURCE DEVICE-UNIT>
 <SOURCE FILE-GENERATION>
 <SOURCE FILE-NAME>
 <MANUAL CHECK>

PARAMETER-FILE

<PARAMETER-FILE-NAME>
 <MANUAL CHECK>

CREATE-VOLUME

<VOLUME-NAME>
 <DEVICE-NAME>
 <DEVICE-UNIT>

LIST-VOLUME

<DEVICE-NAME>
 <DEVICE-UNIT>
 <FILE-NAME>
 <OUTPUT-FILE>

SERVICE-PROGRAM-CUF

DUMP-BACKUP-SYSTEM

<BPUN-USER-NAME>

MASTER-LOG-MODE

<MASTER-LOG-FILE>
 <APPEND-ACCESS>

MODE-STANDARD-VOLUME

MANUAL-STANDARD-VOLUME

MODE-BACKUP-SYSTEM-VOLUME

USER-COPY-LOG-MODE

<LOG-FILE>
 <APPEND-ACCESS>

SET-ALLOCATE-CREATE-DEFAULT

<DEFAULT ANSWER>

SET-SINGLE-SEARCH

RESET-SINGLE-SEARCH

EXIT

EXIT

5.2 *SIMPLE USE OF THE BACKUP-SYSTEM*

The BACKUP-SYSTEM may carry out simple tasks by using the COPY-USERS-FILES command to copy some files. If magnetic tape is to be used, the CREATE-VOLUME command should be used first and the user executing this command becomes the owner of the VOLUME. VOLUME's will be written in the BACKUP-SYSTEM's default format. VOLUME's produced by the old COPY-USERS-FILES command (before SINTRAN III/F) can also be read. All available different magnetic tape formats, produced by the BACKUP-SYSTEM or SINTRAN COPY-USERS-FILES, are automatically detected.

5.3 *COMMANDS*

The system can be entered by using the command:

@BACKUP-SYSTEM

Once the BACKUP-SYSTEM has been entered the following commands are available:

DESCRIBE-ALL-COMMANDS

will give detailed descriptions of each command available and its options and parameters. Listing this command on a hard-copy device is recommended for an inexperienced user.

EXIT

leave the BACKUP-SYSTEM and return to the SINTRAN III operating system.

CREATE-VOLUME

creates a 'VOLUME' on magnetic tape. Only one VOLUME may exist on a tape. A VOLUME may, following the use of the Create command, be written in different formats, STANDARD-VOLUME and BACKUP-SYSTEM-VOLUME, see options under SERVICE-PROGRAM-CUF. A VOLUME can contain files from many users, but it is owned by the user who created the VOLUME, and can only be accessed by the owner or by the user SYSTEM.

LIST-VOLUME

will list the contents of a VOLUME on magnetic tape.

COPY-USERS-FILES

will copy one or more files from a user on one medium to a user on the same medium or a different medium. For media selection, there are options available in the SERVICE-program-CUF to assist with more complex copying requirements. File accessing is by the normal SINTRAN III rules. However, user SYSTEM can access any user's files with the same access rights as the file owner, allowing files to be copied on behalf of a user.

If copying from or to a VOLUME, a user can only access his own tapes. User SYSTEM can, however, have both read and write access to tapes other than his own.

Note: that while DIRECTORY, VOLUME and PARAMETER-FILE are referred as sub-commands, they describe the destination and source types respectively.

If copying between directories, the DESTINATION user may be different from the SOURCE user. If the SOURCE medium is a VOLUME, the parameter DEST. USER-NAME will choose between the original owner of the file or a new user-name. If a new user-name is specified, you will be asked if you want to copy to this new user.

If copying between directories, and if DESTINATION-file already exists, the source and destination date for last opened for write is checked. If the destination is written to later than source, you will be asked if you copy the right direction.

The user must ensure enough space is available for all files to be copied. The BACKUP-SYSTEM will create all the necessary file names.

The BACKUP-SYSTEM will only access the DEFAULT directory of a user when no explicit name is given for the DIRECTORY-NAME. Any directory may be accessed by giving its name explicitly.

Use of the COPY-USERS-FILES command will also result in the contents of the fields FILE-ACCESS, LAST-DATE OPENED FOR READ, LAST-DATE OPENED FOR WRITE, CREATION-DATE and MAX BYTE POINTER being copied from the source file to the destination file.

If you are user SYSTEM or have DIRECTORY-ACCESS to the source, the last date OPENED FOR READ and number of times OPENED will not be updated.

SERVICE-PROGRAM-CUF

can be used to select from the various options relating to the COPY-USERS-FILES command.

The following commands are available under the SERVICE-PROGRAM-CUF:

EXIT

leaves the SERVICE-PROGRAM-CUF and returns to the BACKUP-SYSTEM.

DUMP-BACKUP-SYSTEM

dumps the BACKUP-SYSTEM on the file 'BACKUP-SYSTEM:BPUN'.

MODE-STANDARD-VOLUME, MANUAL-STANDARD-VOLUME,
MODE-BACKUP-SYSTEM-VOLUME

These options are only significant for output to magnetic tape. VOLUMES's exist on magnetic tape only. The information on a VOLUME may be in the following formats:

- STANDARD-VOLUMES - are similar to ANSI defined format, compatible to SINTRAN III/E and earlier versions of SINTRAN.
- BACKUP-SYSTEM-VOLUME — are similar to ANSI defined format plus some SINTRAN -III file system information.

Note that one VOLUME may contain files written in a mixture of these formats.

The device MAG-TAPE-1 unit 0 must have the name MAG-TAPE-1-0, unit 1 must have the name MAG-TAPE-1-1, etc. This can be set by using the command SET-PERIPHERAL-FILE.

SET-ALLOCATE-CREATE-DEFAULT

During file copying, the BACKUP-SYSTEM will require operator input if it cannot Allocate or Create contiguous files, as they are described by the file system information on the original directory or VOLUME. If this situation arises and the operator inputs 'yes', then the following rules apply:

1. Allocated source files will be created as contiguous files if possible or else they will be Created as Indexed files.
2. Contiguous files will be Created as Indexed files.

If the operator inputs 'no', then such files will not be copied. This option may be set to give a default answer to all such questions. This option applies to BACKUP-SYSTEM files only.

This facility is an aid for copying many files interactively and should be used for MODE and Batch jobs.

SET-SINGLE-SEARCH, RESET-SINGLE-SEARCH

SINGLE-SEARCH operates in the same way as the normal search until one file or a group of consecutive files have been copied. The search begins from wherever the tape is positioned, and no tape rewinds are done while in SINGLE-SEARCH mode. Copying terminates at the first non-matching file-name. SINGLE-SEARCH makes it possible to copy a number of files with one pass through a tape. In order to achieve this, the files must be selected in the same order as they appear on the tape. Care must be taken when copying files to tape if SINGLE-SEARCH is to successfully gather all files which a user wishes to retrieve.

MASTER-LOG-MODE, USER-COPY-LOG-MODE

there are two LOG-MODE's, MASTER-LOG for user SYSTEM only, and USER-LOG for public users only. These 'modes' cause copy command information to be written into a LOG file.

