SINTRAN III
Utilities Manual

ND-60.151.02
Rev. A

SINTRAN III
Utilities Manual

ND-60.151.02
Rev. A

NOTICE

PRINTING RECORD

Printing	Notes
11/81	Version 01
05/82	Revision A
	The following pages are revised or new: vi, vii, viii, 1–1, 3–21.
	Sections 6 and 7.
12/82	Revision B
	The following pages are revised: vii.
	Section 5.
06/84	Version 02
02/85	Revision A
	The following sections are new: Sections 3 and 4.

SINTRAN III Utilities Manual
Publ.No. ND--60.151.02A

Norsk Data

Norsk Data A.S
Graphic Center
P.O.Box 25, Bogerud
0621 Oslo 6, Norway

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 Customer Support Information (CSI) and can be ordered as described below.

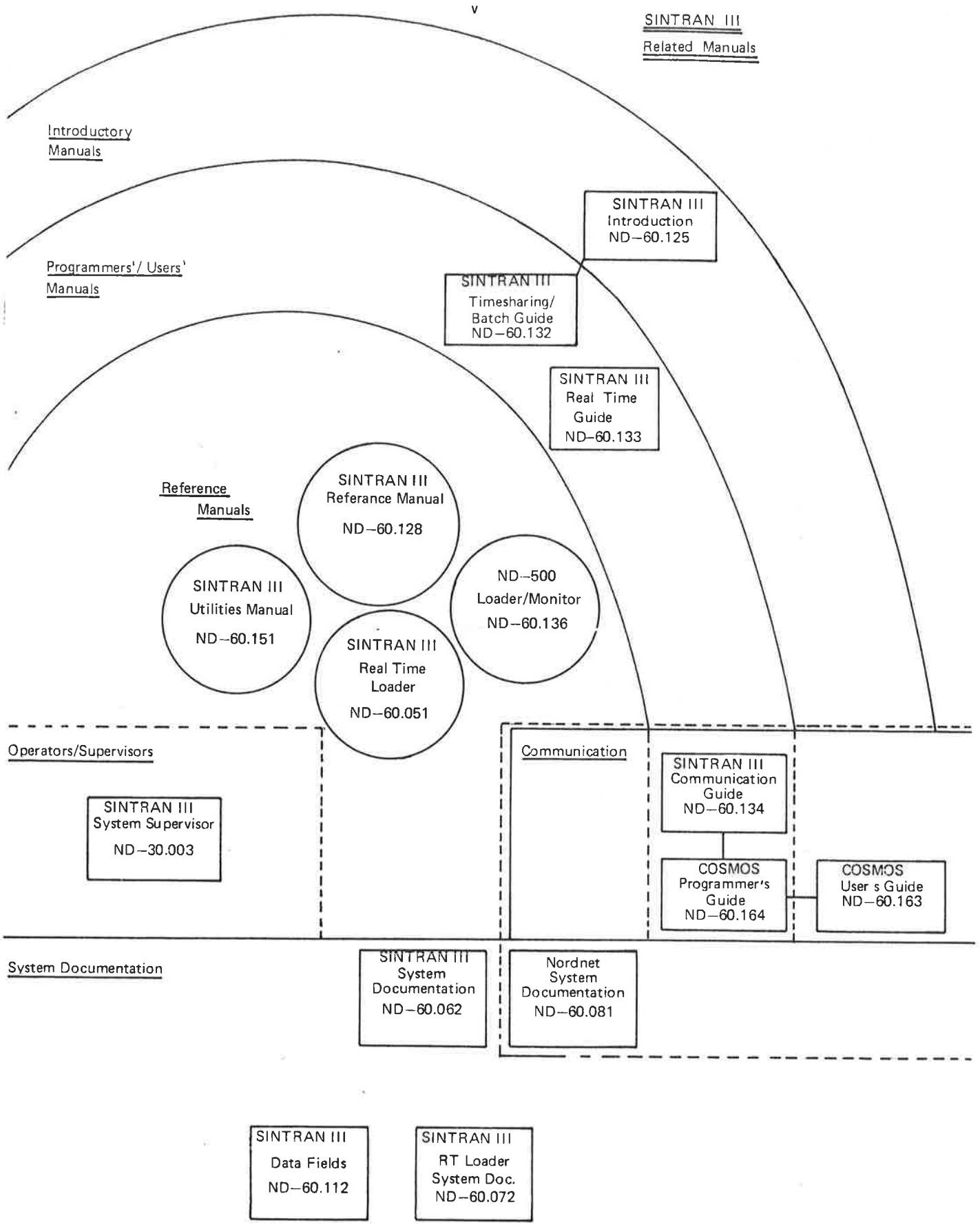
The reader's comments form at the back of this manual can be used both to report errors in the manual and to give an evaluation of the manual. Both detailed and general comments are welcome.

These forms and comments should be sent to:

Documentation Department
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Requests for documentation should be sent to the local ND office or (in Norway) to:

Graphic Center
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway


SINTRAN III
Related Manuals
Introductory
Manuals
SINTRAN III
Introduction
ND–60.125
Programmers'/ Users'
Manuals
SINTRAN III
Timesharing/
Batch Guide
ND–60.132
SINTRAN III
Real Time
Guide
ND–60.133
Reference
Manuals
SINTRAN III
Referance Manual
ND–60.128
SINTRAN III
Utilities Manual
ND–60.151
ND–500
Loader/Monitor
ND–60.136
SINTRAN III
Real Time
Loader
ND–60.051
Operators/Supervisors
SINTRAN III
System Supervisor
ND–30.003
Communication
SINTRAN III
Communication
Guide
ND–60.134
COSMOS
Programmer's
Guide
ND–60.164
COSMOS
User s Guide
ND–60.163
System Documentation
SINTRAN III
System
Documentation
ND–60.062
Nordnet
System
Documentation
ND–60.081
SINTRAN III
Data Fields
ND–60.112
SINTRAN III
RT Loader
System Doc.
ND–60.072

PREFACE

THE PRODUCTS

This manual describes subsystems which run under the SINTRAN III operating system. These subsystems and their product numbers are:

GPM	ND—10124
PERFORM	ND—10022
BACKUP—SYSTEM	ND—10337
LOOK—FILE	ND—10005
FILE EXTRACT UTILITY	ND—10044
JEC	ND—10005
MAIL	(Integrated part of SINTRAN III)
VTM-COMPOUND	ND—10599

MAIL is used to send messages to other users. PERFORM is a simple macro processing system to create mode and batch files. GPM is a general purpose macro generator. The BACKUP-SYSTEM is used to copy files efficiently. LOOK-FILE is used to inspect and modify files. FILE-EXTRACT can be used to extract records from files. JEC is used to control execution of batch and mode jobs. VTM-COMPOUND is used to compound new terminal type descriptions into one terminal type table used by the VIRTUAL TERMINAL MANAGER (VTM).

THE READER

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

PREREQUISITE KNOWLEDGE

Familiarity with SINTRAN III at the public user level is necessary.

THE MANUAL

This manual describes some subsystems under SINTRAN III. The subsystems are not necessary for simple use of SINTRAN III, but may be of considerable use for particular tasks. The manual is mainly a reference manual.

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.

The ND GLOSSARY (ND-40.005) will explain common computer terms and what they mean in ND manuals. ND abbreviations and acronyms are also listed. The glossary should be of interest to anyone using ND equipment.

NOTATION USED IN THE MANUAL

In the examples, user input is underlined. Examples are given in UPPERCASE letters, but lowercase letters are also accepted. When used as parameters, octal numbers are given in the form 377B, where the B denotes octal. In command parameter descriptions, the parameters are enclosed in angular brackets, eg., <parameter>.

Parameters which have default values are enclosed in parentheses, eg., (<parameter>). The default value is used if a null parameter is supplied. Selections in parameter descriptions are separated by slashes, eg., YES/NO.

CHANGES FROM PREVIOUS VERSION

JEC and VTM-COMPOUND are new products. The structure of the chapters which describe PERFORM, BACKUP-SYSTEM, LOOK-FILE and MAIL has been changed. New features in the subsystems are marked with a vertical line.

TABLE OF CONTENTS

+ + +

1 MAIL

MAIL is a subsystem for sending messages to other users. Messages can be sent directly to the terminal of any user who is logged in. The message displayed will not interfere with the work being done at that terminal.

The subsystem operates like a mailbox for users who are not logged in. They will be told that they have mail when they log in. They can read the message sent to them by entering MAIL.

User SYSTEM is allowed to send the same messages to all users or terminals. This is called broadcasting. Some of the MAIL commands are only available to user SYSTEM.

1.1 GENERAL DESCRIPTION

MAIL must be entered both to send messages and to receive messsages that are stored in the mailbox. The subsystem is entered by

@MAIL (<output file>)

The parameter <output file> describes where you want the contents of your mailbox to be written. It will only be requested if you have mail. The default <output file> is your terminal.

MAIL prints an asterisk (*) when it is waiting for you to give a command. The HELP command will display the available commands. You return to SINTRAN III by the EXIT command. MAIL commands can be entered and abbreviated as SINTRAN III commands. If you omit parameters, they will be prompted. Only one user at a time may use MAIL.

1.2 COMMANDS AVAILABLE TO ALL USERS

This section describes the commands available to all users. When sending messages or broadcasts, the message must be terminated by CTRL L. A dollar sign ($) in a message will start a new output line. Messages will be output together with the name of the sender. The maximum message length is 512 characters. All messages will be converted to upper case letters.

EXIT

This command leaves the MAIL subsystem and returns you to SINTRAN III.

HELP

This command lists the available commands.

SEND-DIRECT-MESSAGE <to terminal number>

This command is used to send a message that will be displayed immediately on the terminal specified. The SINTRAN III command @WHO-IS-ON will list the terminal numbers of the users who are logged in. You will be asked to type your message.

SEND-MESSAGE <to user>

This command is used to send a message that will be stored in the mailbox. It can be sent to any user regardless of whether the user is logged in or not. The user will be told that she/he has mail the next time she/he logs in or out. The parameter <to user> is the user name of the receiver. You will be asked to type your message.

LIST-MESSAGE (<output file>)

This command will list the messages in the mailbox on the specified <output file>. The default <output file> is your terminal. The mail index number is used if you want to delete a message.

DELETE-MESSAGE <mail index>

The command will delete a message in the mailbox. The command LIST-MESSAGE can be used to find the <mail index>. Only user SYSTEM is allowed to delete messages sent by other users.

FINISH

This command returns you to SINTRAN III in the same way as EXIT.

1.3 COMMANDS AVAILABLE TO USER SYSTEM

This section describes the commands only available to user SYSTEM. The protected commands are used to broadcast messages and to start and stop MAIL.

DIRECT-BROADCAST

You will be asked to type your message. The message will be displayed immediately on all terminals.

BROADCAST

You will be asked to type your message. The message will be sent to the mailbox of all users.

LIST-BROADCASTS (<output file>)

The command will output all broadcasts and their mail index numbers. The default <output file> is your terminal.

DELETE-BROADCAST <mail index>

The command will delete a broadcast in the mailbox. The <mail index> is found by the command LIST-BROADCASTS.

INITIALIZE <maximum number of messages>

The command must be given before MAIL can be used. It defines the maximum number of messages that can be stored in the mailbox. The command can also be used to delete the contents of the mailbox. The mail is stored on the file (SYSTEM)MAILBOX:DATA.

RUN-MAIL-SYSTEM

The command starts MAIL after starting SINTRAN III or after stopping MAIL by the command STOP-MAIL-SYSTEM. The contents of the mailbox are retained.

STOP-MAIL-SYSTEM

The command makes MAIL unavailable. The contents of the mailbox will not be lost. The mail is stored on the file (SYSTEM)MAILBOX:DATA.

1.4 SENDING MESSAGES FROM MODE AND BATCH JOBS

When MAIL is used in mode and batch jobs, the commands should be preceded by a @. A command and its parameters should be entered on one line. Messages should be entered on a separate line. They have to be terminated by CTRL O CTRL L.

The following mode file uses MAIL:

```
@MAIL
@SEND-MESSAGE P-HANSEN
THIS IS A TEST <CTRL O><CTRL L>
@EXIT
```

The CTRL O CTRL L will normally be displayed as an ampersand (&).

2 PERFORM

Mode or batch files are used to execute sequences of commands that are used repeatedly. The advantage of PERFORM is the flexibility of parameter substitution in such mode and batch files.

For example, mode files can be used to compile, load, and execute programs during development. However, each program needs a separate mode file. PERFORM will instead allow you to enter the program name as a parameter and generate the required mode file with this program name in the appropriate places.

To use PERFORM, you have to create a macro instead of a mode file. The macro allows you to specify which parameters are to be entered from the terminal at each execution. PERFORM will merge the macro with the terminal input, and create a mode file.

Macros are created using an ordinary editor, and many macros can be stored on a file. A predefined library of macros is stored on the file PERFORM-LIB:MCRO.

2.1 CREATING MACROS

A few simple directives, starting with a circumflex (^), are used to define a macro. All directives terminate by a semicolon (;). A macro will have a macro head and a macro body as shown below:

```
^B, <macro name>;

(Macro head defining parameters to be entered from the terminal, their
prompts, and their default values.)
^;
(SINTRAN III commands, input to programs, and dummy parameters in
the required positions. The dummy parameters will be replaced with
actual parameters entered from the terminal.)
^E;
```

The directive ^B,<macro name>; starts a new macro. The <macro name> may consist of up to 16 upper case letters, digits, or hyphen (-). The directive ^E; ends the macro. All user defined macros are normally stored consecutively on one file.

The directive ^; separates the the macro head from the macro body. The other directives to be used in the macro head, are shown below:

DIRECTIVE	MEANING
^P,n,<prompt string>;	Defines a parameter to be entered from the terminal. The parameter will be assigned the number n. The parameter will be prompted by the specified <prompt string>.
^F,n,<prompt string>;	Same as above, except that terminal input is assumed to be a SINTRAN III mass storage file. PERFORM will expand abbreviated file names. The default file type is :SYMB.
^D,n,<default string>;	Default value to be used for parameter n if no terminal input is given.
^L,<information>;	The infomation will be displayed on the terminal when processed by PERFORM.
^C,<comment string>;	Comment. It will be ignored by PERFORM.

The numbers n must be consecutive and in the range 1 - 20. These numbers are used after a reversing slant (\) in the macro body wherever a parameter from the terminal should be inserted. Here is a simple example:

```
^B, FTN;
^F,1,1,PROGRAM TO BE COMPILED: ;
^;
@FORTRAN-100
COMPILE \1,,TEMP:BRF
EXIT
^E;
```

When PERFORM processes this macro, it will ask for the name of the program specified by \1. The answer given at the terminal will be inserted in the command COMPILE \1,,TEMP:BRF in the mode file produced by PERFORM.

In general, PERFORM can be used to insert any text strings. For example, a text string could be a part of a parameter, or it could be a complete SINTRAN III command. If two consecutive reversing slants are encountered, they are treated as one reversing slant. No parameter will be substituted.

The character used to indicate the beginning of a directive can be any character other than A - Z, 0 - 9, or a space. PERFORM uses the first character it finds in the macro file as the directive character. It must be the same character throughout the file. In this manual the the circumflex (^) is used.

2.2 STARTING PERFORM

PERFORM will create a mode file by merging a macro with terminal input. The mode job will normally be started immediately with the terminal as the mode output file. You start PERFORM by writing:

@PERFORM (<macro file>),(<macro name>),
(<macro parameter 1),(<macro parameter 2),...

Omitted parameters will be prompted. The <macro file> is the file containing the macro with the specified <macro name>. The default <macro file> is PERFORM-LIB:MCRO and the default file type is :MCRO. The first macro on the specified file is the default <macro name> .

The parameters <macro parameter 1>, <macro parameter 2> ,... are input parameters to the given macro. If omitted, these will be prompted as specified in the macro.

PERFORM will create a mode file called MACROn:MODE and execute it. The "n" in the file name is a number from 1 - 9. When the mode job has been executed, you will return to SINTRAN III.

Assume the FTN macro in the previous section is stored on a file PMLIB:MCRO. A FORTRAN program QUICKSORT can then be compiled by entering:

@PERFORM PMLIB:MCRO, FTN, QUICKSORT

All parameters can be prompted for.

2.3 EXAMPLE OF USING PERFORM

The following example shows how PERFORM can be used to compile, load, execute, and print FORTRAN programs. The following macro is first written to a macro file using an ordinary editor:

```
(Other macros on the same file)

^B,FTNRUN;
^L,MACRO TO COMPILE, LOAD, AND EXECUTE FORTRAN A PROGRAM;
^P,1,PROGRAM TO BE COMPILED: ;
^F,2,RUNTIME LIBRARY: ;
^D,2,FORTRAN- 1BANK;
^C,FORTRAN-1BANK USED AS DEFAULT RUNTIME LIBRARY;
^P,3,NUMBER OF PRINT COPIES: ;
^;
@DELETE-FILE !1:BRF
@FORTRAN-100
COMPILE !1:SYMB,,"!1:BRF"
EXIT
@DELETE-FILE !1:PROG
@NRL
PROG-FILE "!1:PROG"
LOAD !1:BRF, !2
EXIT
@!1:PROG
@APPEND-SPOOLING-FILE LINE-PRINTER, !1:SYMB, !3,',,
@CC NUMBER OF PRINT COPIES GIVEN AT THE TERMINAL
^E;
```

Three macro parameters are defined: the program to be compiled (!1), the runtime library to be loaded (!2), and the number of copies to be printed (! 3). The default runtime library is FORTRAN- 1BANK.

Assume that the macro is stored on the file PERFORM-LIB:MCRO. A program QUICKSORT is compiled, loaded, executed, and printed as shown below:

```
@PERFORM PERFORM-LIB, FTNRUN
MACRO TO COMPILE, LOAD, AND EXECUTE A FORTRAN PROGRAM
PROGRAM TO BE COMPILED: QUICKSORT
RUNTIME LIBRARY:
NUMBER OF PRINT COPIES: 1
@MODE MACRO1:MODE,TERMINAL

(Output from the execution of the created mode file)
```

The mode file MACRO1:MODE, is created and executed immediately. It is shown below. The terminal is selected as the mode output file.

```
@DELETE-FILE QUICKSORT:BRF
@FORTRAN-100
COMPILE QUICKSORT:SYMB,,''QUICKSORT:BRF''
EXIT
@DELETE-FILE QUICKSORT:PROG
@NRL
PROG-FILE ''QUICKSORT:PROG'' LOAD QUICKSORT:BRF, FORTRAN-1BANK
EXIT
@QUICKSORT:PROG
@APPEND-SPOOLING-FILE LINE-PRINTER, QUICKSORT:SYMB, 1,',,
@CC NUMBER OF PRINT COPIES GIVEN AT THE TERMINAL
```

The mode file MACRO1:MODE will be stored on your user area until it is overwritten by another execution of PERFORM.

2.4 LISTING DEFINED MACROS

The macros defined on a particular macro file can easily be listed. Start PERFORM and let the <macro name> parameter be prompted. Then type a "?", and all macros on the given <macro file> will be listed as shown below:

```
@PERFORM
:MCRO file name: PMLIB:MCRO
MACRO NAME: ?
Macros available in file PMLIB:MCRO

(List of macros on PMLIB:MCRO)

MACRO  NAME:
```

Afterwards PERFORM will once more prompt the <macro name> to be used.

2.5 OPTIONAL CONTROL PARAMETERS

PERFORM accepts some optional parameters. These can be used to specify special mode or batch output files, to control execution, or to select alternative names of the mode file produced. The complete PERFORM call is:

@PERFORM (<macro file>),(<macro name>),(<optional parameters>),
(<macro parameter 1>),(<macro parameter 2>),.....

The <optional parameters> may be used to specify a mode output file other than the terminal. The file name must be preceded by a "<". A new file may be created by enclosing the file name in quotes. The default file type is :SYMB. The <optional parameters> may also include:

>RUN,	Create a mode file and execute it (default)
>CREATE,	Create a mode file, but do not execute it
>BATCHn,	Create a mode file and append to batch number n

The parameters >RUN, >CREATE, and >BATCHn may be abbreviated to >R, >C, and >Bn. PERFORM will by default use the mode file MACROn:MODE. The <optional parameters> may specify another mode file by:

*MODE <file name>,

The default file type is :MODE. This is necessary if the mode job will be waiting in a batch queue the next time PERFORM is called. Otherwise MACROn:MODE will be overwritten. The following are some examples of PERFORM calls:

@PERFORM PMLIB, FTN, <LISTFILE:SYMB
@PERFORM PMLIB, FTN, >CREATE
@PERFORM PMLIB, FTN, <OUTBATCH>BATCH2
@PERFORM PMLIB, FTN, *MODE TESTMACRO:MCRO
@PERFORM PMLIB, FTN, <LISTFILE> CREATE,*MODE TESTMACRO

The macro named FTN on the macro file PMLIB:MCRO is used. The examples show how the <optional parameters> can be used. The macro parameters may follow the <optional parameters>.

2.6 EXTENDED PARAMETER SUBMISSION

Any <macro parameter> in the PERFORM call can be replaced by a file name, preceded by an opening bracket ([ ). The file should contain a list of values for the parameter, one per line.

Mode files will be created and executed repeatedly, taking successive values for the parameter from the file. For example, assume the file PARAMLIST contains:

```
SORT:SYMB
TEST:SYMB
QUICKSORT:SYMB
```

The PERFORM call

```
@PERFORM PMLIB, FTNCOMPILE, [PARAMLIST
```

will compile SORT:SYMB, then TEST:SYMB, and then QUICKSORT:SYMB.

2.7 LIMITATIONS, RESTRICTIONS AND DEFAULTS

The macro name must be unique. If it is defined more than once, the first occurrence is taken. The macro name should not be abbreviated. If it is abbreviated, the first matching occurrence will be taken. The macro cannot be nested, nor invoke other macros.

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

Use the ^F directive rather than the ^P directive in the macro if SINTRAN III file names are to be inserted. The ^F directive will attempt to find the full SINTRAN III file name. If successful, that name will be inserted in the mode file. The default file type is :SYMB.

The reversing slant (\) does not exist on some terminals. The character to use is ASCII 134B. The circumflex (^) is the ASCII character 136B.

PERFORM can be used together with JEC (JOB EXECUTION CONTROL) for further flexibility. JEC is described in this manual.

2.8 PREDEFINED MACROS

PERFORM has the following standard macros stored on the file PERFORMLIB:MCRO. The first macro on the file, FTN, is the default <macro name>.

MACRO 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 DEBUG A COBOL PROGRAM
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
BASIC	COMPILE A BASIC PROGRAM
BASRUN	COMPILE, LOAD AND EXECUTE A BASIC PROGRAM
CREDIR	CREATE AND ENTER A DIRECTORY WITH A USER AREA

Detailed information about each macro is found by inspecting the file by using an editor.

3 JEC - JOB EXECUTION CONTROL

JEC (JOB EXECUTION CONTROL) is a program which lets you control the execution of a batch or mode file by including a few control commands. Intelligent actions can be taken when special situations occur in commands, subsystems, and your programs.

Here are some of the things you can do:

- Terminate execution at any point, for example, where errors are detected. (See page 7.)
- You may execute nested mode files that have a return status showing whether they executed successfully or not. (See page 7.)
- You may use arithmetic. (See page 9.)
- You can create your own numeric and string variables. For instance, you can prompt for the name of the program and the language it is to be compiled in. Thus you can make a single mode file that can compile and load any program. See the example on page 20. You may use your own variables in SINTRAN commands, as parameters to your own programs, as loop counters, or in arithmetic expressions. (See page 9.)
- Answer "questions" that the mode file poses. (See page 10.)
- You may make conditional tests, based on the values of the completion code, the SSI code, or the status code. (See page 12.)
- You may make conditional tests, based on the day, date, or month you execute your mode file. (See page 12.)
- Jump forward and backward to numeric labels defined in your batch or mode file. (See page 11.)
- Create loops so that things can be done a certain number of times. (See page 14.)
- Give input from your terminal to programs you execute in mode jobs. (See page 15, Section 3.3.6.)
- You may turn communication with your terminal on and off in a mode job. (See page 15.)
- You may send output to your terminal, an output file, or both. (See page 15.)
- You may execute mode files on remote systems. The JEC completion code shows whether they executed successfully or not.

- You have the possibility of executing only certain parts of your input file. See the example on page 18.

Before we look at JEC mode files, we will look at what happens when you call JEC interactively from SINTRAN, because that allows us to explain the error codes that JEC uses.

> **A TIP:**
> If you type your mode files in NOTIS-WP, make sure they are in 7- or 8-bit format, not in 16-bit format!

3.1 Interactive JEC and Error Codes

Type @JEC in SINTRAN and you should see something like this:

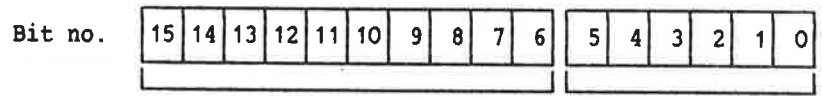
```
@JEC
== Jec ================================================
== Jec ==   Value of completion code is:     0       0B
== Jec ==   Value of SSI code is       :    72     110B
== Jec ==   Last running subsystem was : Notis WP / PED
== Jec ================ ========================= ========
```

The last subsystem you used.

The error code.

The numbers you get will most likely not be the same.

The completion code is stored in a 16-bit word:



This part contains the SSI code if the status code is not zero.

This part is the status code.

Since each digit in an octal number represents three bits, the status code is always the two rightmost digits of the completion code.

The Standard Subsystem Identification code (SSI code) indicates the last subsystem that was running, and the status code indicates what error occurred.

For example, an SSI code of 1 means that the error occurred in the SINTRAN file system (see the following table). If the completion code is 137, you can look in the SINTRAN Reference Manual, ND-60.128, and find that the file system error code 137 means "No spooling for this device."

Here are some SSI codes and the software product(s) they represent. If you are using an older version of one of the products below, it will not produce SSI codes.

SSI code Decimal	SSI code Octal	Product
0-3	0-3B	SINTRAN-III File system (version I)
4-5	4-5B	FORTRAN (version B, library)
6-7	6-7B	COBOL (version F, compiler and library)
20-21	24-25B	PLANC (compiler)
40	50B	SORT-MERGE (version D)
42-43	52-53B	Linkage-Loader (version F)
47	57B	NRL (version J)
72-73	110-111B	NOTIS-WP and PED
96-97	140-141B	NOTIS-TF 500 (version K)
96-97	140-141B	NOTIS-TF 100 (version L)
112	160B	User Environment
117	165B	JEC (version B)
148-159	224-237B	SIB-DML (version E)
216	330B	FILE-HANDLER (version A)
224-225	340-341B	BACKUP-SYSTEM (version F)
260-262	404-406B	COSMOS (version B)
263	407B	TRANSFER-FILE (version B)
265	411B	XMLib

Here are two examples of errors and the codes they produce for JEC. Type the following at your terminal:

```
@DELETE-FILE ASDFG:HJKL ↵
@JEC ↵
```

When you try to delete the nonexistent file ASDFG:HJKL, you will get the message "No such file name". Writing JEC will print the following:

```
== Jec ============================================================
== Jec ==   Value of completion code is:    46      56B
== Jec ==   Value of SSI code is       :     0       0B
== Jec ==   Last running subsystem was : SINTRAN
== Jec ==   Error message: No such file name
== Jec ============================================================
```

The SSI code of 0 means that this is a SINTRAN File System error. If you look in the SINTRAN Reference Manual, ND-60.128, you will see that error 46 is "No such file name".

If you have COSMOS and JEC on your system, and a file called MY-FILE:SYMB, type the following:

```
@TRANSFER-FILE NOSUCH.XYZ MY-FILE ↵
@JEC ↵
```

You should get this message:

```
== Jec ========================================================
== Jec ==   Value of completion code is: 16993   41141B
== Jec ==   Value of SSI code is       :   263     407B
== Jec ==   Last running subsystem was : COSMOS File Transfer
== Jec ==   Error message: Unknown remote system name
== Jec ==   Error in     : XMSG
== Jec ========================================================
```

If you are wondering why the completion code does not start with 407 as the first three octal digits, here is the answer. The last subsystem that was running (407B, which is Transfer File) called subsystem 411, which is XMLib, and error 41 of XMLib occurred.

3.1.1 Why Use the Error Codes?

When you type @JEC BEGIN in a JEC mode file, the completion code will be zero. It will remain unchanged until an error occurs. You can thus specify what should happen when a specific error occurs by using its error code in a @JEC IF statement. For instance, you could type a statement like this in a JEC mode file:

```
@JEC IF completion-code > 0 TERMINATE
```

That would stop the mode file execution if an error occurs.

Note that for some systems it may be better to type:

```
@JEC IF status-code > 27B TERMINATE
```

This is because some ND subsystems use the following system of status codes:

```
    0 = OK
 1-17B = Informative messages
20-27B = Most likely to be informative messages
30-47B = Most likely to be error conditions
50-76B = Error conditions
   77B = Fatal error
```

Look in the manual for the subsystem you are interested in to see what codes are error messages.

Here are some things that may happen during a mode job that require special action:

- You cannot access a file because it is already open or does not exist.
- The first of many compilations does not succeed so there is no reason to continue.
- A remote system in your COSMOS system may not be available at the moment you run your mode job.
- A program you try to start may not be available.

The JEC mode file will not abort when these things happen, so you could start an alternative program, create the file you need, or jump over other commands that are no longer needed.