5.4

COMMANDS — DETAILED DESCRIPTION

The following is the complete output that can be obtained by using the DESCRIBE-ALL-COMMANDS for all the available commands:

MAIN COMMANDS BACKUP-SYSTEM

COMMAND NUMBER: 1 LEAD TEXT: BA-SY

DESCRIBE-ALL-COMMANDS
<LIST FILE>

... DESCRIBES ALL BACKUP-SYSTEM COMMANDS
... WITH THEIR CORRESPONDING PARAMETERS.
...
... --- GENERAL INFORMATION ---
...
... IF A PARAMETER HAS DEFAULT VALUE, IT WILL BE DISPLAYED BETWEEN SLASHES
... (/ ... /) FOLLOWING THE PARAMETER NAME.
... "EMPTY DEFAULT" MAY OCCUR, INDICATING THAT AN ANSWER IS NOT REQUIRED.
...
... THREE BUILT-IN FUNCTIONS ARE AVAILABLE IN COMMAND INPUT: HELP,(?),(ESC).
... . HELP : FUNCTION FOR LISTING COMMANDS OR A SUBSET OF COMMANDS.
... . HELP HAS COMMAND-NAME AS PARAMETER. IF HELP IS TYPED
... . AS PARAMETER TO HELP THE COMMANDS WILL BE LISTED WITH
... . THEIR CORRESPONDING EXPLANATIONS.
... . ? : A (?) FOLLOWING AN AMBIGUOUS COMMAND ACTS AS HELP WITH
... . THE COMMAND-NAME AS PARAMETER. IF THE COMMAND IS UNIQUE
... . IT GIVES AN EXPLANATION OF THIS COMMAND. A (?) GIVEN
... . IN PLACE OF A PARAMETER WILL EXPLAIN THIS PARAMETER.
... . (ESC): ESCAPE CAN BE USED TO ABORT PARAMETER COLLECTING IN
... . A COMMAND.
...
... IF (ESC) IS ANSWERED TO A QUESTION FROM AN EXECUTING COMMAND, THE COMMAND
... IS ABORTED.
... IF (ESC) IS GIVEN BETWEEN COMMUNICATION-STATES, IT CAUSES EXIT
... OR USER-BREAK.
... THE BACKUP-SYSTEM WILL ACCEPT SEVERAL COMMANDS WRITTEN ON THE SAME LINE.
... WHEN THESE COMMANDS ARE PROCESSED, THE BACKUP-SYSTEM WILL TRACE THEM BY
... OUTPUTTING THE LEAD-TEXT,COMMAND-NAMES AND PARAMETERS COLLECTED.

PARAMETER NUMBER: 1 LIST FILE
... SPECIFY NAME OF LIST FILE (MAX. 16 CHARACTERS)
... (DEFAULT VALUE: OWN TERMINAL)

COMMAND NUMBER: 2 LEAD TEXT: BA-SY

COPY-USERS-FILES

... COPIES A FILE OR A SET OF FILES.
... SOURCE AND DESTINATION CAN BE ONE OF THE FOLLOWING TYPES:
... ! DIRECTORY: - ANY FILE-SYSTEM DIRECTORY WHICH HAS USERS.
... ! VOLUME: - ANSI LABELED MAGNETIC TAPE.
... ! PARAMETER-FILE: - VALID FOR SOURCE TYPE ONLY.
... ! IT CONTAINS THE NAMES OF FILES TO BE COPIED.
... ! THESE FILES MUST RESIDE ON A DIRECTORY.
...
... DIRECTORY,VOLUME,PARAMETER-FILE ARE SUB-COMMANDS AVAILABLE ONLY
... UNDER THE COPY-USERS-FILES COMMAND.
... IF AN ERROR OCCURS IN COPYING A FILE, THE DESTINATION-FILE IS
... NORMALLY DELETED.

DESTINATION TYPE - SUB-COMMANDS:

 COMMAND NUMBER: 1 LEAD TEXT: DESTINATION TYPE

DIRECTORY

<DEST. DIRECTORY-NAME>
 <DEST. USER-NAME>

... DIRECTORY INDICATES THAT THE DESTINATION DEVICE IS A DIRECTORY.

PARAMETER NUMBER: 1 DEST. DIRECTORY-NAME

... SPECIFY NAME OF DESTINATION DIRECTORY. (MAX. 16 CHARACTERS)
 ... (DEFAULT VALUE: DEFAULT DIRECTORY)

PARAMETER NUMBER: 2 DEST. USER-NAME

... SPECIFY DESTINATION USER NAME. (MAX. 16 CHARACTERS)
 ... (DEFAULT VALUE: OWN USER NAME)
 ... WHEN SOURCE IS VOLUME, YOU CAN CHOOSE BETWEEN THIS USER-NAME AND
 ... THE USER-NAMES INCLUDED IN THE FILE NAMES ON THE VOLUME, OR
 ... YOU CAN GIVE A NEW USER-NAME WHEN IT IS CHANGED IN THE VOLUME.

 COMMAND NUMBER: 2 LEAD TEXT: DESTINATION TYPE

VOLUME

<DEST. VOLUME-NAME>
 <DEST. DEVICE-NAME>
 <DEST. UNIT-NUMBER>
 <DEST. FILE-GENERATION>

... VOLUME INDICATES THAT DESTINATION DEVICE IS A MAG-TAPE VOLUME.
 ... IF ONE FILE IS TOO BIG FOR THE VOLUME, YOU MAY CONTINUE ON THE NEXT.

PARAMETER NUMBER: 1 DEST. VOLUME-NAME

... SPECIFY NAME OF DESTINATION VOLUME. (MAX. 6 CHARACTERS)

PARAMETER NUMBER: 2 DEST. DEVICE-NAME

... SPECIFY NAME OF DEVICE (MAG-TAPE-1 OR MAG-TAPE-2).

PARAMETER NUMBER: 3 DEST. UNIT-NUMBER

... SPECIFY UNIT NUMBER WHERE VOLUME IS MOUNTED.
 ... (MAX. UNIT NUMBER: 3)

PARAMETER NUMBER: 4 DEST. FILE-GENERATION

... SPECIFY FILE GENERATION WANTED FOR FILES. (MAX. 4 CHARACTERS)

SOURCE TYPE - SUB-COMMANDS:

 COMMAND NUMBER: 1 LEAD TEXT: SOURCE TYPE

DIRECTORY

<SOURCE DIRECTORY-NAME>
 <SOURCE USER-NAME>
 <SOURCE FILE-NAME>
 <MANUAL CHECK>

... DIRECTORY INDICATES THAT SOURCE DEVICE IS A DIRECTORY.

PARAMETER NUMBER: 1 SOURCE DIRECTORY-NAME
 ... SPECIFY NAME OF DIRECTORY CONTAINING SOURCE-FILES.
 ... (MAX. 16 CHARACTERS) (DEFAULT: DEFAULT DIRECTORY.)

PARAMETER NUMBER: 2 SOURCE USER-NAME
 ... SPECIFY USER NAME OF OWNER OF SOURCE-FILES.
 ... (MAX. 16 CHARACTERS) (DEFAULT VALUE: OWN USER)

PARAMETER NUMBER: 3 SOURCE FILE-NAME
 ... SPECIFY FILE-NAME OF FILES TO BE COPIED.
 ... (MAX. 21 CHARACTERS) (DEFAULT: ALL USERS FILES)

PARAMETER NUMBER: 4 MANUAL CHECK
 ... YES,NO,L : YES, MEANS STOP BEFORE EACH FILE IS COPIED.
 ... ! NO, MEANS NO CHECK WITH OPERATOR.
 ... ! L , MEANS ALL FILES COPIED WILL THEN BE LISTED.

 COMMAND NUMBER: 2 LEAD TEXT: SOURCE TYPE

VOLUME

<SOURCE VOLUME-NAME>
 <SOURCE DEVICE-NAME>
 <SOURCE DEVICE-UNIT>
 <SOURCE FILE-GENERATION>
 <SOURCE FILE-NAME>
 <MANUAL CHECK>

... VOLUME INDICATES THAT SOURCE DEVICE IS A MAG-TAPE VOLUME.
 ... IF THERE IS AN ERROR IN A SOURCE-FILE, YOU WILL GET DIFFERENT
 ... QUESTIONS IF YOU WILL TRY TO RECOVER OR SKIP THE FILE.

PARAMETER NUMBER: 1 SOURCE VOLUME-NAME
 ... SPECIFY NAME OF MAG-TAPE VOLUME. (MAX. 6 CHARACTERS)

PARAMETER NUMBER: 2 SOURCE DEVICE-NAME
 ... SPECIFY DEVICE-NAME OF MAG-TAPE. (MAG-TAPE-1 OR MAG-TAPE-2)

PARAMETER NUMBER: 3 SOURCE DEVICE-UNIT
 ... SPECIFY UNIT NUMBER OF MAG-TAPE. (MAX. UNIT NUMBER: 3)

PARAMETER NUMBER: 4 SOURCE FILE-GENERATION
 ... SPECIFY FILE GENERATION OF MAG-TAPE SOURCE-FILES.
 ... (MAX. 4 CHARACTERS) (DEFAULT: ALL GENERATIONS)

PARAMETER NUMBER: 5 SOURCE FILE-NAME
 ... SPECIFY FILE-NAME OF SOURCE-FILES. (MAX. 39 CHARACTERS)
 ... (DEFAULT: ALL FILES OF SPECIFIED FILE GENERATION)

PARAMETER NUMBER: 6 MANUAL CHECK
 ... YES,NO,L : YES, MEANS STOP BEFORE EACH FILE IS COPIED.
 ... ! NO, MEANS NO CHECK WITH OPERATOR.
 ... ! L , MEANS ALL FILES COPIED WILL THEN BE LISTED.

COMMAND NUMBER: 3 LEAD TEXT: SOURCE TYPE

PARAMETER-FILE

<PARAMETER-FILE-NAME>
<MANUAL CHECK>

... PARAMETER-FILE SPECIFIES THAT SOURCE-FILE SELECTION
... IS CONTROLLED BY COMMANDS FROM A FILE WHICH CONTAINS
... A LIST OF FILE-NAMES.
... USING SUCH A PARAMETER-FILE HAS THE SAME EFFECT AS USING
... A COPY COMMAND FOR EACH FILE-NAME IN THE PARAMETER-FILE.
... ALL FILES WILL BE COPIED TO THE SAME DESTINATION.
... ALL SOURCE FILES MUST RESIDE ON A DIRECTORY.
...
... A LEFT PARENTHESIS "(" APPEARING ANYWHERE IN A LINE,
... DEFINES THE BEGINNING OF A FILE NAME. THE FIRST FOLLOWING
... SPACE WILL TERMINATE EACH FILE-NAME.
... LINES WITH DIFFERENT LAYOUT WILL BE IGNORED.
... GENERAL LAYOUT: (DIRECTORY:USER)FILE-NAME
... DIRECTORY NAME MAY BE OMITTED IN THE FILE NAME.

PARAMETER NUMBER: 1 PARAMETER-FILE-NAME

... SPECIFY PARAMETER-FILE NAME. (MAX. 21 CHARACTERS)

PARAMETER NUMBER: 2 MANUAL CHECK

... YES,NO,L : YES, MEANS STOP BEFORE EACH FILE IS COPIED.
... ! NO, MEANS NO CHECK WITH OPERATOR.
... ! L , MEANS ALL FILES COPIED WILL THEN BE LISTED.

COMMAND NUMBER: 3 LEAD TEXT: BA-SY

CREATE-VOLUME

<VOLUME-NAME>
 <DEVICE-NAME>
 <DEVICE-UNIT>

... CREATES A VOLUME ON A MAGNETIC TAPE. AFTER THIS COMMAND
 ... THE OLD INFORMATION ON THIS TAPE WILL BE UNAVAILABLE.

PARAMETER NUMBER: 1 VOLUME-NAME
 ... SPECIFY VOLUME NAME. (MAX. 6 CHARACTERS)

PARAMETER NUMBER: 2 DEVICE-NAME
 ... SPECIFY DEVICE NAME WHERE THE TAPE IS MOUNTED.
 ... (MAG-TAPE-1 OR MAG-TAPE-2)

PARAMETER NUMBER: 3 DEVICE-UNIT
 ... SPECIFY THE UNIT NUMBER WHERE THE TAPE IS MOUNTED. (0-3)

COMMAND NUMBER: 4 LEAD TEXT: BA-SY

LIST-VOLUME

<DEVICE-NAME>
 <DEVICE-UNIT>
 <FILE-NAME>
 <OUTPUT-FILE>

... COMMAND TO LIST THE CONTENTS OF A VOLUME.

PARAMETER NUMBER: 1 DEVICE-NAME
 ... SPECIFY DEVICE NAME ON WHICH THE VOLUME IS TO BE FOUND.
 ... (MAG-TAPE-1 OR MAG-TAPE-2)

PARAMETER NUMBER: 2 DEVICE-UNIT
 ... SPECIFY UNIT NUMBER ON WHICH THE VOLUME IS TO BE FOUND (0-3)

PARAMETER NUMBER: 3 FILE-NAME
 ... NAME OF FILES TO BE LISTED FROM VOLUME (STATED AS IN
 ... LIST-FILE IN SINTRAN NOT INCLUDING DIRECTORY AND USER NAME.)

PARAMETER NUMBER: 4 OUTPUT-FILE
 ... OUTPUT-FILE NAME FOR LISTING

COMMAND NUMBER: 5 LEAD TEXT: BA-SY

SERVICE-PROGRAM-CUF

... ENTERS A SERVICE PROGRAM FOR COPY-USERS-FILES.
 ... IT COMPRISES A SET OF COMMANDS FOR CHANGING DEFAULT VALUES
 ... AND MODES FOR COPY-USERS-FILES. SOME COMMANDS ARE RESTRICTED
 ... TO USER SYSTEM AND THEY WILL HAVE AN ASTERISK (*) IN THE
 ... COMMAND NAME.
 ...
 ... TO LEAVE THE SERVICE PROGRAM USE COMMAND: EXIT

CUF-SERV - SUB-COMMANDS:

COMMAND NUMBER: 1 LEAD TEXT: CUF-SERV

DUMP-BACKUP-SYSTEM
 <BPUN-USER-NAME>

... DUMPS THE BACKUP-SYSTEM ON THE FILE:
 ... ! BACKUP-SYSTEM-C:BPUN
 ... THIS FILE MUST EXIST BEFORE A DUMP COMMAND CAN BE EXECUTED
 ... AND IT CAN BELONG TO ANY SPECIFIED USER.
 ... THE COMMAND IS INTENDED TO BE USED WHEN DEFAULT VALUES AND
 ... MODES HAVE BEEN CHANGED. THIS COMMAND IS RESTRICTED TO
 ... USER SYSTEM.

PARAMETER NUMBER: 1 BPUN-USER-NAME
 ... SPECIFY USER-NAME OF USER WHERE YOU KEEP YOUR BPUN-FILES
 ... DEFAULT USER IN THE BACKUP-SYSTEM IS USER SYSTEM.

COMMAND NUMBER: 2 LEAD TEXT: CUF-SERV

MASTER-LOG-MODE
 <MASTER-LOG-FILE>
 <APPEND-ACCESS>

... RESTRICTED TO USER SYSTEM.
 ... IF LOG-FILE IS DEFINED, THEN DESTINATION, SOURCE, DATE
 ... OF COPYING AND NAME OF FILES COPIED WILL BE LOGGED.
 ... IF THE DUMP COMMAND IS USED AFTER THIS COMMAND, THE LOG-
 ... FILE MUST ALWAYS BE PRESENT WHEN COPYING AS USER SYSTEM.

PARAMETER NUMBER: 1 MASTER-LOG-FILE
 ... SPECIFY FILE-NAME OF WANTED LOG-FILE OR (CR) TO RESET
 ... MASTER-LOG-MODE.