3.2 An Introductory Example of a JEC Mode File

Here is a small example of a JEC mode file that lets you compile as many or few COBOL programs as you want:

```
@JEC BEGIN
@JEC MESSAGE 'Mode file to compile COBOL-500 program modules'
@JEC DEFINE <number>,<name>
@JEC DEFINE <counter>=1
@JEC INQUIRE <number> 'How many files do you want to compile?'
@CC    ----------------------------------------%
@JEC FOR <counter> IN <counter>:<number> DO       % THE MODE
@JEC INQUIRE <name> 'What is the program name?'   % FILE LOOPS
@JEC ND COBOL-500                                 % HERE, BUT
COMPILE <name>,0,<name>                           % HAS A
EXIT                                              % CONTROLLED
@JEC WHILE COMPLETION-CODE = 0                    % EXIT IF C-C
@JEC END-FOR                                      % IS NOT 0.
@CC    ----------------------------------------%
@JEC IF COMPLETION-CODE > 0 GO TO 1000
@JEC MESSAGE 'Compiling went fine'
@JEC END
@JEC.1000
@JEC MESSAGE 'Compiling failed, error in <name>'
@JEC PRINT-COMPLETION-CODE
@JEC END
```

When you run the above mode file, you will be asked how many files you want to compile, and then you will be asked for each file name. The mode job ends early if any compilation fails due to the WHILE COMPLETION-CODE = 0 statement.

Of course, the mode file needs a few more tests, for instance, to see if the object file already exists. It could also be expanded to let you choose between COBOL-100 and COBOL-500, or even other languages.

3.3 The JEC Commands

Here are the JEC mode and batch file commands, with short explanations:

```
@JEC BEGIN                                        %Starts a mode job
@JEC END                                  %Ends a mode job execution
@JEC TERMINATE                           %Ends a mode file execution
@JEC CLEAR-COMPLETION-CODE                  %Resets completion code
                                            %          and SSI code

@JEC DEFINE <variable-name>                   %Declares variable(s)
@JEC DEFINE <variable-name> = <value>    %Declares & initializes
@JEC INQUIRE <variable-name> <'message'> %Lets user input value
@JEC <command-or-program>                %Use this when parameters
                                                      %are variables
@JEC RECOVER <program>  %Use this when parameters are variables

@JEC GO TO <numeric-label>                      %Unconditional jump
@JEC IF <JEC-test> GO TO <numeric-label>          %Conditional jump
@JEC IF <JEC-test> <command-or-program>     %Conditional command
@JEC IF <JEC-test> TERMINATE              %Conditional termination
@JEC IF <JEC-test> PERFORM <num.-label>
@JEC IF <JEC-test> PERFORM <num.-label> THROUGH <num.-label>

@JEC <numeric-label>                              %Label definition
@JEC ON-ERROR TERMINATE                   %Conditional termination
@JEC ON-ERROR GO TO <numeric-label>               %Conditional jump
@JEC FOR <variable-name> IN <range> DO              %Begins a loop
@JEC WHILE <condition>              %Use to exit early from loops
@JEC END-FOR                                          %Ends a loop
@JEC PERFORM <numeric-label>
@JEC PERFORM <numeric-label> THROUGH <numeric-label>

@JEC PRINT-DATE                            %Outputs the current date
@JEC PRINT-COMPLETION-CODE           %Outputs the completion code

@JEC MODE-INPUT                           %Gets input from mode file
@JEC MODE-OUTPUT                          %Sends output to mode file
@JEC TERMINAL-INPUT                        %Gets input from terminal
@JEC TERMINAL-OUTPUT                       %Sends output to terminal
@JEC WAIT-FOR-CR                %Wait for user to press the ↵ key
```

Let us take a closer look at these commands.

3.3.1 BEGIN, END, and TERMINATE

@JEC BEGIN and @JEC END both initialize the completion code to zero. @JEC BEGIN should always start a mode or batch job and @JEC END should end it:

```
@JEC BEGIN
@JEC       %  JEC and SINTRAN commands
@JEC END
```

Once @JEC END is encountered, the execution of your mode or batch job ends. If you do not end a mode job with @JEC END, you may have problems with the next mode file you run if it does not use @JEC.

A mode file to be run as a batch job should look like this:

```
@ENTER user-name,password,project-password,max-time
@JEC BEGIN
@JEC       %  JEC and SINTRAN commands
@JEC       %  Do not use TERMINAL-INPUT or TERMINAL-OUTPUT,
@JEC       %  INQUIRE, WAIT-FOR-CR or MESSAGE.
@JEC END
```

@JEC TERMINATE ends the execution of the batch or mode file it is in. It will not reset the completion code to zero. You use @JEC TERMINATE in mode files called from other mode files.

If you use nested mode files, @JEC BEGIN and @JEC END should only appear once in the entire mode job. @JEC TERMINATE can be used in the nested files. Here is an example:

File: LOAD-MODE:MODE

```
@ENTER SYSTEM XXXXX,,10,,
@JEC BEGIN
@CC    various other commands
@JEC MODE (UTIL)XMSG-START:MODE,,  ------>
@CC The XMSG file should NOT contain
@CC JEC BEGIN and JEC END.
@JEC MODE (UTIL)SET-TERM-TYPE:MODE,,
@CC The SET-TERM file should NOT
@CC contain JEC BEGIN and JEC END.
@CC    various other commands
@JEC END
```

File: XMSG-START:MODE

```
@JEC ON-ERROR TERMINATE
@JEC SINTRAN-SERVICE
@STOP-XMSG
@EXIT
@CC  other commands
@CC  other commands
@CC  end of file
```

If an error occurs in the file XMSG-START:MODE, the rest of the file will not be executed, but none of the variables JEC uses in the LOAD-MODE:MODE file will be affected. It would be a big mistake to start XMSG file with @JEC BEGIN. It would also be wrong to end it with @JEC END.

Here is one way to alter the LOAD-MODE file above to see whether the nested mode file XMSG-START executed properly:

```
@JEC CLEAR-COMPLETION-CODE
@JEC MODE (UTIL)XMSG-START:MODE,,
@JEC IF COMPLETION-CODE = 0 GO TO 500
@JEC MESSAGE 'An error occurred in XMSG-START:MODE file'
@JEC PRINT-COMPLETION-CODE
@JEC 500
```

In the nested files, you may use TERMINATE in an IF statement, for example:

```
@JEC IF COMPLETION-CODE > 27B TERMINATE
```

See also page 12.

3.3.2 CLEAR-COMPLETION-CODE

CLEAR-COMPLETION-CODE will set the completion code and the SSI code to zero. Here is an example:

```
@JEC DELETE-FILE <VAR1>:NRF
@JEC IF COMPLETION-CODE = 46 GO TO 200 % No such file name.
@JEC IF COMPLETION-CODE > 0 GO TO 1000 % Exit if error.
@JEC 200
@JEC CLEAR-COMPLETION-CODE
@JEC ND COBOL-500
DEBUG-MODE
COMPILE <VAR1>:SYMB,0,"<VAR1>"
EXIT
@JEC IF COMPLETION-CODE > 0 GO TO 1000
@CC       %Here you could load the :BRF file, for example.
@JEC 1000 %Here you could type @JEC END, for example.
```

3.3.3 DEFINE and INQUIRE

By using DEFINE, you can create your own variables that you use in IF and FOR statements, or as macros in command parameters. You may give them values when you define them, or when you execute your mode file by using INQUIRE.

Here are a number of different examples:

Define and Initialize Strings:

```
@JEC DEFINE <file-1>='old-prog'
@JEC DEFINE <file-2>=delete-me
@JEC DEFINE <suffix>='data'
@JEC DELETE-FILE <file-1>:<suffix>
@JEC DELETE-FILE <file-2>:<suffix>
```

Note two things. Strings need only be enclosed in single quotes ('name' not "name", for example) when they start with a digit. All variable names must start with a less than sign (<) and end with a greater than sign (>). Variable names may not contain spaces.

Define and Initialize Numeric Variables:

```
@JEC DEFINE <var1>   = 10
@JEC <var2>   = <var1>
```

If a variable is already defined, you can omit DEFINE when you assign it a value:

```
@JEC <payday> = 21
@JEC <var2>   = <var2> * <var2>
@JEC <var3>   = (<var1> * 10) + 2 + <var3>
```

As you can see, arithmetical expressions are allowed. Use +, -, *, and / to add, subtract, multiply, and divide. **NOTE - Always precede and follow the signs +, -, * or / by a blank.** It looks nicer and it is the only thing we allow! Extra blanks are allowed.

Do not multiply or divide by JEC variables such as DAY. DAY is explained on page 12. If you need to multiply DAY by a variable, do it like this:

```
@JEC <var1> = DAY
@JEC <var2> = <var1> * <x>
```

Define and Ask User to Give the Value:

Here is an example of INQUIRE. Note the use of @JEC PASCAL when the compiler is called:

```
@JEC BEGIN
@JEC DEFINE <file-to-compile>
@JEC DEFINE <list>
@JEC INQUIRE <file-to-compile>
@JEC INQUIRE <list> 'Give list file name and type:'
@JEC PASCAL  % You must type @JEC here so that PASCAL  ;
             % gets the values stored in the variables
COMPILE <file-to-compile>,<list>,<file-to-compile>
EXIT
@JEC END
```

As you can see, INQUIRE can be followed by a message if you so choose. In the above example, this will appear on the screen when you execute your JEC mode file:

```
Give list file name and type:_
```

If there is no text after @JEC INQUIRE, you get this when you execute:

```
VALUE FOR <file-to-compile>?_
```

If you want to compile COB-DB:SYMB, you simply answer COB-DB or 'COB-DB'. But if the file name begins with a number, you must enclose it in single quotes.

Getting Values from a File

At times, you may want to give so many values that you do not want to do it interactively or in your mode file. You may, for example, want to change the file access to all 50 files you have. You do that as follows:

```
@LIST-FILES,,FILE-LIST:DATA
```

The file FILE-LIST will look like this:

```
FILE 1 : (PACK-ONE:UTILITY)EX:SYMB;1
 ... files 2 to 49 ...
FILE 50 : (PACK-ONE:UTILITY)FORMAT:TEXT;1
```

Edit it so that everything to the left of the first parenthesis is gone:

```
(PACK-ONE:UTILITY)EX:SYMB;1
 ... files 2 to 49 ...
(PACK-ONE:UTILITY)FORMAT:TEXT;1
```

Then create a mode file like this:

```
@JEC BEGIN
@JEC DEFINE <public>,<friend>,<own>,<file>,<i>,<number>
@JEC MESSAGE 'Specify the three access types you want'
@JEC INQUIRE <public>
@JEC INQUIRE <friend>
@JEC INQUIRE <own>
@JEC INQUIRE <number> 'How many files do you have?'
@JEC FOR <i> IN 1 : <number> DO
@JEC <file>=FILE-LIST:DATA(<i>)
@CC     <FILE> will be equal to record <i> in FILE-LIST:DATA
@JEC SET-FILE-ACCESS <file> <public> <friend> <own>
@JEC END-FOR
@JEC END
```

If you do not know how many files you have, the loop can look like this:

```
@JEC FOR <I> IN 1 : 1000 DO
@JEC <file>=FILE-LIST:DATA(<i>)
@CC     <FILE> will be equal to record <i> in FILE-LIST:DATA
@JEC WHILE COMPLETION-CODE = 0
@CC     You will safely exit the loop when you reach
@CC     the end of the file FILE-LIST:DATA
@JEC SET-FILE-ACCESS <file> <public> <friend> <own>
@JEC END-FOR
```

Editing Text in INQUIRE

When you are inputting values to an INQUIRE command, you may make typing mistakes. In that case, use the ← key to erase the error. JEC accepts the same control characters for editing as SINTRAN.

3.3.4 GO TO, IF, FOR and PERFORM

Unconditional Jumps (GO TO)

You can jump unconditionally to another part of the mode file:

```
@JEC GO TO 100
...                 % Other JEC statements
@JEC 100            % This is a numeric label
```

If you want to use easier to understand labels, do it like this:

```
@JEC DEFINE <compile> = 500
@JEC GO TO <compile>
...                          % Other JEC statements
@JEC <compile>:              % This is also a numeric label
```

The colon (:) tells JEC that <compile> is a label and not the name of a program to be executed. See the example on page 20. You only need to use a colon when you use a variable as a label.

Conditional Jumps (IF)

There are four types of conditional jumps using IF:

```
@JEC IF <JEC-test> GO TO <numeric-label>
@JEC IF <JEC-test> <command-or-program>
@JEC IF <JEC-test> TERMINATE
@JEC IF <JEC-test> ;
 PERFORM <numeric-label> THROUGH <numeric-label>
```

Remember that the semicolon continues the line.

Conditional tests (IF <JEC-test>)

The <JEC-test> may use the following operators in JEC tests:

=	<	>	OR	AND	NOT	(	)	><

The following JEC variables may be used in JEC tests:

NAME	EXPLANATION
COMPLETION-CODE	Error code.
STATUS-CODE	The last two octal digits in the completion code.
SSI-CODE	Subsystem code.
DATE	A string with 8 letters, for example, 84.09.18 means September 18th, 1984.
DAY	An integer from 1 to 31 or a string from "MONDAY" to "SUNDAY".
MONTH	An integer from 1 to 12.
RUN-MODE	Either 'B' or 'M', depending on whether it is a Batch or Mode job.

You may also test any variables you define. Remember not to mix data types. Do not type @JEC IF DATE = DAY GO TO 1000, for example!

Examples of IF <JEC-test> Statements

Complex expressions must be enclosed in parentheses. The following examples show legal JEC tests:

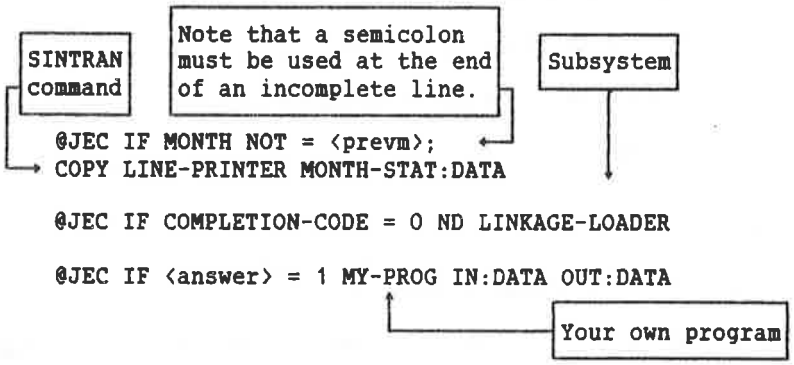
```
@JEC IF COMPLETION-CODE > 0B TERMINATE
@JEC IF (SSI-CODE = 6B AND STATUS-CODE < 20B) GO TO 100
@JEC IF (DAY < 8 AND DAY = 'MONDAY') GO TO 100
@JEC IF (DAY = 20 AND (NOT DATE = 84.01.20)) GO TO 100
@JEC IF <answer> = 2 GO TO 2000
@JEC IF <answer> NOT > 0 GO TO 3000
@JEC IF COMPLETION-CODE = 0 BRF-LINKER

@JEC DEFINE <payday> = 21  % Omit DEFINE if <payday>
@CC                        % is already defined.
@JEC IF <payday> NOT = DAY THEN TERMINATE

@JEC IF RUN-MODE = 'B' GO TO <batch>
@JEC GO TO <mode>
```

JEC uses decimal numbers by default. Octal numbers must be followed by a B. A numeric label such as 100 in GO TO 100 must be defined somewhere in the mode or batch file by the command @JEC 100. Both forward and backward jumps are legal. Only incurable hackers should use octal numbers in labels.

You may use SINTRAN III commands, subsystems, or your own programs as <command or program>.



Conditional Jump (ON-ERROR)

There are two types of conditional jumps using ON-ERROR:

1) @JEC ON-ERROR TERMINATE

2) @JEC ON-ERROR GO TO <numeric-label>

The statement after ON-ERROR is performed if the completion code is not equal to zero. Note that you cannot use <command or program> or PERFORM <label> THROUGH <label> after ON-ERROR. Use instead:

```
@JEC IF COMPLETION-CODE > 0 <program or SINTRAN command>
@JEC IF COMPLETION-CODE > 0;
PERFORM <numeric-label> THROUGH <numeric-label>
```

If you use @JEC ON-ERROR, and an error occurs, the following rules apply:

1) The error can occur anywhere in the file.

2) The action TERMINATE or GO TO will be performed when the next @JEC statement is encountered.

You should only use @JEC ON-ERROR once in a file!

Here are two examples:

1) @JEC ON-ERROR GO TO 5000

2) @JEC ON-ERROR GO TO <finish>
   ...
   @JEC <finish>:

FOR Loops

You can create loops as follows:

```
@JEC FOR <variable-name> IN <range> DO          %Begins a loop
@JEC END-FOR                                      %Ends a loop
```

This will execute the same program ten times:

```
@JEC DEFINE <i>, <program-name>
@JEC INQUIRE <program-name> 'Which program do you want to run?'
@JEC FOR <i> IN 1:10  DO
@JEC RECOVER <program-name>
@JEC END-FOR
```

Here is a complete mode file that a system supervisor might use to log out all the users on Terminal Access Devices (TADs):

```
@JEC BEGIN
@JEC DEFINE <i>, <x>
@JEC 1000
@JEC INQUIRE <x> 'How many TADs does your system have?'
@JEC DEFINE <lasttad>= 767 + <x>
@JEC IF <lasttad> < 767 GO TO 1000   % No TADs have LDN < 767
@JEC FOR <i> IN 767:<lasttad>  DO
@JEC STOP-TERMINAL <i>
@JEC END-FOR
@JEC END
```

Another example of a FOR loop is given on page 5.

3.3.5 PRINT Commands

The command @JEC PRINT-DATE writes the current date to the batch or mode output file.

```
@JEC PRINT-DATE
== Jec ========================================================
          Year      Month       Day        Time
          1984       12         13       11.32.19
                   December  Thursday
== Jec ========================================================
```

@JEC PRINT-COMPLETION-CODE outputs the completion code.

You can print the value of any variable you define. If your variable is called <name>, type:

```
@JEC MESSAGE '<name>' or:
@JEC MESSAGE 'Name is <name>'
```

3.3.6 Terminal and Mode Input/Output

You can enter parameters to programs within a mode job from your terminal. Input cannot be entered to batch jobs or SINTRAN III commands. The commands to switch terminal input and output on and off are:

```
@JEC TERMINAL-INPUT    %Input to programs from terminal
@JEC TERMINAL-OUTPUT   %Output from programs to terminal
@JEC MODE-INPUT        %Turn terminal input off
@JEC MODE-OUTPUT       %Turn terminal output off
```

Note that @JEC END turns terminal input and output off.

The command @JEC TERMINAL-INPUT will let you input parameters from your terminal. Make sure you remove input parameters from your mode file. Let us say you have a program called AVERAGE:PROG that expects three numbers to be input. You could execute it five times like this:

```
@JEC BEGIN
@JEC DEFINE <i>
@JEC TERMINAL-INPUT
@JEC FOR <i> IN 1:5  DO
@RECOVER AVERAGE
@JEC END-FOR
@JEC END
```

If you can write a short program that expects input, try running the above mode file using your terminal as the output file. Then try it again using another file as the output file. You can still give input from your terminal, but your program prompts will not appear; they are sent to the output file!

Add a line with "@JEC TERMINAL-OUTPUT" to the mode file above and then all prompts from your program AVERAGE will appear on your terminal.

@JEC TERMINAL-OUTPUT will output prompts to your terminal when your terminal is not the output file. Things written to a file will not be sent to your terminal.

@JEC MODE-INPUT turns TERMINAL-INPUT off again, and @JEC MODE-OUTPUT turns TERMINAL-OUTPUT off. Note that terminal I/O is off when you type @JEC BEGIN. Note that if you do not terminate your mode job with @JEC END and terminal input was on, it will still be on when you run the next mode file from your terminal. Remember @JEC BEGIN and END!

It can often be useful to pause while executing a mode file. By writing:

```
@JEC WAIT-FOR-CR 'Insert floppy no. <i>'
```

you let the mode file "pause" until the user pushes the ↵ key. The ↵ key is also called the CR (Carriage Return) key. You may have any message, or none at all, after WAIT-FOR-CR.

Here is an example from a mode file used to copy many files to or from floppy diskettes:

```
@JEC RELEASE-DIR <dir>
@JEC MESSAGE 'Remove diskette <number>'
@JEC DEFINE <number> = <number> + 1
@JEC MESSAGE 'Insert diskette <number>'
@JEC WAIT-FOR-CR
@JEC ENTER-DIR <dir> <dev> <unit>,,,
@CC       Copy files to or from the diskette.
```

3.3.7 Comments Start with %

The percentage sign (%) indicates that the rest of the line only contains comments. If a JEC command consists of more than one line, any incomplete lines must end with a semicolon (;), for example:

```
@JEC IF (COMPLETION-CODE < 400B AND COMPLETION-CODE > 500B);
GO TO 100                              %Example of split line
```

3.4 Examples of JEC Mode and Batch Files

This section shows examples of JEC commands used within batch and mode files.

3.4.1 An Example Using SORT-MERGE

The following mode file will only print the output file from the ND SORT-MERGE program if no errors occur.

```
@JEC BEGIN
@JEC DEFINE <INPUT>,<OUTPUT>
@JEC INQUIRE <INPUT>;
'Give the file name and type of the file you want to sort:
@JEC INQUIRE <OUTPUT>;
'Which output file? Enclose name in "" if file is new;'
@JEC SORT-MERGE
RECORD-DESCRIPTION 80, 1, TEXT
KEY-DESCRIPTION 1, 10, ASCENDING, ASCII
SORT <INPUT>, <OUTPUT>
EXIT
@JEC PRINT-COMPLETION-CODE
@JEC IF COMPLETION-CODE > 0 TERMINATE
@COPY-FILE PHILIPS, <OUTPUT>
@DELETE-FILE <OUTPUT>
@JEC END
```

3.4.2 Compiling, Loading, and Executing a COBOL Program

The next example shows how a COBOL program is compiled, loaded, and executed. Special actions are taken if compilation errors occur. TEST:PROG will communicate directly with the terminal during execution.

```
@JEC BEGIN
@JEC PRINT-DATE                %Outputs today's date.
@COPY-FILE TEST:SYMB, (PACK-TWO:P-HANSEN)TEST:SYMB
@COBOL-100
COMPILE TEST:SYMB, TEST:ERR, TEST:BRF
EXIT
@JEC IF (COMPLETION-CODE > 0B AND SSI-CODE = 6B) GO TO 111
@CC Go to compiler error part. COBOL-100 has SSI code 6B.
@BRF-LINKER
PROG-FILE TEST:PROG
LOAD TEST:BRF, COBOL-1BANK:BRF
EXIT
@JEC IF STATUS-CODE > 27B TERMINATE
@cc      %
@cc      %    Codes from 0 to 26 are most likely to be
@cc      %    only informational messages in many products.
@cc      %
@JEC TERMINAL-INPUT  %Input to TEST:PROG from terminal.
@TEST:PROG
@JEC MODE-INPUT
@JEC TERMINATE
@JEC 111             %Compilation error handling part.
@COPY-FILE LINE-PRINTER, TEST:ERR
@DELETE-FILE TEST:ERR
@JEC END
```

3.4.3 A Batch File Example

This is a batch file which is to be executed the 20th of every month. Note that @ENTER and double escape are placed outside the @JEC BEGIN and @JEC END commands.

```
@ENTER P-HANSEN,HANS,,,
@JEC BEGIN
@JEC IF DAY = 20 SALARY:PROG
@JEC IF DATE = 83.12.20 ADDSALARY:PROG
@COPY-FILE ND-SAT-II.LINE-PRINTER, OUTSALARY:DATA
@CC PRINTING ON THE REMOTE COMPUTER ND-SAT-II
@JEC IF (STATUS-CODE > 0B AND SSI-CODE < 4B);
DELETE-FILE OUTSALARY:DATA   %Split JEC command
@CC SSI code < 4B INDICATES FILE SYSTEM ERROR
@JEC END
<CTRL 0> <ESCAPE> <CTRL 0> <ESCAPE>
```

3.4.4 A Flexible Compile and Load Mode File

Here is quite a lengthy example. This mode file will compile and load any COBOL, FORTRAN-100, or FORTRAN-500 program. Note how labels are used.

```
@JEC BEGIN
@JEC DEFINE <Fort-500>=500, <Fort-100>=100
@JEC DEFINE <Cobol>=200, <compile>=900
@JEC DEFINE <load-100>=1000, <failure>=8000, <success>=300
@JEC MESSAGE 'Mode file to compile and load a program'
@JEC DEFINE <lang>,<name>,<compiler>,<library>
@JEC MESSAGE 'Which compiler do you want to use?'
@JEC MESSAGE 'FORTRAN-100 = 1          FORTRAN-500 = 5'
@JEC MESSAGE 'COBOL       = 2'
@JEC INQUIRE <lang> 'Answer with 1, 2 or 5:'
@JEC INQUIRE <name> 'What is the name of your program ?'
@CC    ----------------------------------------%
@JEC IF <lang> = 5 GO TO <Fort-500>
@JEC IF <lang> = 1 GO TO <Fort-100>
@JEC IF <lang> = 2 GO TO <Cobol>
@JEC END
@CC    ----------------------------------------
@JEC <Fort-100>              % --- FORTRAN-100  -----
@JEC <compiler> = FORTRAN-100
@JEC <library> = FORTRAN-1BANK
@JEC GO TO <compile>
@CC    ------------------------------------
@JEC <Cobol>                 % --- COBOL -------
@JEC <compiler> = COBOL
@JEC <library> =  COBOL-1BANK
@JEC GO TO <compile>
@CC    ------------------------------------
@JEC <compile>:           % Compile and load an ND-100 program.
@JEC DELETE-FILE <name>:BRF
@JEC CLEAR-COMPLETION-CODE   % In case file did not exist.
@JEC <compiler>
COMPILE <name>,0,"<name>"
EXIT
@JEC IF (COMPLETION-CODE > 0) GO TO <failure>
@CC    ------------------------------------
```

(continued on next page)

(continued from previous page)

```
@JEC <load-100>:          % This label is only for information.
@JEC DELETE-FILE <name>:PROG
@JEC CLEAR-COMPLETION-CODE
@JEC BRF-LINKER
PROG-FILE "<name>"
LOAD <name>,<library>
EXIT
@JEC IF COMPLETION-CODE > 0 GO TO <failure>
@JEC GO TO <success>
@CC    ------------------------------------
@JEC <Fort-500>:
@JEC CREATE-FILE <name>:NRF 0
@JEC CLEAR-COMPLETION-CODE     % In case the file already existed.
@JEC FORTRAN-500
COMPILE <name>,0,<name>
EXIT
@JEC IF COMPLETION-CODE > 0 GO TO <failure>
@JEC ND LINKAGE-LOADER
ABORT-BATCH OFF
DELETE-DOMAIN <name>
SET-DOMAIN "<name>"
OPEN "<name>",,,,,,,
LOAD <name>
LOAD (SYSTEM)FORTRAN-LIB
EXIT
@JEC IF COMPLETION-CODE > 0 GO TO <failure>
@JEC GO TO <success>
@CC    ------------------------------------
@JEC <success>:
@JEC MESSAGE 'Compiling and loading went fine'
@JEC END
@JEC <failure>:
@JEC MESSAGE 'Compiling or loading failed'
@JEC PRINT-COMPLETION-CODE
@JEC END
```

3.4.5 Use of Arithmetic to Create a Continuous File

This mode file creates a continuous file that uses all of your remaining free pages if possible.

```
@JEC BEGIN
@JEC MESSAGE 'Mode file to create the largest possible;
 continuous file.'
@JEC DEFINE <file-name>,<max>=0,<size>=0,<change>=1000
@JEC INQUIRE <file-name>
@JEC 100                    % The program returns here every time
@JEC <max> = <size>         % we successfully create the file.
@JEC DELETE-FILE <file-name>
@JEC DELETE-FILE <file-name>:DATA
@JEC CLEAR-COMPLETION-CODE
@JEC IF <change> < 2 GO TO 5000  % Create a file of size <max>.
@JEC <size> = <size> + <change>
@JEC <change> = <change> / 2
@CC    ------------------------------------------------------
@JEC 2000
@JEC CREATE-FILE <file-name> <size>
@JEC IF COMPLETION-CODE=0 GO TO 100         % Success!
@JEC IF COMPLETION-CODE=67B OR COMPLETION-CODE=75B GO TO 3000
@JEC PRINT-COMPLETION-CODE
@JEC MESSAGE '<file-name> has not been created '
@JEC END
@CC     -------------------------------------------------------
@JEC 3000                   %   <size> was too big
@JEC CLEAR-COMPLETION-CODE
@JEC IF <change> < 2 GO TO 5000  % Create a file of size <max>
@JEC <size> = <size> - <change>
@JEC <change> = <change> / 2
@JEC GO TO 2000
@CC ------------------------------------------------------
@JEC 5000                   % The maximum size has been found
@JEC CREATE-FILE <file-name> <max>
@JEC MESSAGE '<file-name> is <max> pages big '
@JEC FILE-STATISTICS <file-name>,,,,,
@JEC END
@CC
```

3.5 The JEC Library

Programs you write may also read or update the completion code. The JEC library contains two subroutines for this purpose:

```
UEISECCODE(SSI-CODE,COMPL-CODE,STAT)    (write operations)
UEIFECCODE(SSI-CODE,COMPL-CODE,STAT)    (read operations)
```

Each parameter is an integer stored in 2 bytes. The parameter STAT is the status from the monitor call performing the read and write operations. For example, your program EXAMPLE-PROG may contain the subroutine call to update the status code and the SSI code:

```
IF NUMBER = 0 THEN UEISECCODE(710B,71050B,STAT)
```

A JEC command in the mode file can then test the status code and the SSI code after executing your program. The following commands in the mode file can be used:

```
@EXAMPLE-PROG
@JEC (IF SSI-CODE = 710B) OR (COMPL-CODE = 50B) TERMINATE
```

The JEC library for one-bank programs is called JEC-LIB-1B:BRF, and for two-bank programs JEC-LIB-2B:BRF.