PARAMETER NUMBER: 2 APPEND-ACCESS
 ... YES-NO : YES, MEANS APPEND TO THE LOG-FILE,
 ... ! NO, MEANS WRITE FROM START (NO HISTORY)

COMMAND NUMBER: 3 LEAD TEXT: CUF-SERV

MODE-STANDARD-VOLUME

... WHEN THE BACKUP-SYSTEM IS USED IN THIS MODE, THE VOLUMES
 ... PRODUCED WILL BE COMPATIBLE WITH S-III COPY-USERS-FILES
 ... VOLUMES, AND CAN BE USED WITH ALL VERSIONS OF SINTRAN-III
 ... (THIS COMMAND WILL ONLY AFFECT OUTPUT TO TAPE)

COMMAND NUMBER: 4 LEAD TEXT: CUF-SERV

MANUAL-STANDARD-VOLUME

... PLACES THE BACKUP-SYSTEM IN THE SAME MODE AS THE
 ... MODE-STANDARD-VOLUME COMMAND.
 ... THE EXCEPTION IS FILES WITH "HOLES". THE SYSTEM WILL
 ... ASK IF SUCH FILES SHOULD BE COPIED OR SKIPPED.
 ... (THIS COMMAND WILL ONLY AFFECT OUTPUT TO TAPE)

COMMAND NUMBER: 5 LEAD TEXT: CUF-SERV

MODE-BACKUP-SYSTEM-VOLUME

... THIS MODE CAN ONLY BE USED WHEN THE VOLUME IS TO BE USED SOLELY
 ... BY THE BACKUP SYSTEM. THE VOLUMES CANNOT BE INTERCHANGED
 ... WITH SINTRAN-III COPY-USERS-FILES.
 ... THE DIFFERENCES IN THE VOLUMES CONCERNS FILES WITH "HOLES",
 ... WHERE IN THIS MODE, HOLES ARE MARKED ON THE TAPE BY A SPECIAL
 ... LABEL, ENABLING THE BACKUP-SYSTEM TO "REMEMBER" THE LOGICAL
 ... LAYOUT OF A FILE.
 ... THE BACKUP-SYSTEM IS IN THIS MODE BY DEFAULT.
 ... (THIS COMMAND WILL ONLY AFFECT OUTPUT TO TAPE)

COMMAND NUMBER: 6 LEAD TEXT: CUF-SERV

USER-COPY-LOG-MODE

<LOG-FILE>

<APPEND-ACCESS>

... IF LOG-FILE IS DEFINED, THEN DESTINATION, SOURCE, DATE
 ... OF COPYING AND NAME OF FILES COPIED WILL BE LOGGED.
 ... THIS COMMAND IS RESTRICTED TO PUBLIC USERS, AND WILL HAVE
 ... NO EFFECT IF USED UNDER USER SYSTEM. USER SYSTEM SHOULD
 ... USE THE MASTER-LOG-MODE COMMAND.

PARAMETER NUMBER: 1 LOG-FILE

... SPECIFY FILE NAME OF LOG-FILE OR (CR) TO RESET USER-LOG-MODE

PARAMETER NUMBER: 2 APPEND-ACCESS

... YES-NO : YES, MEANS APPEND TO FILE

... ! NO, MEANS WRITE FROM START (NO HISTORY)

COMMAND NUMBER: 7 LEAD TEXT: CUF-SERV

SET-ALLOCATE-CREATE-DEFAULT
<DEFAULT ANSWER>

... THE BACKUP-SYSTEM WILL TRY TO ALLOCATE OR CREATE
... A DESTINATION-FILE EQUIVALENT TO THE SOURCE-FILE. THIS MAY NOT
... ALWAYS BE POSSIBLE FOR ALLOCATED OR CONTIGUOUS FILES. IF
... THE FILE CANNOT BE CREATED AS DEFINED BY THE SOURCE FILE,
... THE OPERATOR WILL BE ASKED FOR INSTRUCTIONS ABOUT THIS FILE.
... OPTIONS ARE SKIP FILE OR TRY TO MAKE THE FILE CONTIGUOUS
... OR INDEXED. THE QUESTION REQUIRES A YES-NO ANSWER,
... WHERE YES MEANS TRY, NO MEANS SKIP.

... THIS COMMAND CAN SPECIFY A DEFAULT ANSWER FOR ACTIONS TO BE
... TAKEN WHEN REQUIRED BY THE BACKUP-SYSTEM.
... THE FILES WILL THEN BE TREATED ACCORDING TO THIS ANSWER.
... THE BACKUP-SYSTEM HAS INITIALLY NO DEFAULT ANSWER, QUESTIONS
... MUST BE ANSWERED FROM THE TERMINAL. TO RESET THE BACKUP-SYSTEM
... TO THIS STATE, SIMPLY TYPE CARRIAGE-RETURN WHEN ASKED FOR
... DEFAULT-ANSWER IN THIS COMMAND.

PARAMETER NUMBER: 1 DEFAULT ANSWER
... ANSWER SHOULD BE YES, NO, OR CARRIAGE RETURN.

COMMAND NUMBER: 8 LEAD TEXT: CUF-SERV

SET-SINGLE-SEARCH

... THE NORMAL SEARCH ALGORITHM ON TAPE IS FROM BEGINNING OF
... VOLUME TO END OF VOLUME IN ORDER TO FIND ALL FILES MATCHING
... A GIVEN FILE-NAME.
... SINGLE-SEARCH OPERATES IN THE SAME WAY UNTIL ONE MATCHING
... FILE OR GROUP OF FILES HAVE BEEN COPIED. COPYING TERMINATES
... AT THE FIRST NON-MATCHING FILE-NAME.
... THE TAPE REMAINS POSITIONED AFTER THE LAST COPIED FILE.
... SINGLE-SEARCH MAKES IT POSSIBLE TO COPY A NUMBER OF DIFFERENT
... FILES, WITH ONE PASS THROUGH THE TAPE. IN ORDER TO ACHIEVE THIS
... THE FILES MUST BE SELECTED IN THE SAME ORDER AS THEY APPEAR ON
... THE TAPE.
... (THIS COMMAND ONLY AFFECTS INPUT FROM TAPE.)

COMMAND NUMBER: 9 LEAD TEXT: CUF-SERV

RESET-SINGLE-SEARCH

... RESETS TO THE NORMAL SEARCH ALGORITHM.
... (THIS IS THE NORMAL AND DEFAULT MODE FOR THE BACKUP-SYSTEM)

COMMAND NUMBER: 10 LEAD TEXT: CUF-SERV

EXIT

... RETURN TO THE BACKUP-SYSTEM.

COMMAND NUMBER: 6 LEAD TEXT: BA-SY

EXIT

... LEAVES THE BACKUP-SYSTEM AND RETURNS TO SINTRAN-III.

5.5 LABEL FORMATS ON MAGNETIC TAPE VOLUMES

Implementation of magnetic tape VOLUME's in the SINTRAN-III BACKUP-SYSTEM is based upon:

American National Standard Magnetic Tape Labels for Information Interchange X3.27-1969.

However, some deviations from the standard have been made. Deviations are marked by a dollar sign (\$) in the explanation.

General rules:

- the general tape layout is as follows

VOL1 HDR1 HDR2 UHL1*-file1-*EOF1*HDR1 HDR2 UHL1*-file2-* $\left\{ \begin{array}{c} \text{EOF1} \\ \text{OR} \\ \text{EOV1} \end{array} \right\} **$

where VOL1,HDR1,HDR2,UHL1, EOF1 and EOV1 are tape labels, and asterisks are tape marks.

- All labels are 80 character blocks.
- All information in the labels are recorded as ASCII characters with the parity bit cleared.
All unused character positions will contain spaces.
\$\$\$ The user option field (3) in the label UHL1 contains binary information.
- File data is recorded as 2048 character blocks.
These blocks may contain any character. (0-255 dec.)

\$\$\$\$ Deviation From Standard

- Only the first file on a volume may be a multivolume-file.
- A non standard label, HOLE, has been introduced.
This label can be inserted between the file data blocks.
The important information in this label is a 32-bit binary number contained in characters 77-80 of the label. The backup-system uses this number in the following way:
 - Each 2048 character block on the tape corresponds to a 1024 16-bit word block on the disk referred to as a page. The pages are numbered 0, 1, 2, 3, etc. to establish a logical sequence of pages. If the logical sequences is not contiguous, then a 'HOLE label' defines where the next block on the tape logically belongs in the disk file. In order to represent a 'logical HOLE' on the tape, the HOLE label will be inserted in front of the next block, stating this block's logical number. Blocks of 2048 characters without a HOLE label are expected to belong to a contiguous logical area and will cause the logical block number to be incremented by one.