We suggest you use SSI-CODEs from 700B to 777B, since they will not be used by any Norsk Data products.

3.6 Some Technical Details

When you type @JEC BEGIN, JEC creates two scratch files:

1) **JEC-xxxxx:DATA** contains all the defined variables and their values, as well as various global information if FOR loops or PERFORM are used.

2) **JEC-xxxxx:MODE** is constructed when you use your own variables in SINTRAN commands, as program parameters, or as program names. The variables you define are replaced with their values on this file, and the file is started by JEC.

The 5 x's (xxxxx) stand for the address of the RT description of your background program, batch processor, or TAD (Terminal Access Device). This means that the file name will always be unique, even if you run several mode or batch jobs simultaneously.

Both files are deleted by the statement @JEC END.

3.7 JEC Syntax

Here is a complete syntax of JEC.

You only need to use the underlined syntax. Note that THROUGH or THRU can be used. Likewise, both GO TO and GOTO are allowed.

BEGIN

CLEAR-COMPLETION-CODE

DEFINE <identifier> % up to 40 ASCII characters long

DEFINE <identifier>=numeric literal

<identifier>=<identifier>

END

```
FOR <identifier> IN [ <identifier>  <identifier> ]  DO
                    [ integer   :   integer      ]

    [ WHILE <condition> ]
END-FOR
```

```
GO TO [ numeric label ]
```

```
IF <condition> THEN [ GO TO [ numeric label ]
                    | TERMINATE
                    | PERFORM numeric label [ THRU ] numeric label
                    |                       [ THROUGH ]
                    | program name / SINTRAN III command ]
```

```
INQUIRE <identifier> ['string of ASCII characters
                                and/or <identifier>']
```

```
MESSAGE 'ASCII string  and/or <identifier>'
```

```
MODE-INPUT
```

```
MODE-OUTPUT
```

```
ON-ERROR [ TERMINATE          ]
         [ GO TO numeric label ]
```

```
PERFORM [ numeric label [ [THRU|THROUGH] numeric label ] ]
```

PRINT-COMPLETION-CODE

PRINT-DATE

TERMINAL-INPUT

TERMINAL-OUTPUT

TERMINATE

WAIT-FOR-CR 'ASCII string'

% Comments in the mode/batch file

Index

4 BACKUP-SYSTEM

4.1 Introduction

The BACKUP-SYSTEM offers a variety of facilities for copying files to and from disks, floppy disks, and magnetic tapes. Files stored on remote computer systems may also be copied.

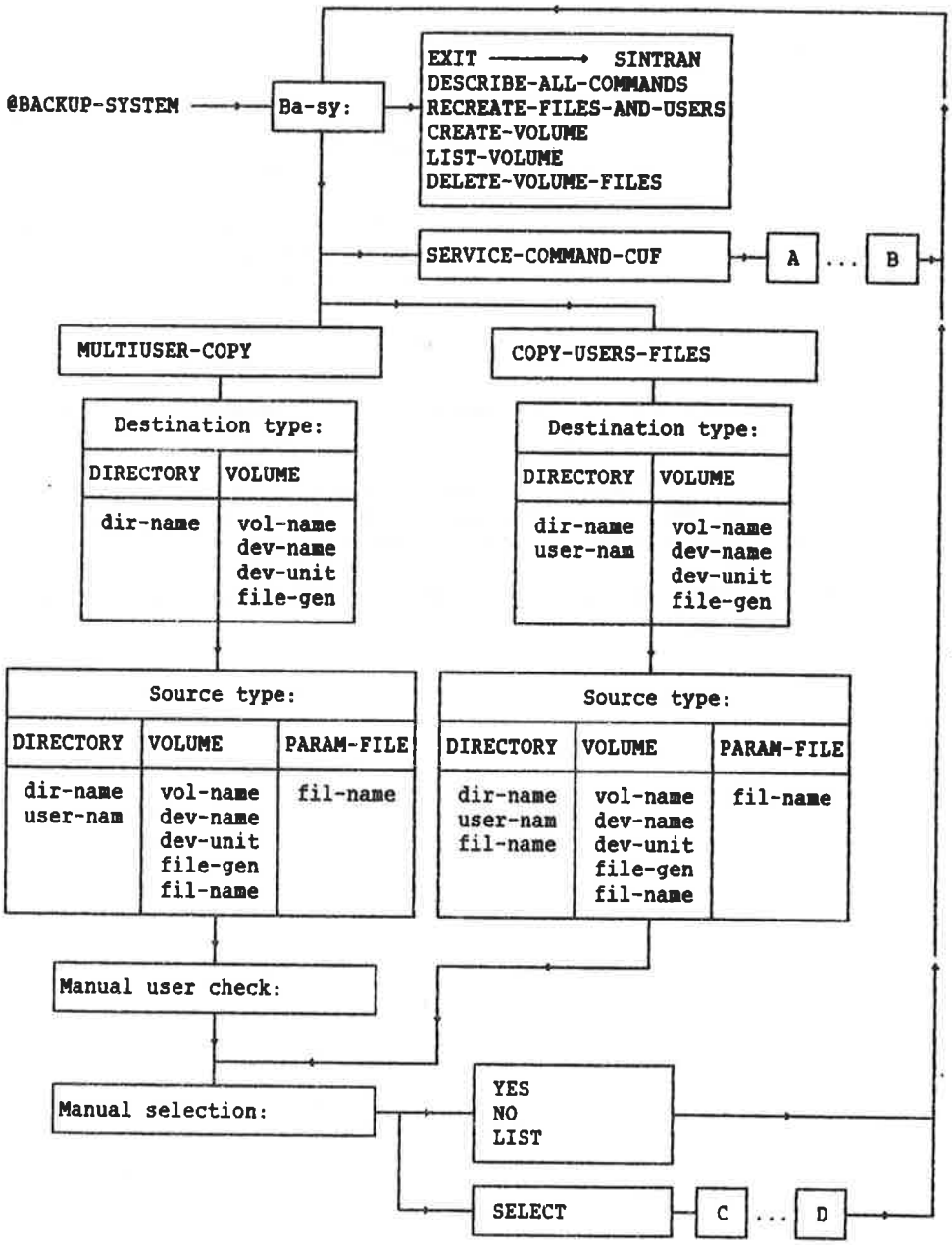
The files may be copied for archiving, backup, or other purposes. To enable communication with other computer installations, the ANSI standard label format is available for magnetic tapes.

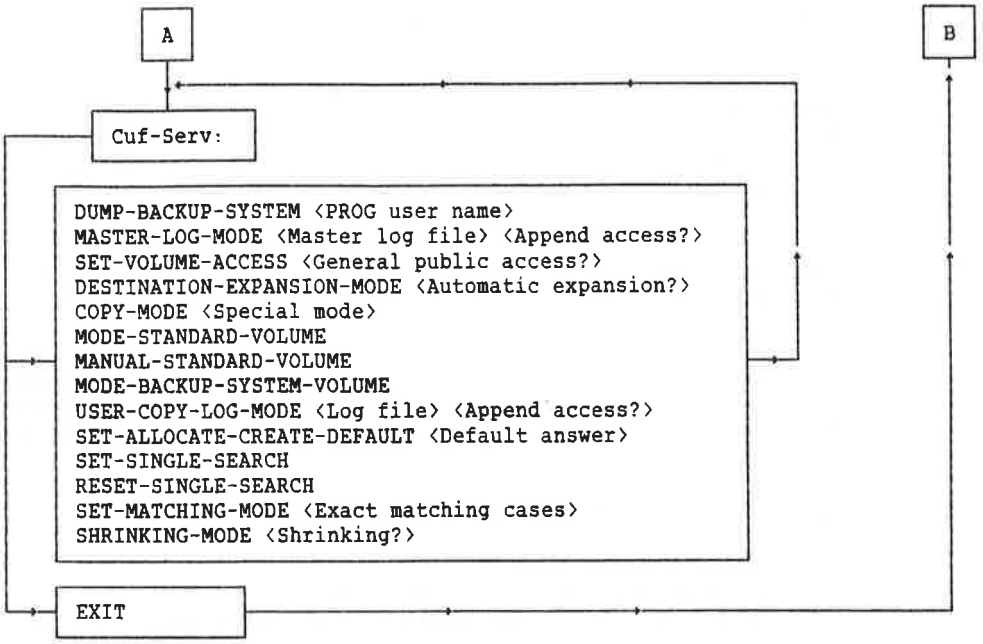
Entering commands to the BACKUP-SYSTEM is easy, but slightly different from SINTRAN III. Some commands have subcommands, i.e., the parameter sequence is not solely determined by the first command entered. Online help information is available for every prompted command, subcommand or parameter at all levels of communication.

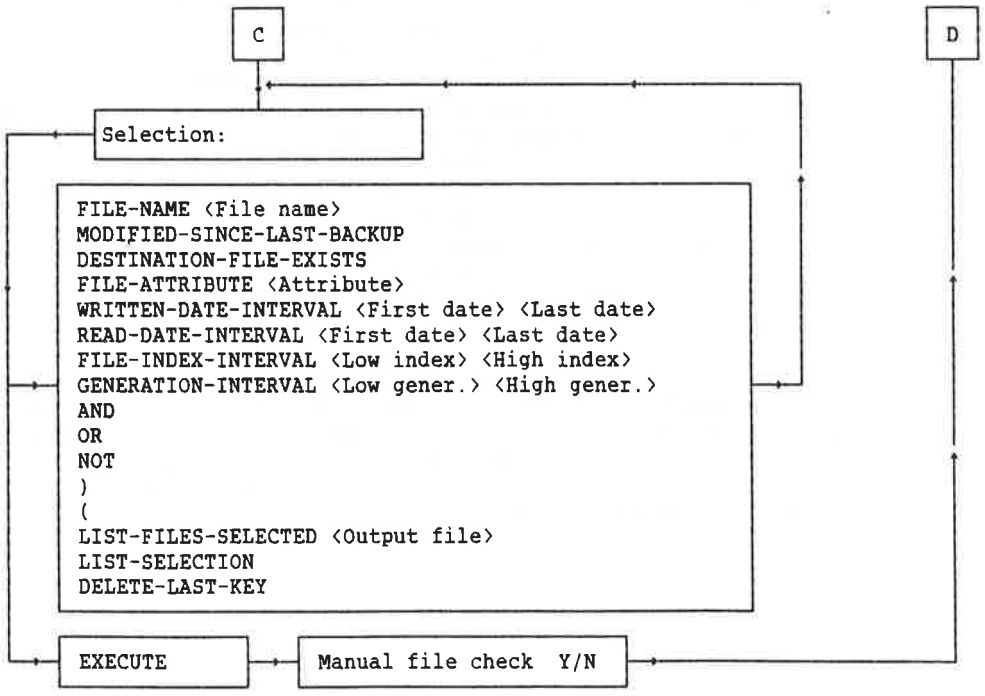
The old SINTRAN III commands @COPY-USERS-FILES, @CREATE-VOLUME and @LIST-VOLUME are now available under the BACKUP-SYSTEM. The commands have some extended and altered facilities.

The BACKUP-SYSTEM can handle files produced under older versions of SINTRAN III.

Here is a pictorial overview of the BACKUP-SYSTEM:




A
B
Cuf-Serv:
DUMP-BACKUP-SYSTEM <PROG user name>
MASTER-LOG-MODE <Master log file> <Append access?>
SET-VOLUME-ACCESS <General public access?>
DESTINATION-EXPANSION-MODE <Automatic expansion?>
COPY-MODE <Special mode>
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
SET-MATCHING-MODE <Exact matching cases>
SHRINKING-MODE <Shrinking?>
EXIT


C
D
Selection:
FILE-NAME <File name>
MODIFIED-SINCE-LAST-BACKUP
DESTINATION-FILE-EXISTS
FILE-ATTRIBUTE <Attribute>
WRITTEN-DATE-INTERVAL <First date> <Last date>
READ-DATE-INTERVAL <First date> <Last date>
FILE-INDEX-INTERVAL <Low index> <High index>
GENERATION-INTERVAL <Low gener.> <High gener.>
AND
OR
NOT
)
(
LIST-FILES-SELECTED <Output file>
LIST-SELECTION
DELETE-LAST-KEY
EXECUTE
Manual file check Y/N

4.2 Command Summary

This section gives an overview of the commands available. Three of the commands, COPY-USERS-FILES, MULTIUSER-COPY and SERVICE-PROGRAM-CUF, have their own set of subcommands. One of the subcommands available under COPY-USERS-FILES and MULTIUSER-COPY has one further level of subcommands.

A detailed description of all commands, subcommands, and parameters is available interactively by entering the command DESCRIBE-ALL-COMMANDS. You may also answer prompted commands or subcommands by typing HELP (<command name>) to have information displayed.

Information about legal parameters is available by answering a prompt with a question mark (?). Information about a particular command or subcommand is available by terminating a command name with a question mark, for example, COPY-USERS-FILES?.

Below is a list of all the commands and their parameters:

```
HELP (<command name>)
DESCRIBE-ALL-COMMANDS (<output file>)
COPY-USERS-FILES (Destination type: Subcommand),
                 (Source type: Subcommand)
                 (Manual selection: Subcommand)
MULTIUSER-COPY (Destination type: Subcommand)
               (Source type: Subcommand)
               (<Manual user check?>)
               (Manual selection: Subcommand)
CREATE-VOLUME <volume name>,<device name>,(<device unit>).
DELETE-VOLUME-FILES <volume name>,<device name>,
                    (<device unit>),
                    (<generation of first file to delete>),
                    (<file name>)
LIST-VOLUME <device name>,(<device unit>),(<file name>),
            (<output file>)
SERVICE-PROGRAM-CUF (CUF-SERV: Subcommands)
EXIT
```

The COPY-USERS-FILES command has subcommands to describe the source and destination of the files to be copied. In this case, parentheses do not indicate parameters with default values, but subcommands. The subcommands will be prompted by Destination type:, Source type: and Manual selection:. A volume is a set of files stored sequentially, for example, on magnetic tape.

Destination type subcommands in the COPY-USERS-FILES command are:

```
DIRECTORY (<destination directory name>),
          (<destination user name>)
VOLUME <destination volume name>,<destination device name>,
       (<destination device unit>),
       (<destination file generation>)
```

Source type subcommands in the COPY-USERS-FILES command are:

```
DIRECTORY (<source directory name>),(<source user name>),
          (<source file name>)
VOLUME <source volume name>,<source device name>,
       (<source device unit>),(<source file generation>),
       (<source file name>)
PARAMETER-FILE <parameter file name>
```

Manual selection subcommands in the COPY-USERS-FILES command are:

```
YES
NO
LIST
SELECT (Selection: Subcommands),<manual file check?>
```

Selection subcommands in the SELECT alternative in manual selection are:

```
FILE-NAME (<file name>)
MODIFIED-SINCE-LAST-BACKUP
DESTINATION-FILES-EXIST
FILE-ATTRIBUTE (<attribute>)
WRITTEN-DATE-INTERVAL (<first date>),(<last date>)
FILE-INDEX-INTERVAL (<low file index>),(<high file index>)
GENERATION-INTERVAL (<low file generation>),
                    (<high file generation>)
AND
OR
NOT
(
)
LIST-FILES-SELECTED (<output file>)
LIST-SELECTION
DELETE-CURRENT-SELECTION
DELETE-LAST-KEY
EXECUTE
```

The MULTIUSER-COPY command is suitable for copying more than one user's files in one operation. All user names matching the entered source user name will be copied. Below is a list of the subcommands:

Destination type subcommands in the MULTIUSER-COPY command are:

```
DIRECTORY (<destination directory name>)
VOLUME <destination volume name>,<destination device name>,
       (<destination device unit>),
       (<destination file generation>)
```

Source type subcommands in the MULTIUSER-COPY command are:

```
DIRECTORY (<source directory name>),(<source user name>)
VOLUME <source volume name>,<source device name>,
       (<source device unit>),(<source file generation>),
       <source file name>)
PARAMETER-FILE <parameter file name>
```

The MULTIUSER-COPY command then has a parameter <manual user check?> before the manual selection subcommands. These subcommands are identical to the manual selection subcommands in the COPY-USERS-FILES command. The default <manual user check?> is to list all matching user names.

Special copying modes for the COPY-USERS-FILES and MULTIUSER-COPY commands are selected by using the SERVICE-PROGRAM-CUF. The SERVICE-PROGRAM-CUF uses the prompt Cuf-serv:. The following subcommands are available:

```
HELP (<command name>)
DUMP-BACKUP-SYSTEM (<bpun user name>)
MASTER-LOG-MODE (<master log file>),(<append access?>)
SET-VOLUME-ACCESS (<general public access?>)
DESTINATION-EXPANSION (<automatic expansion?>)
COPY-MODE <special mode>
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
SET-MATCHING-MODE (<exact matching cases?>)
SHRINKING-MODE (<shrinking?>)
EXIT
```

Some of these subcommands are restricted to user SYSTEM.

4.3 Simple Use of the BACKUP-SYSTEM

One or more files may be copied between disks, floppy disks, or magnetic tape by the COPY-USERS-FILES command. On sequential storage media like magnetic tapes, a volume must be created instead of a directory. The user giving the CREATE-VOLUME command will be the owner of the volume.

A floppy disk may also be used as a sequential storage medium. In that case, a volume must be created on it. The first file of a volume may extend over several volumes. You will be asked to enter a new volume when large files make this necessary, or when there are more files to be copied when using the COPY-USERS-FILES command.

The following is an example of how files can be copied from a disk to a volume on a floppy disk. The two files CHAPTER-ONE:TEXT and CHAPTER-TWO:TEXT are copied to the volume EXVOL. User P-HANSEN has default directory PACK-TWO. The prompts include the default values between slashes ('....').

```
@BACKUP-SYSTEM ↵
Ba-sy: CREATE-VOLUME ↵
  Volume name: EXVOL
  Device name: FLOPPY-DISC-1 ↵
  Device unit: 0 ↵
Ba-sy: COPY-USERS-FILES ↵
  Destination type: VOLUME ↵
    Destination volume name: EXVOL ↵
    Destination device name: FLOPPY-DISC-1 ↵
    Destination device unit: 0 ↵
    Destination file generation'1':  1 ↵
  Source type: DIRECTORY ↵
    Source directory name 'PACK-TWO': ↵
    Source user name 'P-HANSEN': ↵
    Source file name '': ;TEXT ↵
  Manual selection: YES ↵

FILE 4: (PACK-TWO:P-HANSEN)CHAPTER-ONE:TEXT;1
        INDEXED FILE 3 PAGES MODIFIED 29/08-83 (YES/NO?) YES ↵

FILE 5: (PACK-TWO:P-HANSEN)CHAPTER-TWO:TEXT;1
        INDEXED FILE 7 PAGES MODIFIED 14/10-83 (YES/NO?) YES ↵

FILE 9: (PACK-TWO:P-HANSEN)MEMO:TEXT;1
        INDEXED FILE 1 PAGE. MODIFIED 17/03-83 (YES/NO?) NO ↵

Ba-sy: EXIT ↵
```

The copied files will have the same names on the volume as on the source directory. The copied files can be copied back to the disk by using EXVOL as source and PACK-TWO as destination.

If new backup copies of the same files will be stored on the volume later, you can use the <destination file generation> parameter. For example, the file generations can be numbered consecutively. This will help you distinguish between different generations of the same file later on. Alternatively, you could use a date identification, for example, 1125 meaning November 25th.

The SINTRAN III standard device names are FLOPPY-DISC-1, FLOPPY-DISC-2, MAG-TAPE-1, MAG-TAPE-2, MAG-TAPE-3, and MAG-TAPE-4. The device MAG-TAPE-1 unit 0 must have the peripheral file name MAG-TAPE-0, unit 1 must have the name MAG-TAPE-1, etc. To use a volume on a floppy disk, the FLOPPY-DISC-1, unit 0, must have the name FLOPPY-1, unit 1 the name FLOPPY-2, etc.

Another example shows how files can be copied from a disk to a floppy disk. A directory is first created and entered. Then a user with a 616 page user area is created on the directory. Note that some floppy disk systems only allow 148 pages to be used.

```
@CREATE-DIRECTORY BACKUP-84-52,FLOPPY-DISC-1,0 ↵
@ENTER-DIRTORY BACKUP-84-52, FLOPPY-DISC-1,0 ↵
@CREATE-USER BACKUP-84-52: P-HANSEN ↵
@GIVE-USER-SPACE,BACKUP-84-52: P-HANSEN, 616 ↵
@BACKUP-SYSTEM ↵
Ba-sy: COPY-USERS-FILES ↵
  Destination type: DIRECTORY ↵
    Destination directory name'PACK-TWO': BACKUP-84-52 ↵
    Destination user name'P-HANSEN': ↵
  Source type: DIRECTORY ↵
    Source directory name'PACK-TWO': ↵
    Source file name'': ↵
  Manual selection: NO ↵
Ba-sy: EXIT ↵
@RELEASE-DIRECTORY BACKUP-84-52 ↵
```

All files belonging to user P-HANSEN will be copied to the floppy disk. When the files are to be retrieved from the backup copy later, only the command @ENTER-DIRECTORY should be given before the BACKUP-SYSTEM is entered. Note that the source and destination should be interchanged.

Version E of the BACKUP-SYSTEM lets you make a sophisticated selection of source files, and it allows you to copy several users in one command.

You may, for instance, select those files which have been written to in the last week. You may also define logical combinations of simple selections.

For instance, the selection

```
@BACKUP-SYSTEM ↵
Ba-sy: COPY-USERS-FILES ↵
  ...
  Manual selection: SELECT ↵
    Selection: WRITTEN-DATE-INTERVAL ↵
      First date: 84.12.3 ↵
      Last date: ↵
    Selection: AND NOT FILE-NAME :BRF ↵
    Selection: EXECUTE ↵
```

will cause all the files written to since December 3, 1984 to be copied, excluding all :BRF files. Since no last date was specified, the last date is today (actually, it is to infinity).

This may be a useful selection when one user is taking a backup of his/her files.

Note that if any files have suffixes starting with :BRF, such as :BRFA, :BRFB, :BRFZ; :BRF1, etc., they too will not be copied. You can be sure that only exact matches are copied/not copied as follows:

```
Ba-sy: SERVICE-PROGRAM-CUF ↵
  Cuf-serv: SET-MATCHING-MODE ↵
    Exact matching cases: ALL ↵
  Cuf-serv: EXIT ↵
Ba-sy: COPY-USERS-FILES ↵
(continue as above)
```

This will copy files with the suffix :BRFA, :BRFB, etc., but not :BRF.

The first time a user takes a personal backup of his/her files, s/he should avoid selecting WRITTEN-DATE-INTERVAL. Instead, s/he should copy all the files. Subsequent backups need only copy files that have been written to since the last backup.

4.4 Detailed Description of Commands

The BACKUP-SYSTEM can be entered by typing @BACKUP-SYSTEM. You return to SINTRAN III by giving the EXIT command. The BACKUP-SYSTEM uses the prompt BA-SY:.

Files may be copied to and from remote computer systems, provided COSMOS and SINTRAN III version I or later are available. You use information about the remote system as a prefix to the name of a file, directory, or mass storage device. Remote system information consists of the following parts:

<SYSTEM>(<DIRECTORY>:<USER>(<PASSWORD>:<PROJECTPASSWORD>)).

Below is an example of complete remote system information:

RONALD(PACK-ONE:MARY(XYZ:ACCOUNTS)).

Most parts of the identification have default values. For example a floppy disk device on a remote computer system RONALD is identified by RONALD.FLOPPY-DISC-1. Further information is found in the manual COSMOS User Guide (ND-60.163).

Several commands may be written on the same line. When these commands are processed, the BACKUP-SYSTEM will trace them by outputting the prompts, commands, and parameters.

4.4.1 Interactive Help Information

Detailed information about all commands, subcommands, and parameters is available interactively by the command:

DESCRIBE-ALL-COMMANDS (<output file>)

The default <output file> is your terminal. The output is quite long, and you may want to use a mass storage file or a printer as the <output file>.

Entering HELP, a question mark (?), or <ESCAPE> is useful in many situations. HELP (<command name>) is used to list commands or a subset of commands. HELP does the same for prompted subcommands also.

A question mark (?) given as the answer to a prompted command, subcommand, or parameter will display information. A question mark following an ambiguous command will list the commands matching the given command abbreviation. Information about a particular command or subcommand is displayed by entering the command name followed by a question mark.

When information requested by HELP or ? has been displayed, you will once more be prompted for the command, subcommand, or parameter. <ESCAPE> can be used to cancel a command or subcommand. If not given as an answer to a prompted subcommand or parameter, <ESCAPE> will return you to SINTRAN III.

4.4.2 Handling Volumes on Magnetic Tapes and Floppy Disks

A volume must be created on a sequential storage medium before files can be stored on it. Files stored on a volume may also be listed or deleted. The command to create a new volume is:

```
CREATE-VOLUME <volume name>,<device name>,(<device unit>)
```

The <volume name> has a maximum of six characters. The <device name> and <device unit> specify where the floppy disk is inserted, or where the magnetic tape is mounted. The <device name> may be on a remote system, for example, RONALD(FLOPPY-USER) FLOPPY-DISC-1. The <device units> are numbered from 0-3. If only one device unit exists, it is number 0.

After this command, files already on the floppy disk or magnetic tape specified will be unavailable. Only one volume may exist on a floppy disk or magnetic tape. A volume can contain files from many users. The first file on a volume may extend over several volumes.

Volumes will be written in the BACKUP-SYSTEM's default format unless the SERVICE-PROGRAM-CUF is used. All available volume formats produced by the BACKUP-SYSTEM will automatically be detected and handled properly when read. This also applies to volumes produced by the @COPY-USERS-FILES command in SINTRAN III version F and older versions.

The file names on a volume can be output by the command:

```
LIST-VOLUME <device name>,(<device unit>),(<file name>),
            (<output file>)
```

All file names matching the <file name> parameter will be output. No directory name can be used in the file name. The default <output file> is your terminal. The <device name> and <device unit> must describe where the volume is.

Files stored on a volume can be deleted. The command below will delete the specified file and the files following it.

```
DELETE-VOLUME-FILES <volume name>,<device name>,
                    (<device unit>),
                    (<generation of first file to delete>),
                    (<file name>)
```

Files cannot be deleted randomly, because a volume is a sequential set of files. The default value of <generation of first file to delete> is all file generations. Manual check is mandatory, i.e., you have to confirm that you want to delete the files by entering YES or NO from your terminal.

4.4.3 Copying a User's Files

One or more files can be copied from one mass storage medium to another. The command to copy one user's files is:

```
COPY-USERS-FILES (Destination type: Subcommand),
                 (Source type: Subcommand),
                 (Manual selection: Subcommand)
```

The destination type may be specified as DIRECTORY or VOLUME. The source type may be specified as DIRECTORY, VOLUME, or PARAMETER-FILE. The manual selection may be specified as YES, NO, LIST or SELECT. To copy files to a directory on a disk or floppy disk, you should use the subcommand:

```
DIRECTORY (<destination directory name>),
          (<destination user name>)
```

The <destination directory name> and <destination user name> must exist on a disk or floppy disk. The default values are your own user name and your default directory. A remote directory name, for example, RONALD.PACK-TWO, may be specified. The default user entered on the remote system is the destination user.

To copy files to a volume on a floppy disk or magnetic tape, you can use the destination type subcommand:

```
VOLUME <destination volume name>,<destination device name>,
       <destination device unit>,(<destination file generation>)
```

The <destination device name> and <destination device unit> describe where the magnetic tape is mounted or where the floppy disk is inserted. The <destination device name> may be remote. The <destination file generation> can be used to give the copied files a generation name of up to four characters. The file generation allows a set of files to be stored on the same volume more than once.

To copy files from a directory to any of the described destinations, you can use the source type subcommand:

```
DIRECTORY (<source directory name>),(<source user name>),
          (<source file name>)
```

The <source directory name> may be remote. The default user entered on the remote system is the source user. The <source user name> cannot be ambiguous. More than one user's files can be copied in one operation by another command, called MULTIUSER-COPY.

To copy files to a volume, you can use the source type subcommand:

```
VOLUME <source volume name>,<source device name>,
       <source device unit>,(<source file generation>),
       (<source file name>)
```

The <source device name> may be remote. The default (<source file generation>) is all generations.

A third source type subcommand can be used if the names of the files to be copied are stored on a file. This makes it possible to copy different users' files in one operation. The subcommand to use is:

```
PARAMETER-FILE <parameter file name>
```

The <parameter file name> must contain a list of file names or user names. The default file type is :SYMB. Only words in the parameter file which contain a left parenthesis, "(", are treated as file names. The other words are ignored, i.e., the output from the SINTRAN III command @LIST-FILES will be accepted. The files listed must reside on directories. The command has the same function as the COPY-USERS-FILES command with directory as source type.