Example:

```

log. block no:  0      5    6    7      100 101      120
                data HOLE data data data HOLE data data HOLE data
                (5)                (100)          (120)

```

where data represents file data blocks (2048 characters) and HOLE is a HOLE label, contents of HOLE label in ().

VOLUME HEADER LABEL

POSITION	FIELD	NAME	LENGTH	CONTENTS
1- 3	1	label identifier	3	VOL
4	2	label number	1	1
5-10	3	volume serial number	6	(volume name) \$
11	4	accessibility	1	(space)
12-31	5	(not used)	20	(spaces)
32-37	6	(not used)	6	(spaces)
38-51	7	owner identification	14	(name of owner) \$
52-79	8	(not used)	28	(spaces)
80	9	label standard level	1	(spaces)

\$ field 3 and 7

- These fields contain any alphanumeric characters. If the field is not fully filled with characters, the last character in the string is a apostrophe. This character is used to mark the end of string and is not part of the name. The unused part of such a field is filled with spaces.

FIRST FILE HEADER LABEL

POSITION	FIELD	NAME	LENGTH	CONTENTS
1- 3	1	label identifier	3	HDR
4	2	label number	1	1
5-21	3	file identifier	17	(file name) \$
22-27	4	set identification	6	(file type) \$
28-31	5	file section number	4	(0001-0002-nnnn)
32-35	6	file sequence number	4	(0001-0002-nnnn)
36-39	7	generation number	4	(file generation) \$
40-41	8	generation version number	2	(version number) \$
42-47	9	creation date	6	(ANSI Standard date)
48-53	10	expiration date	6	(spaces) \$
54	11	accessibility	1	(space)
55-60	12	block count	6	000000
61-73	13	system code	13	(spaces)
74-80	14	(not used)	7	(spaces)

\$ field 3:

- Apostrophe is used to mark end of string. This character is not part of the name. Unused part of field is filled with spaces.

\$ field 4:

- Only the four first characters are used in this field. If shorter than four characters, apostrophe is used to mark end of string.

\$ field 7:

- Any alphanumeric characters. Field is left justified and apostrophe is used to mark end of string. The character code in this field identifies a backup generation of files.

\$ field 8:

- This field contains numbers from 1 to 99. Characters are left justified and one digit numbers will have a apostrophe in the right character. This number identifies different versions of files with identical file identifiers and set identifications (fields 3 and 4) and each version must be treated as an individual file.

\$ field 9 and 10:

- Creation and expiration date are not used and will contain spaces.

 SECOND FILE HEADER LABEL

POSITION	FIELD	NAME	LENGTH	CONTENTS
1- 3	1	label identifier	3	HDR
4	2	label number	1	2
5	3	record format	1	U
6-10	4	block length	5	(no of characters)
11-15	5	record length	5	(spaces)
16-50	6	res. for operating systems	35	(name of owner \$ & MAX BYTE POINTER)
51-52	7	(not used)	2	(spaces)
53-80	8	(not used)	28	(spaces)

\$ field 6:

- Up to 16 alphanumeric characters starting from position 16 identifying owner of this file.
If name is shorter than 16 characters, apostrophe is used to mark end of name.
- 32-41 max byte pointer of file.

 END OF FILE LABEL

POSITION	FIELD	NAME	LENGTH	CONTENTS
1- 3	1	label identifier	3	EOF
4	2	label number	1	1
5-54	3-11	(same as HDR1)	50	(corresponds HDR1)
55-60	12	block count	6	(number of blocks)
61-80	13-14	(not used)	20	(spaces)

END OF VOLUME LABEL

POSITION	FIELD	NAME	LENGTH	CONTENTS
1-3	1	label identifier	3	EOV
4-80	2-14	same as EOF1	77	(corresponds EOF1)

USER LABEL

POSITION	FIELD	NAME	LENGTH	CONTENTS
1-3	1	label identifier	3	UHL
4	2	label number	1	1
5-80	3	user option	76	(file information) \$

Explanation of field 3

\$\$ This field differs from the ANSI label standard. The field contains binary information for the ND subsystem: BACKUP-SYSTEM SINTRAN III.

Field 3:

POSITION		CONTENTS
WITHIN FIELD	WITHIN LABEL	
1-2	5-6	version number of this file (1-255 dec.)
3-4	7-8	total number of versions (1-255 dec.)
5-8	9-12	filesystem-standard creation-date
13-76	17-80	S-III filesystem object entry

 NON STANDARD 'HOLE' LABEL

POSITION	FIELD	NAME	LENGTH	CONTENTS
1-3	1	label identifier	3	HOL
4	2	label number	1	E
5-80	3	user option	76	(information) \$

Explanation of field 3:

Field 3:

POSITION		CONTENTS
WITHIN FIELD	WITHIN LABEL	
1-72	5-76	THIS BLOCK IS NOT PART OF THE DATA!
		CHARACTERS 77-80 CONTAIN A NUMBER.
73-76	77-80	(32-bit binary number stating the logical block number of the following data block).

6 LOOK-FILE

6.1 INTRODUCTION

LOOK-FILE is a utility system which enables a user to print data, modify data and browse through the data contained in a file. The DUMP commands allow a variety of format for output on a terminal or a printing device.

6.2 COMMANDS — SUMMARY

The available commands with their required parameters are:

HELP	<command-name or CR>
EXIT	
CALCULATE	
DUMP-WORDS	<block number><word number><length>
DUMP-BYTES	<block number><word number><length>
DUMP-ALL	<block number><word number><length>
PATCH	<block number><word number>
OPEN-FILE	<file name><block size>
ZERO	<block number>
MOVE-BLOCK	<from file name><number of blocks> <from block number><to block number>
CLOSE-FILE	
BACK	
FORWARD	
COMPARE BLOCK	<compare file name><number of blocks> <from block number>
LIST-FILE	<listing file name>
ON-LIST	
OFF-LIST	
SEARCH	<from block number><number of blocks>

6.3 *COMMANDS — GENERAL RULES*

The system may be entered by typing

ⓈLOOK-FILE

All the commands may be abbreviated in the same way allowed by SINTRAN. All parameters may be entered on the same line as the command, or they may be omitted and the system will prompt the user for each command in the required order.

Parameters which require a numeric value may be entered as decimal numbers, eg. 123, or as octal values which must be followed by the letter B, eg. 123B.

The OPEN-FILE command must be used to open a file before it is referred to by any of the other commands.

The format used for printing information includes:

- 1) The word number in decimal.
- 2) The word number in octal.
- 3) A single character indicating the mode being used for the current line, ie. B for Byte and W for Word.
- 4) 5 16-bit words output in the mode being used.
- 5) The 5 words output as ASCII characters.

Note: any characters whose octal value is less than 40B will be output as an amperand (&). The DUMP-ALL command will also output the ASCII character for values less than 40B.

6.4 *COMMANDS — DETAILED DESCRIPTION*

HELP

List one or all command(s) with the required parameters.

EXIT

Leave the system and return to SINTRAN. Note that all files will be closed.

OPEN-FILE

Opens a 'global' file which will be used for further operations by other commands. Other commands will be opened/closed automatically by the specific command being used. The default block size is 512 16-bit words. The maximum allowed block size 2048 16-bit words.

Note: The current global file is closed before the OPEN-FILE command is executed. The file is opened for WRITE access.

CLOSE

Close all currently open files.

CALCULATE

Simple calculations may be performed on decimal or octal values. The resulting value will be displayed. The operators available are:

- + for addition
- for subtraction
- * for multiplication
- / for division

DUMP-WORDS, DUMP-BYTES, DUMP-ALL

Display a block of data from the currently opened global file on the user terminal or, optionally on a LIST-FILE. The display format depends on which command is being used. The length of data requested must not be longer than the value given in the OPEN-FILE command for this file.

If the length requested is less than the file's block size, then any DUMP command will display only the number of words requested. If a subsequent FORWARD/BACK command is used, then it will move the number of words in the file's block, and thus some of the block referred by the DUMP command will *not* be displayed.

Note: 1. The block numbers begin at 0.
2. The word numbers begin at 1.