If the parameter file includes user names, all files belonging to these users will be copied. The formats of strings accepted as user names are the ones identical to the output from the SINTRAN III command @LIST-USERS. That is, the strings starting with space, colon, space, directory name:user name, for example:

```
 : PACK-TWO:FLOPPY-USER
```

The COPY-USERS-FILES command accesses files by the normal SINTRAN III rules. However, user SYSTEM may access any user's files with the same access rights as the file owner, allowing files to be copied on behalf of the user.

You can select the files to be copied. To make the system list each file and wait for you to confirm copying, do this:

```
Ba-sy:'COPY-USERS-FILES ↵
   ... Destination and source type ...
  Manual Selection? YES ↵
```

Then you have to enter YES or NO for each file name listed on your terminal.

To copy all files without listing or manual confirmation, do the following:

```
Ba-sy: COPY-USERS-FILES ↵
   ... Destination and source type ...
   Manual Selection? NO ↵
```

To copy all files and list their names, but without manual confirmation, do this:

```
   Manual Selection? LIST ↵
```

You can make advanced selections as follows:

```
Ba-sy: COPY-USERS-FILES ↵
   ... Destination and source type ...
   Manual Selection? SELECT ↵
   Selection: (You make your selections) ↵
   ...
   Selection: EXECUTE ↵
   Manual file check? _
```

Various subcommands are available for making advanced selection of files in addition to the selections specified under source type. For example, you may copy all not allocated files modified since the last backup copy was taken.

The selection prompts allow you to specify various selection keys, and to use the logical operators AND, OR, NOT, and parentheses between the selection keys. The selection prompts allow you to specify a file name as a selection key by the command:

```
FILE-NAME (<file name>)
```

All files matching both the source type specification and the given <file name> are selected. No directory name is accepted.

To select only the files modified since the last backup was taken, user SYSTEM can use the command:

```
MODIFIED-SINCE-LAST-BACKUP
```

The command should be used with a log file to keep track of the backup copies. Log files are described in the SERVICE-PROGRAM-CUF's subcommand MASTER-LOG-MODE. The source type should be directory.

You can select a copying mode that only copies files if the destination file exists in advance. The destination type should be directory. The command to use is:

```
DESTINATION-FILES-EXIST
```

File attributes are indexed, continuous, allocated, peripheral, spooling, terminal, or temporary. You may select files by attributes. The default attribute is indexed. Use the command:

```
FILE-ATTRIBUTE (<attribute>)
```

You may select files modified in a specified time frame. If the source type is directory, a log file is required. If the source type is volume, the parameter <last date> is the date the file was copied to the volume. The command to use is:

```
WRITTEN-DATE-INTERVAL (<first date>),(<last date>)
```

The dates are entered as yy.mm.dd, for example, 84.12.31. The default is from the beginning of time to the end of time. The interval includes the specified dates.

You may select files that have been read in a specified time frame:

```
READ-DATE-INTERVAL (<first date>),(<last date>)
```

See WRITTEN-DATE-INTERVAL above.

Each file belonging to a user on a directory has an index number. The file index number is the number output in front of each file name in the SINTRAN III command @LIST-FILES. To select files by file index numbers, you can use the command:

```
FILE-INDEX-INTERVAL (<low file index>),(<high file index>)
```

The default <low file index> is 0, and the default <high file index> is the maximum index number used.

The interval includes the specified index numbers. On a volume, the sequence number of the file will be used as the file index.

File generations can be created on volumes. To copy an interval of source file generations, you can use the command:

```
GENERATION-INTERVAL (<low file generation>),
                    (<high file generation>)
```

The parameters have a maximum of four characters. The first parameter has no limit as its default value. The second parameter has no limit as its default value if the file generations are numeric. Otherwise, the low file generation is the default value. The interval includes the specified file generations.

The selections provided by the described commands can be combined by using the logical operators AND, OR, NOT, and parentheses. A couple of examples show how to use the logical operators:

```
Selection: FILE-NAME :DATA ↵
Selection: AND ↵
Selection: ( ↵
Selection: FILE-ATTRIBUTE INDEXED ↵
Selection: OR ↵
Selection: FILE-ATTRIBUTE CONTINUOUS ↵
Selection: ) ↵
```

Each line is terminated by carriage return, and "Selection:" will appear again. Selections may also be combined on one line, for example:

```
Selection: NOT ( FILE-NAME :TEXT AND FILE-INDEX-NUMBER 0,10 ) ↵
```

If you give two logical operators adjacent to each other in a meaningless sequence, for example, AND OR, the last will overrule the first. ANDs will be evaluated before ORs:

```
Selection: FILE-NAME A OR FILE-NAME B AND FILE-NAME C ↵
```

means

```
Selection: FILE-NAME A OR ( FILE-NAME B AND FILE-NAME C ) ↵
```

and not

```
Selection: ( FILE-NAME A OR FILE-NAME B ) AND FILE-NAME C ↵
```

When you have specified selections, you can get a list of all the files affected. Your terminal is the default <output file>. The command to use is:

```
LIST-FILES-SELECTED (<output file>)
```

You may also have all current selection keys listed on the terminal. The command to use is:

```
LIST-SELECTION
```

You may delete the complete selection or the last specified selection only. The commands to use are:

```
DELETE-CURRENT-SELECTION
DELETE-LAST-KEY
```

When you have completed the selection, you can stop the "Selection:" prompts by typing the command:

EXECUTE

You will then be prompted for <manual file check?> before the selected files are copied.

The parameter <manual file check?> must be answered by YES, NO or LIST. YES will cause a manual check of all files. NO and LIST will copy all files without asking for confirmation. LIST will output the names of all files copied.

The BACKUP-SYSTEM tries to set maximum access rights to the destination files before copying if the destination is not remote. That is, you cannot protect destination files against yourself or user SYSTEM by setting no write access.

Public users can only access their own volumes, or volumes owned by FLOPPY-USER. User SYSTEM, however, has both read and write access to all volumes. The BACKUP-SYSTEM may also be dumped in a copying mode where all users have access to any other user's volumes.

The <destination user name> may differ from the <source user name> when copying between directories. If the <destination user name> differs from the original owner of the file on a source volume, you will be asked which user should receive the copy. A new user name may also be specified.

When copying between directories, the <destination file name> may already exist. In that case, the source and destination dates for LAST OPENED FOR WRITE are checked. If the destination was written to later than the source, you will be asked whether you are copying in the right direction.

The user must ensure that enough space is available for all files to be copied. User SYSTEM may select a copying mode where the destination user's space will be expanded if necessary. All of the necessary file names will be created automatically. Only the default directory of a user will be accessed when no explicit directory name is given. Any directory may be accessed by stating its name explicitly.

The COPY-USER-FILES command will also copy the contents of the fields FILE ACCESS, LAST DATE OPENED FOR READ, LAST DATE OPENED FOR WRITE, CREATION DATE, and MAX BYTE POINTER. For user SYSTEM and for users with directory access to the source, the LAST DATE OPENED FOR READ and number of times OPENED will not be updated on the source file.

4.4.4 Copying Several Users' Files

The command COPY-USERS-FILES copies several users' files in one operation by specifying each file name or user name in a parameter file. More advanced copying facilities are offered by the command:

```
MULTIUSER-COPY (Destination type: Subcommands)
               (Source type: Subcommands)
               (<Manual user check?>)
               (Manual selection: Subcommands)
```

The subcommands are quite similar to those of the COPY-USERS-FILES commands, but lack the parameters related to destination user names. This section will only explain subcommands and parameters which differ from those of the COPY-USERS-FILES command.

The destination type subcommands in the MULTIUSER-COPY command are DIRECTORY or VOLUME. The subcommand DIRECTORY has only <destination directory name> as a parameter. The subcommand is restricted to user SYSTEM.

The users specified as source may not always exist on the destination directory. If you use the DESTINATION-EXPANSION command in the SERVICE-PROGRAM-CUF, the users will be created automatically. VOLUME is identical to that of COPY-USERS-FILES.

The source type subcommands in the MULTIUSER-COPY command are DIRECTORY, VOLUME or PARAMETER-FILE. The subcommand DIRECTORY has <source directory name> and <source user name> as its only parameters. The subcommand PARAMETER-FILE should contain file names or user names preceded by the string ":", i.e., space, colon, space, and possibly directory names. For example, the following will be recognized as a user name:

```
XX : PACK-TWO:P-HANSEN
```

The output from the SINTRAN III command @LIST-USERS will be accepted as a parameter file. To check which user's files will be copied, you have the parameter <manual user check?>. Possible answers are YES, NO and LIST. LIST is the default value.

The other subcommands and parameters in MULTIUSER-COPY are identical to those of COPY-USERS-FILES.

Note:

Only user SYSTEM may restore the source object entries unmodified. S/he does that by using the selection MODIFIED-SINCE-LAST-BACKUP, that is taking incremental backup. Otherwise, the open count and the date last read will be updated on the source.

4.4.5 Selecting Special Copying Modes

Various options for the commands COPY-USERS-FILES and MULTIUSER-COPY can be selected. To do this you must first enter:

SERVICE-PROGRAM-CUF (Cuf-serv: Subcommands)

A set of subcommands is available. The prompt used is CUF-SERV:. You return to the BACKUP-SYSTEM by the EXIT command. HELP, ?, and <ESCAPE> are available as before.

You change how the BACKUP-SYSTEM works by using different subcommands to be described later. User SYSTEM can make permanent modifications in the BACKUP-SYSTEM. The subcommand to use is:

DUMP-BACKUP-SYSTEM (<prog user name>)

The BACKUP-SYSTEM will be dumped on the file BACKUP-X:PROG which must exist in advance. "X" stands for the version. The <prog user name> must specify the user name where system :PROG files are normally stored. The user may be a remote user. Default is user SYSTEM. The @DUMP-PROGRAM-REENTRANT command ought to be given afterwards.

The information on a volume may be in different formats. A volume may also contain files in a mixture of formats. The three following commands will select copying formats. They will only affect output to magnetic tape. Some SINTRAN III file system information is copied together with the files in all formats.

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

The subcommand MODE-STANDARD-VOLUME selects the ANSI defined format. A hole in a file is a page not written to and not allocated space. Holes will be copied as empty pages. If MANUAL-STANDARD-VOLUME is used, copying of files with holes must be confirmed from your terminal.

The BACKUP-SYSTEM-VOLUME format will mark holes by a label instead of copying empty pages. This format can only be used with files to be handled by the BACKUP-SYSTEM. The BACKUP-SYSTEM is initially in this mode.

User SYSTEM may allow public users other than the owner to access a volume. This is done by the subcommand below, followed by the dumping of the BACKUP-SYSTEM to make the modification permanent. The parameter should be YES or NO. The default is NO.

SET-VOLUME-ACCESS (<general public access?>)

Continuous and allocated files may cause problems when copied. Such files cannot always be allocated in the way they are described by the file system information on the source directory or volume. In that case, you will be asked if the files should be stored in another way. If you answer YES, the following rules apply:

1) Allocated source files will be created as continuous files if possible, or else they will be created as indexed files.

2) continuous files will be created as indexed files.

If you answer NO, files will not be copied. To set a default answer to all such questions, you can use the subcommand:

SET-ALLOCATE-CREATE-DEFAULT (<default answer>)

The original terminal answer mode can be reset by using this command with <RETURN> as <default answer>. This facility is useful when copying many files in mode and batch jobs.

The normal search algorithm on a volume goes from beginning to end. All files matching the given <source file name> are copied. A special single search mode for volumes on magnetic tape may be switched on and off by the commands:

SET-SINGLE-SEARCH
RESET-SINGLE-SEARCH

The single search mode operates in the same way as the normal search, until one file or a group of consecutive files has been copied. Copying terminates at the first nonmatching file name. The search begins from wherever the magnetic tape is positioned. The tape is not rewound while in single search mode.

The single search mode 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 volume. Care must be taken when copying files to tape, if a single search is to successfully gather all the files a user wishes to retrieve.

For example, you may use the subsystem FILE-MANAGER. The FILE-MANAGER can produce a parameter file where the file names are sorted in different orders, for example, alphabetic. Files will then be stored on the volume in this order.

Information for all files copied by the BACKUP-SYSTEM can be stored on a log file. The information includes source, destination, date of copying, and which files are copied. The subcommands to use are:

MASTER-LOG-MODE <master log file>,(<append access?>)
USER-COPY-LOG-MODE <log file>,(<append access?>)

The master log mode is for user SYSTEM only. The user log mode is for public users only. Both commands must specify a log file where the information should be stored. The log modes are reset by giving <RETURN> as the log file.

The <append access?> question is answered by YES or NO. YES will cause the log information to be appended to the log file, instead of overwriting the old information on the file. YES is default. If the BACKUP-SYSTEM is dumped in the master log mode, the <master log file> must always be present when copying as user SYSTEM.

A <source file name> will normally mean all matching file names. A subcommand can be used to restrict this to identical file names only. The subcommand to demand exact matching in different cases is:

SET-MATCHING-MODE (<exact matching cases?>)

The legal values for <exact matching cases> are ALL, PAR, or NO. PAR will only demand exact matching of file names in parameter files. The default is exact matching in NO cases. If an empty file name or file type is searched for, all files will be accepted as in the normal matching mode.

The destination user area may sometimes be too small to hold the copied files. User SYSTEM may use a command to expand the user areas automatically:

DESTINATION-EXPANSION-MODE (<automatic expansion?>)

The answers are YES or NO. NO is default. This command affects output to directories only. When you use the MULTIUSER-COPY command, the destination users may not exist in advance. If the automatic expansion is selected, the user names will be created. The BACKUP-SYSTEM can be dumped to make this modification permanent.

A file may occupy more pages than needed to contain its data, for example, after text editing. Indexed files in destination directories may be shrank so that they do not exceed the MAX BYTE POINTER of the source file. The subcommand to use is:

SHRINKING-MODE (<shrinking?>)

The answer to the parameter should be YES or NO. The default is NO. Care should be exercised, since the MAX BYTE POINTER does not always give the last valuable byte of a source file.

There is a new subcommand under the SERVICE-PROGRAM-CUF command:

COPY-MODE <Special mode>

It lets you set one of the following special modes:

COPY:	The object entries of the files, that is, opened dates, access rights, etc., are not copied.
ARCHIVE:	The source files or their pages are deleted after copying.
OVERWRITE-INCREMENTAL:	The existing versions of the destination files should be overwritten.
NO-OVERWRITE:	Even if versions of the destination files exist, new file versions will be created.
CONTINUOUS-DESTINATION:	The destination files will be continuous even if the source files are not.
INDEXED-DESTINATION:	The destination files will be indexed even if the source files are not.

A more detailed description is obtained by pressing ? when you receive the prompt "Special mode: ".

An important development in version F is the possibility of archiving files that have not been used for a long time. Archiving means copying the source files to the backup disk, and deleting them or their pages on the source disk. That will save space on the source disk.