BACK, FORWARD

These commands should be used together with the DUMP commands to move forwards or backwards from the current block number in the current file. These commands change the current block number.

PATCH

This command allows modification of any word in the current block of data. Patch will modify successive words in the block of data until a full stop character (.) is typed. New value can be given as decimal integer (f.ex. 1) or octal integer (f.ex. 177777B) or a character string (f.ex. 'EX')

Note: All bits in a word are modified, including the left-most bit which is sometimes used for parity.

ZERO

Clear to binary zeroes the complete block specified on the current global file. The block size given in the OPEN-FILE command for this file is used.

MOVE-BLOCK

Move the first one or more blocks from the named file to the current global file. The block size of the current global file will be used.

COMPARE-BLOCK

Compare the entire named file with the current global file. Any data blocks which are not identical on both files are printed. The compare file is opened/closed automatically.

LIST-FILE

The named file is opened for printed output from any DUMP, COMPARE-BLOCK or SEARCH commands. This file will remain open until a CLOSE or EXIT command is used. The ON-LIST/OFF-LIST commands may be used to optionally print some data blocks.

ON-LIST

Switch print for the LIST-FILE on.

OFF-LIST

Switch print for the LIST-FILE off.

SEARCH

This command will search for the first occurrence of a string of words in a number of data blocks. IF the string being searched for is found, then the data block containing it is displayed, and printed if a LIST-FILE is switched on. After the data block has been displayed, typing 'NO' will stop further searching. Otherwise, the search will continue looking for the next occurrence.

7 NORD FILE EXTRACT UTILITY COMMAND

7.1 *INTRODUCTION*

File extract is a general purpose UTILITY program which can extract records from one file and write onto another file or output device.

In addition, by using the split option, records not satisfying given extract selection criteria can be placed in a second output file, thus providing a complete file split possibility.

The program provides for complex record selections invoked by simple parameters. The user may define his output record layout in several ways. Also, a wide range of output environment choices are available.

These facilities make the program useful in various data processing situations. It's use may range from very simple runs to rather sophisticated processing.

FILE-EXTRACT is written in FORTRAN. It handles standard SINTRAN III text files, including variable record length files. Maximum record size is set to 1024 bytes.

7.1.1 *Purpose*

FILE—EXTRACT is a utility enabling ND users to process files without writing specific programs. This sort of file processing may be relevant during program development, testing or simply validation and correction of data files.

FILE—EXTRACT contains facilities such as:

- the extraction of subsets from files based on record numbering
- the extraction of subsets from the files based on individual record contents
- the rearranging of files
- the appending of files or subsets of files to other files
- file splitting by one run
- reformatting of files according to record layout, length and organization
- providing output records containing input record number
- providing output records containing the master record's physical address (see Section 7.2.4.4)
- conversion of transactions from various systems to a common layout
- generation of readable reports containing heading and page numbering routed to a terminal or a line printer
- saving of parameter input in MODE files for later automatic processing (see Section 7.2.1.1)
- building or procedures to be processed with limited run-time parameter input (see Section 7.2.1.2)

These facilities may be combined in various ways thus meeting new demands as they occur.

7.2 COMMAND STRUCTURE

FILE—EXTRACT may be called from a terminal when in SINTRAN III command mode.

Command Structure:

@FILE EXTRACT

-- NORD FILE EXTRACT UTILITY COMMAND, VER. DD MM YY --

INPUT FILE: <\$MODE> <\$AUTO> <\$KEY> <,Fnnn>

OUTPUT FILE: _____ <,X> <,A> <:>

<SPLIT OPTION OUTPUT FILE 2: _____ <,A> >

EXTRACT SPECIFICATIONS:

<<SHOW> <extract selection criteria> <:> >

< _____ >

OUTPUT RECORD LAYOUT SPECIFICATIONS:

<<SHOW> <Wnn> <L> <LO> <Hnn> <PAGE[="xxxx"]> <R> <E>

<P> <C> <T> <record layout> <:> >

< _____ >

INPUT RECORDS: 99999, OUTPUT RECORDS: 99999 I = = = > ----*----I

The program will request input from the user as shown above.

All input fields, except for INPUT-FILE, accept default values. Thus, a "default run" will cause the input file to be listed on the terminal.

The default value is indicated by typing CARRIAGE RETURN in the specific input field.

However, the command structure is made in such a way that the required options may be activated by use of simple parameters. Any other functions are automatically avoided.

7.2.1 Input File

The input file may be specified as any randomly accessible SINTRAN III text file. A default file type <:SYMB> is assumed when type is not specified. The file is immediately checked for legal access. If not obtained, an error message will be written to the terminal before program termination.

7.2.1.1 Mode File Save Option

The mode file save option may be invoked by typing `<$MODE>` in response to the input file question. The following text will be written on the terminal:

MODE SAVE FILE:

In the file specified in answer to this question, all COMMAND INPUT will be saved as a SINTRAN III MODE file. In this way, specifications given for an extract run may be saved for later automatic processing, thus enabling the user to generate procedures under the guidance of the program.

7.2.1.2 Limited Automatic Command Input

The LIMITED AUTOMATIC COMMAND INPUT option may be invoked by typing `<$AUTO>` in response to INPUT FILE. The program will immediately ask for:

AUTO RUN-TIME COMMAND FILE:

and then read the command input lines from the file specified here. This facility is quite similar to the execution of FILE-EXTRACT from a MODE file. The difference is that a command line in the AUTO RUN-TIME COMMAND FILE may contain the text \$TERM, meaning that this line is to be prompted from the terminal.

This option is very useful for complex predefined procedures, where some features are to be requested at run-time. An example could be a pregenerated report procedure where the user is to specify, at run-time, the output device as terminal or line printer, or perhaps some additional extract selection criteria to be read in. All other parameters and the report layout will automatically be read from the command file.

Such a command file may be generated by the MODE FILE SAVE OPTION (see Section 7.2.1.1) and then edited by QED or PED. Remember to remove tabs when in QED (command M TO(0)).

7.2.1.3 Fixed Record Length Input File Option

To process a fixed record length input file not containing record delimiting characters (octal 015, 012, i.e., CR, LF), the F option must be used. The parameter should follow input file name and be specified as follows:

< ,Fnnnn >

where nnnn specifies input file record length in bytes (maximum 1024 bytes).

Note that the output file, as a rule, will receive/have the same organization as the input file.

The following conditions will, however, make a sequential output file out of a "fixed" input file:

- output file organization change option specified (see Section 7.2.2.3)
- terminal output wait option specified (see Section 7.2.4.6)
- line printer/terminal heading option specified (see Sections 7.2.4.7, 7.2.4.8, 7.2.4.9 and 7.2.4.10)

7.2.1.4 Indexed Access via KEY file

Indexed access via KEY file is initiated by typing <\$KEY> in response to the input file question. The program will then ask for:

KEY FILE NAME:

The KEY file is only supposed to indicate which records of the input file are to be read and in which order. The KEY file must be a symbolic file, each record starting with a pointer to a corresponding record within the main input file. Any trailing contents of a KEY file record will be ignored by FILE-EXTRACT. A KEY file will normally be output of a FILE-EXTRACT run using the "Random Key Inclusion Option" and must follow the format used here (see Section 7.2.4.5). The file could then be sorted or processed in any way before being utilized as KEY file.

For situations which could benefit from this option, see examples mentioned in Section 7.2.4.5.

7.2.2 *Output File*

Output file may be any existing/nonexistent SINTRAN III disk file or an output device such as line printer or terminal.

The file name is specified due to the standard SINTRAN syntax. That is, nonexistent files must be enclosed by double quotes, etc.

Note that random write is always used unless output file TERM (terminal) is selected or the WAIT option (see Section 7.2.4.6) is switched on. So, when writing to any other sequential only output device, a dummy WAIT option must be used.

Default output file is the terminal.

7.2.2.1 Output File Append Option

The parameter <,A> following output file name, invokes the output file append option. This means that the output will be appended at the end of the given file.

Note that this option requires an existing output file and is not valid for such output devices as terminal or line printer.

7.2.2.2 File Split Option

A <:> at the end of the output file input line invokes the file split option. The following test will be written to the terminal:

SPLIT OPTION OUTPUT FILE:

Records read, but not qualifying to be written to the main output file according to the extract selection criteria given (see Section 7.2.3) will now be written to the SPLIT OPTION OUTPUT FILE. If this option is not specified, those records will simply be bypassed by FILE-EXTRACT.

The append option <,A> is also available for the split file (see Section 7.2.2.1).

7.2.2.3 *Output File Organization Change (X Option)*

The X option is used to switch the output file organization, thus making a sequential file containing end of record characters out of a random, fixed length record file and vice versa.

Consider a sequential, variable record length input file. By using the X option, a random, fixed length record output file will be produced. The output record length will automatically be computed from the output record layout specifications given (see Section 7.2.4). Note that X option switch to random file organization will be ignored when used together with certain other options (see Section 7.2.1.3).

Sequential records, delimited by End of Record characters will be produced when the X option is specified in conjunction with the fixed record length input file option (see Section 7.2.1.3).

Output file organization change may be useful in several situations. Consider a fixed length random data file needing some special editing. The X option can produce a QED or PED recognisable version of the file, which could then be edited and finally reconverted to its original organization using the X option once again.

7.2.3 *Extract Selection Specifications*

One or two input lines are available for extract selection specifications. The commands given here determine which records are to be written to the output file.

There are four types of selections available:

- specification of input file record intervals in question (see Section 7.2.3.7)
- specification of input record field values to be satisfied/not satisfied (see Sections 7.2.3.1 and 7.2.3.2)
- specification at text strings which are to occur/not occur within a record (see Section 7.2.3.3)
- specification of a text string which is to occur/not occur within a specified subset of a record (see Section 7.2.3.4)

The selection criteria specified may be connected by the logical operands `<.AND.>` and `<.OR.>` (see Section 7.2.3.5).

Finally, parentheses nesting on groups of selection criteria are allowed (see Section 7.2.3.6).

Together, these options provide a sophisticated data selection tool that may be used for the diverse tasks.

Note that extract criteria, logical operands, values and parentheses must not be separated by spaces. Spaces are treated as command line terminators.

7.2.3.1 Numeric Field Evaluation

A numeric field evaluation criterion is to be specified in the following manner:

<STARTPOS> [—ENDPOS] <operation code> <MIN VALUE>
[—MAX VALUE]

where

STARTPOS

is the start byte number of numeric field within input record.

ENDPOS

End byte number of numeric field within input record. May be omitted for 1 digit fields.

OPERATION CODE

One of the following operation codes must be specified:

=	equal to
≠	not equal to
>	greater than
<	less than

MIN VALUE

is the numeric value for operation codes =, ≠ or the value to compare with the codes < and >.

MAX VALUE

is the maximum value that may be specified for operation codes = or ≠. It then specifies the upper numeric limit for a range specification, thus providing the additional operation codes "inbetween" and "not inbetween".

Example:

15 — 18 = 1590 — 1862

This means that if this particular extract selection criterion is to be satisfied, byte 15 through 18, within an input record, must contain a numeric value within the range 1590 to 8262.

7.2.3.2 Text Field Evaluation

A text field evaluation criterion is specified as follows:

<STARTPOS> [—ENDPOS] <operation code> <"text string">

where:

STARTPOS

is the start byte number within input record to be evaluated.

ENDPOS

is the end byte number within input record to be evaluated. May be omitted for one byte field.

OPERATION CODE

The two following operation codes are allowed:

= equal to
 ≠ unequal to

TEXT STRING

The text string may contain any character and must be surrounded by double quotes.

Note that the length of the text string must be the same as the field length specified by the STARTPOS/ENDPOS elements.

If shorter, a limited text string search will be assumed (refer to Section 7.2.3.4).

If longer, the specification will not be accepted and the program terminated with an error message.

Example:

45 — 50 = "OSLO 5"

7.2.3.3 Text String Search

A text string search specification will cause the entire input record to be scanned for the existence of the given text string.

A text string search is specified as follows:

TEXT <operation code> <"text string">

where:

TEXT

specifies search within the entire record.

OPERATION CODE

The two following operation codes are allowed:

= equal to
≠ unequal to

TEXT STRING

Any text enclosed by double quotes may be specified.

Example:

TEXT = "COMMUNICATION"

7.2.3.4 Limited Text String Search

A limited text string search will cause the specified subset of the input record to be scanned for the existence of the given text string.

Syntax:

<STARTPOS> <-ENDPOS> <operation code> <"text string">

where:

STARTPOS

is the start byte number within input record where the text search is to be done.

ENDPOS

is the end byte number limiting search area within input record.

operation code

The two following operation codes are allowed:

= equal to
 ≠ unequal to

text string

The search text string may contain any characters (except double quote) and must be enclosed by double quotes.

Note: the length of the text string must be less than the record subset specified by startpos/endpos.

Example:

45 — 90 = "BOX"

This may extract those customer records having a P.O. Box address within the address fields subset of the record.

7.2.3.5 Logical Operands

A logical operand is used to connect two extract selection criteria of any kind.

Together with the parentheses nesting (see Section 7.2.3.6) this facility enables complex extract selections to be made.

Syntax:

<extract criterion A> <logical operand> <extract criterion B>

where:

extract criterion A and B

is the same as Sections 7.2.3.1, 7.2.3.2, 7.2.3.3 or 7.2.3.4 except for the input file record interval option as in Section 7.2.3.7.

logical operand

The two following operands are allowed:

.AND.	both criterion A and criterion B must be fulfilled
.OR.	either criterion A or B must be fulfilled

Example:

15 — 18 = 1590 — 8260 .OR. 45 — 50 = "OSLO 5"

7.2.3.6 Parentheses Nesting

Parentheses nesting is available for expressing more complex selections.

Extract criteria/groups of extract criteria connected with logical operands may be surrounded by parentheses/levels of parentheses.

Example:

```
((1 — 2 = "T1" .OR. 1 — 2 = "T2") .AND. 10 = 2) .AND. (15 — 22 > 90000
.OR. 23 = "**")
```

This could mean something like "select those records of type T1 or T2 having status code 2 and either have a balance over 90,000 or are marked with a start in position 23".

Rules:

A start parenthesis must be placed before an extract criterion or together with another start parenthesis.

An end parenthesis must be placed after an extract criterion or together with another end parenthesis.

7.2.3.7 Input File Record Intervals

By specifying input file record intervals, one may select subsets of the input file to be evaluated.

Also, this option provides a file rearranging possibility due to the fact that the program will process input file records in the same order as indicated in the command line.

If a record interval is followed by another one specifying records already bypassed, the input file will be rewound before those records are processed.

Syntax:

<start record no.> — <end record no.> ,

where:

record no.

Record no. is specified with 1 to 9 digits

—

is start/end delimiter

,

is interval terminator. May be followed by parentheses or any other extract selection criterion including another input file record interval specification.

Note:

When record intervals are used to rearrange a file and the file split option is active (see Section 7.2.2.2) split file records will be duplicated every time the input file is rewound.

7.2.3.8 Show First Input File Record Option

Typing "SHOW" and the RETURN button at the beginning of the command line, the first input file record will be written to the terminal together with a position mask line such as:

```
123456789.123456789.123456789.123456789.1234....
7205PETTERSEN,PER    OSLO 5    223652    80000
```

This information is meant to be of assistance to the operator to see the position number for the different fields to be made extract selections from and has nothing to do with the actual output from the run.

The program will immediately accept input of extract selection specifications.

Note:

By typing another SHOW, the next record will be shown, thus providing selection of a record type layout representative record.

7.2.3.9 Command Line Continuation Option

Terminating the first command line with a <:> will provide another line for extract selection input.

Note:

Used together with the limited automatic command input (see Section 7.2.1.2) the first line may be specified beforehand, while the second may be used for additional operator selections at run-time.

7.2.4 *Output Specifications*

One or two input lines are available for various output specifications. A number of parameters are available to specify how records selected by the extract specifications are to be written (refer also to Section 7.2.3).

There are two main types of specifications available:

1. Specification of output record layout as one or more of the following elements:
 - a copy of input record
 - subsets of input record
 - imbedded constants
 - input record number
 - output record number
 - input record random address
2. Specification of output environment such as:
 - terminal output wait at full screen option
 - line printer/terminal heading specification
 - line printer/terminal predefined headings
 - page numbering
 - split file record as a copy of input record in spite of output specifications

Default (CR) makes the output record a copy of input record.

7.2.4.1 Input Record Subsets Specification

Subsets of input record can be specified to build the output record or to be a part of it.

Syntax:

<start position> [—end position] [,]

where:

start position

starts the position within input record to be copied to the output record.

end position

ends the position within input record to be copied. May be omitted when only one character is to be copied.

is specification delimiter in case of more specifications.

Example:

50 — 55, 1 — 20

This will produce an output record containing position 50 through 55 and finally the first 20 characters of the input record.

Note:

When the output record is specified to contain subsets of the input record, input records shorter than the subsets specified will result in an output record filled with spaces as a substitution for the missing input characters.

As a result, this facility can provide a file reformatting possibility, e.g., produce a fixed record length file out of a variable length one.

7.2.4.2 Output Record Constants

Constants may be imbedded in any position of output record.

Syntax:

"text" [,]

where:

text

may be any character except for double quotes.

[,]

is used as delimiter in case of more specifications.

Example:

50 — 55, "ABC", 1 — 26

This will insert the string "ABC" within the input record subsets specified.

7.2.4.3 Input Record Number Inclusion

The input record number may be specified to be the first element of the output record.

Syntax:

<L> [,]

The command will result in a 5 digit line number indicating source record number of input file.

Note: It cannot be used together with the <LO> or <R> options.

7.2.4.6 Terminal Output Wait Option

The WAIT option is intended to be used with the terminal as output file. It simply makes the program wait for an input character for every given number of lines written to the terminal, thus enabling the user to study one screen of information before filling the next one.

The user may, at this point, interrupt the extract run by typing an X (exit). Any other character, including carriage return, will make the process continue.

Syntax:

W [nn] [,]

where:

nn

is a number of lines to be written before waiting for carriage return. Default value is 24 for standard VDU screens.

is specification delimiter in case of more parameters.

7.2.4.7 Line Printer/Terminal Output Heading Option

The heading option enables the output from FILE-EXTRACT to be generated as simple reports with a one line heading, optionally together with page number (see also Section 7.2.4.8).

Syntax:

H [nn] [,]

where:

nn

is the number of lines per page. Default value is 24 (VDU terminal).

is parameter delimiter.

Note:

A common line counter is used for the heading and wait options. Therefore, if in doubt, the last line numbering specified in the command line will be used.

When all output specifications are given and the heading option is specified, the program will write a heading mask to the terminal and wait for user input:

```
HEADING MASK:
123456      123456      123456789.123456789.
KUNDENR.  KONTONR.     N A V N
```

The first two lines above are produced by the computer. It simply represents a position mask of the output record, dimensioning the input record subsets chosen in the output specifications, corrected with constants if any. This mask indicates where to type the leading text in order to produce a readable report. Used together with the show option (see Section 7.2.4.12), the heading should have all changes to be correctly specified.

7.2.4.8 Line Printer/Terminal Page Numbering Option

The page numbering option will provide a page number to be written before each heading. The parameter will have no effect when the heading option is not specified.

Syntax:

PAGE [= "page text"] [,]

where:

PAGE

This text which will invoke the option.

page text

The user may define his own 6 character long page text in his own language. Default text is "PAGE".

Example:

PAGE = "SIDE:"

This will, when used together with the heading option for each page, produce a heading such as:

SIDE: 9999

```
HEADING LINE . . . . .
DETAIL OUTPUT LINE 1  * * * * *
DETAIL OUTPUT LINE 2  * * * * *
* * * * *
* * * * *
* * * * *
```

7.2.4.9 Predefined Heading as Extract Command Line

In some cases, it may be useful to have the extract selection specifications written together with the output. This is provided by the E option, which will automatically produce the extract command line as the heading line.

Syntax:

E [nn] [.,]

The option works exactly like the H option (see Section 7.2.4.7) except it doesn't ask for heading input. Besides, the page numbering option (see Section 7.2.4.8) will automatically be invoked.

7.2.4.10 Predefined Heading as Position Mask

The P option produces a position mask as a predefined heading. This may be useful when record contents are to be studied in their original compressed format.

Syntax:

P [nn] [.,]

This option is similar to the E option (see Section 7.2.4.9).

7.2.4.11 Split File Copy Option

Normally, the split file output (see Section 7.2.2.2) will contain record layout similar to the main output (no page numbering and no headings). In some cases, it may be useful to provide a split file containing records as a copy of the input records. Thus, the C option will turn off any other output record layout specifications on split file writes.

Syntax:

C [.,]

7.2.4.12 Show First Input File Record Option

The "SHOW" option is also provided as a first command to this output specifications input line. It works exactly in the same way as described above (see Section 7.2.3.8). In this case it is meant as a tool to produce an output record from the right subsets of the input record and also to help design the heading line.

Syntax:

SHOW

7.2.4.13 Command Line Continuation Option

Terminating the first command line with a <:> will provide another line for output specification input.

Note:

Used together with the limited automatic command input (see Section 7.2.1.2), the first line may be specified previously while the second one may be used for additional operator's choice at run-time.

7.2.4.14 Skip Output Record Trailing Spaces

In order to reduce disk space and increase processing speed, skipping trailing spaces may be desired. The option is supposed to be used in conjunction with variable record length output files.

Syntax:

T [,]

7.3 *RUN-TIME STATUS MESSAGES*

In order to enable the user to keep track of the program's progress, a run-time status message line is implemented:

INPUT RECORDS: 99999, OUTPUT RECORDS: 99999 | = = > ----*-----|

For every 100 input records processed, this line will be written to the terminal. The right side graph indicates the percentage (in bytes) of the input file being processed, thus enabling the user to estimate when the process will be finished.

***** **SEND US YOUR COMMENTS!!!** *****

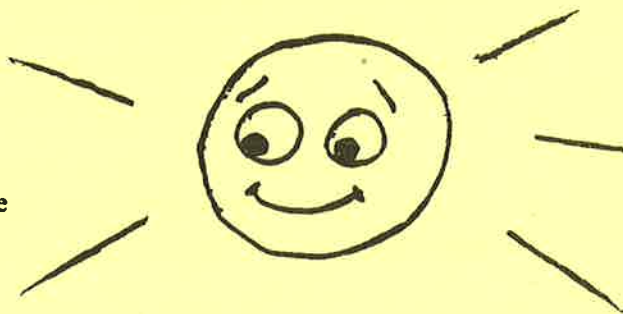


Are you frustrated because of unclear information in this manual? Do you have trouble finding things? Why don't you join the Reader's Club and send us a note? You will receive a membership card - and an answer to your comments.

Please let us know if you

- * find errors
- * cannot understand information
- * cannot find information
- * find needless information

Do you think we could improve the manual by rearranging the contents? You could also tell us if you like the manual!!



***** **HELP YOURSELF BY HELPING US!!** *****

Manual name: SINTAN III Utilities Manual

Manual number: ND - 60.151.01

What problems do you have? (use extra pages if needed)

Do you have suggestions for improving this manual?

Your name: _____ Date: _____

Company: _____ Position: _____

Address: _____

What are you using this manual for? _____

Send to: Norsk Data A.S.
Documentation Department
P.O. Box 4, Lindeberg Gård
Oslo 10, Norway

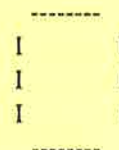


Norsk Data's answer will be found on reverse side

Answer from Norsk Data

Answered by _____ Date _____

Date _____



Documentation Department

Oslo 10, Norway