For instance, if you wish to archive files not used since 83.12.31, you may use the new subcommand COPY-MODE under the SERVICE-PROGRAM-CUF command to set the BACKUP-SYSTEM in the ARCHIVE mode.

```
@BACKUP-SYSTEM ↵
Ba-sy: SERVICE-PROGRAM-CUF ↵
   Cuf-serv: USER-COPY-LOG-MODE ↵
      Log file '' : ARCHIVED-FILES:LIST ↵
      Append access 'YES' : ↵
   Cuf-serv: COPY-MODE ↵
      Special mode '' : ARCHIVE ↵
   HAVE YOU SUFFICIENT BACKUP OF THE FILES TO ARCHIVE? YES ↵
   Cuf-serv: EXIT ↵
Ba-sy: COPY-USERS-FILES ↵
...
   Manual selection: SELECT ↵
   Selection: WRITTEN-DATE-INTERVAL ↵
      First date '' : 75.1.1 ↵
      Last date '' : 83.12.31 ↵
   Selection: AND READ-DATE-INTERVAL ↵
      First date '' : ↵  *
      Last date '' : 83.12.31 ↵
   Selection: EXECUTE ↵
```

* Note that the first date limit for read is 0 when not specified.

That will select the proper files to archive. All files last written to in the years 1975 to 1983 and read before 1984 (or never read) will be archived.

When you use the COPY-USERS-FILES command, as in this example, the source files will be deleted.

To have delete access to the files, one should normally be logged in as the source user. To have write access to the destination files, one should normally be entered as the destination user. Thus, the source and destination users should be the same. They should be on different directories, or the destination type should be VOLUME if the destination is a floppy disk or tape.

When user SYSTEM uses the MULTIUSER-COPY command to archive, the pages of the source files will be completely deleted, but the file names will remain. Thus the owner of the files will be able to see when the files where archived by using the @FILE-STATISTICS command.

4.4.6 Recreating Files and Users

There is a new command:

```
RECREATE-FILES-AND-USERS <destination directory name>,
        <parameter file name>, <manual user check?>,
        <manual file check>
```

You can use it to create files and users listed in a parameter file.

Public users may not create users. A parameter file cannot be used when copying from a volume. You may instead create all the destination files on empty users, and copy the volume by selecting DESTINATION-FILES-EXIST. Only those files for which the destination files already exist will then be copied.

Suppose you have a list of files to be copied from a volume to an empty user. This list is placed on a file called RECOVER-FILES:LIST. Then the BACKUP-SYSTEM is used as follows:

```
Ba-sy: RECREATE-FILES-AND-USERS ↵
  ...
  Parameter file name: RECOVER-FILES:LIST ↵
  ...
Ba-sy: COPY-USERS-FILES ↵
   Destination type: DIR ↵
   ...
   Source type: VOLUME ↵
   ...
   Manual selection: SELECT ↵
      Selection: DESTINATION-FILES-EXIST ↵
      Selection: EXECUTE ↵
   ...
```

4.5 Some Important Changes in the BACKUP-SYSTEM

The default destination directory is now the directory of the destination user given. Likewise, the default source directory is now the default directory of the source user given.

Check all old mode files so that this new version of the BACKUP-SYSTEM will not access the wrong directories!

4.6 Label Formats on Magnetic Tape Volumes

Implementation of magnetic tape volumes is based upon the 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- * [EOF1 OR EOV1] **

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

- All labels are 0 character blocks.

- All information in the labels is 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 204 character blocks. These blocks may contain any character (0-255).

$$$$ Deviation From Standard

- Only the first file on a volume may be extended to other volumes.

- A nonstandard 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 are not continuous, then a HOLE label defines where the next block on the magnetic tape logically belongs in the disk file. In order to represent a logical HOLE on the magnetic 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 continuous 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 of 204 characters, and HOLEs are labels. The contents of the HOLE labels are shown in parentheses.

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 may contain any alphanumeric characters. If the field is not fully filled with characters, the last character in the string is an apostrophe. This character is used to mark the end of the 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:

- An apostrophe is used to mark the end of the string. This character is not a part of the name. The unused part of a field is filled with spaces.

$ field 4:

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

$ field 7:

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

$ field 8:

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

$ fields 9 and 10:

- Creation and expiration dates are not used, and the fields 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	reserved for operating systems	35	(name of owner) $ & MAX BYTE POINTER)
51-52	7	(not used)	2	(spaces)
53-80	8	(not used)	2	(spaces)

$ field 6:

- Up to 16 alphanumeric characters, starting from position 16, identifying the owner of this file. If the name is shorter than 16 characters, an apostrophe is used to mark the end of name.

- 32-41 contains the MAX BYTE POINTER of the 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 to 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 to 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 BACKUP-SYSTEM and SINTRAN III file system.

Field 3:

POSITION WITHIN FIELD	POSITION WITHIN LABEL	CONTENTS
1-2	5-6	version number of this file (1-255)
3-4	7-8	total number of versions (1-255)
5-8	9-12	file system standard creation date
9-12	13-16	not used
13-76	17-80	SINTRAN III file system object entry

NONSTANDARD '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 WITHIN FIELD	POSITION WITHIN LABEL	CONTENTS
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)

5 LOOK-FILE

LOOK-FILE is a subsystem which enables a user to print data, modify data, and browse through the data contained in a file. The contents of different files may also be compared. The data contained in a file may be output as bytes, words or ASCII characters. Bytes and word may be output as octal, decimal, or hexadecimal values.

5.1 COMMAND SUMMARY

The available commands with their parameters are:

```
EXPLAIN-COMMAND <command>
HELP (<command>)
OPEN <file name>,(<block size>),(<access>)
CLOSE
DUMP (<block number>),(<from word number>),(<number of words>)
BYTE-DUMP (<block number>),(<from word number>),
          (<number of words>)
NEXT
PREVIOUS
SET-BLOCK-CONTENTS (<block number>),<value>
ZERO (<block number>)
COMPARE <file name>,(<first block number>),(<number of blocks>)
DEFINE-PRINT-FILE <file name>
ON-OFF-PRINTER (<1=on/0=off>)
MOVE <from file name>,<number of blocks to move>,
                <first block in source file>,<first block in dest. file>
SET-PRINT-FORMAT (<B=octal/H=hexadecimal/D=decimal>)
PATCH (<block number>),(<word number>)
SEARCH (<first block number>),(<number of blocks>)
CALCULATE <operand>,<operator>,<operand>
PROGRAM-INFORMATION
PROGRAM-STATUS
EXIT
```

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

5.2 GENERAL RULES

The subsystem may be entered by:

@LOOK-FILE

The available commands can be entered in the same way as SINTRAN III commands. Parameters which require a numeric value may be entered as decimal numbers, eg., 129D, or octal numbers, eg., 156B.

The subcommands will output the contents of a file. Each output line will include the following:

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 words output in the mode being used.
5. The 5 words as 10 ASCII characters.

A word is 16 bits. Any character whose ASCII value is less than 40B will be output as an ampersand (&).

5.3 DETAILED DESCRIPTION OF COMMANDS

This section describes the LOOK-FILE commands in detail. SINTRAN III commands can be executed by typing @ and the SINTRAN III command with parameters on one line.

EXPLAIN-COMMAND <command>

This command displays information about a command and its parameters. The <command> cannot be ambiguous.

HELP (<command>)

This lists all commands matching <command>. If no parameter is given, all commands will be listed.

PROGRAM-INFORMATION

This command displays general information about LOOK-FILE on the terminal, eg., its purpose, its command editing facilities, and its abbreviation rules.

OPEN <file name>,(<block size>),(<access>)

The command opens a file which will be used for further operations by other LOOK-FILE commands. The default block size is 512 words. The maximum allowed block size is 4096 words. Access can be R for read or W for write. The default is W.

CLOSE

The file specified in the OPEN command will be closed. An open print file will not be closed.

DUMP (<block number>),(<from word number>),(<number of words>)

The command displays the specified words from the open file. Use SET-PRINT-FILE to send the display to a file or to a printer. The optional output file is called a print file. The words will normally be displayed as octal numbers. This can be changed by the command SET-PRINT-FORMAT. The default <block number> is 0, the default value for <from word number> is 1, and the default value for <number of words> is 140. That amount of data fits most terminal screens.

BYTE-DUMP (<block number>),(<from word number>),(<number of words>)

This displays the specified words in the open file. The command SET-PRINT-FILE can be used to save a copy of the output on a file or write it to a printer. Each word will be displayed as two octal bytes. This can be changed by the command SET-PRINT-FORMAT. The default <block size> is 0, the default value for <from word number> is 1, and the default value for <number of words> is 120. That amount of data fits most terminal screens.

NEXT

The command displays information from the next block of the open file. The information may also be output to a print file. The amount of information output is determined by the <number of words> parameter in the DUMP or BYTE-DUMP command.

PREVIOUS

The command displays the previous block of the open file on the terminal. The information is optionally also output to a print file.

DELETE-PRINT-FILE <print file>

The specified <print file> will receive copies of the information output to the terminal by the commands DUMP, BYTE-DUMP, NEXT, SEARCH, etc. New files can be created by enclosing the file name in quotes ("..."). The output to the print file is switched on and off by the command ON-OFF-PRINTER.

ON-OFF-PRINTER (<1=on/0=off>)

This command switches output to the print file on and off. The default is off.

ZERO (<block number>)

All words in the specified block of the open file will be filled with binary zeros. The default block number is 0.

COMPARE <file name>,(<first block number>),(<number of blocks>)

This command compares the specified part of the <file name> with the open file. The block size given in the OPEN-FILE command is used. All differences will be output on the terminal, and optionally on a print file. The default <first block number> is 0, and the default number of blocks is 1.

MOVE <from file name>,<number of blocks to move>,
<first block in source file>,<first block in destination file>

This command moves the given number of blocks from the <from file name> to the open file.

SET-PRINT-FORMAT (<B=octal/H=hexadecimal/D=decimal>)

This command selects the output from the commands DUMP, BYTE-DUMP, NEXT and PREVIOUS to be octal, decimal, or hexadecimal. The default and initial printing format is octal.

PATCH (<block number>),(<word number>)

This command examines or modifies the open file. The address and the old value of the specified word are displayed. The value can be modified by entering a new value followed by <carriage return>. Just <carriage return> causes no change. The input value may be given as octal (B), decimal (D), or two characters ('AB'). The default is octal. The next words will be displayed until a period (.) is given. The default <block number> is 0, and the default <word number> is 1.

Some examples of how to give input when patching:

000001 (1)/000000 : ⏎		Return causes no change
000002 (2)/000000 : 'AA' ⏎		Change to AA (0405501B)
000003 (3)/000000 : 123 ⏎		Change to 000123B
000004 (4)/000000 : 123D ⏎		Change to 000173B
000005 (5)/000000 : . ⏎		Stop patching and write the block back.

The symbol ⏎ means carriage return.

SEARCH (<first block number>),(<number of blocks>)

The command searches for specified information in the open file. The information to be found may consist of up to 50 words. Each word may be given as octal (B), decimal (D), or as two characters ('AB'). The default is octal. Enter the information you want to search for as in the PATCH command. If the information is found in the open file, it will be output. You will then be asked if you want to continue searching. Answer by YES or NO. The default <first block number> is 0, and the default <number of blocks> is 1.

SET-BLOCK-CONTENTS (<block number>),<value>

All words in the specified block of the open file will be filled with the given value. The value must be prompted, ie., it cannot be given on the same line as the rest of the command. The value is given as octal (B), decimal (D), or two characters ('AB'). The default is octal.

CALCULATE <operand>,<operator>,<operand>

The command is used to perform simple calculations on octal or decimal operands. The default is decimal values. Legal <operators> are +, -, *, and /. The result is displayed in decimal and octal format.

PROGRAM-STATUS

The command displays information about the open file, the current block size, file access, and printing format.

EXIT

The command returns you to SINTRAN III. The open file will be closed.

6 FILE-EXTRACT

FILE-EXTRACT is a general purpose subsystem 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. You may define your output record layout in several ways. Also, a wide range of output environment choices are available.

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

6.1 PURPOSE

FILE-EXTRACT is a subsystem enabling 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 6.2.4.4)
- 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 6.2.1.1)
- Building or procedures to be processed with limited run time parameter input (see section 6.2.1.2)

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

6.2 COMMAND STRUCTURE

FILE-EXTRACT may be called from a terminal by:

@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.

6.2.1 Input File

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

6.2.1.1 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.

6.2.1.2 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 6.2.1.1) and then edited by QED or PED. Remember to remove tabs when in QED (command M TO(0)).

6.2.1.3 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 6.2.2.3)

— terminal output wait option specified (see section 6.2.4.6)

— line printer/terminal heading option specified (see sections 6.2.4.7, 6.2.4.8, 6.2.4.9 and 6.2.4.10)

6.2.1.4 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 6.2.4.5). 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 6.2.4.5.

6.2.2 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 6.2.4.6) 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.

6.2.2.1 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.

6.2.2.2 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 6.2.3) 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 6.2.2.1).

6.2.2.3 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 6.2.4). Note that X option switch to random file organization will be ignored when used together with certain other options (see section 6.2.1.3).

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 6.2.1.3).

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 recognizable version of the file, which could then be edited and finally reconverted to its orginal organization using the X option once again.

6.2.3 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 6.2.3.7)
- Specification of input record field values to be satisfied/not satisfied (see sections 6.2.3.1 and 6.2.3.2)
- Specification at text strings which are to occur/not occur within a record (see section 6.2.3.3)
- Specification of a text string which is to occur/not occur within a specified subset of a record (see section 6.2.3.4)

The selection criteria specified may be connected by the logical operands <.AND.> and <.OR.> (see section 6.2.3.5).

Finally, parentheses nesting on groups of selection criteria are allowed (see section 6.2.3.6).

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.

6.2.3.1 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 "in between" and "not in between".

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.

6.2.3.2 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 6.2.3.4).

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

Example:

45 — 50 = "OSLO 5"

6.2.3.3 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"

6.2.3.4 Limited Text String Search

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

6.2.3.5 Logical Operands

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

Together with the parentheses nesting (see section 6.2.3.6) 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 6.2.3.1, 6.2.3.2, 6.2.3.3 or 6.2.3.4 except for the input file record interval option as in section 6.2.3.7.

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"

6.2.3.6 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.

6.2.3.7 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 delimitor

,

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 6.2.2.2) split file records will be duplicated every time the input file is rewound.

6.2.3.8 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.

6.2.3.9 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 6.2.1.2) the first line may be specified beforehand, while the second may be used for additional operator selections at run time.

6.2.4 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 6.2.3).

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 makes the output record a copy of input record.

6.2.4.1 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, eg., produce a fixed record length file out of a variable length one.

6.2.4.2 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.

6.2.4.3 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.

6.2.4.4 Output Record Number Inclusion

The output record number inclusion option will produce an output record containing a successive 5 digit record numbering as it's first element.

Syntax:

<LO>[,]

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

6.2.4.5 Random Key Inclusion Option

The random key inclusion option will cause the input record's random address to be included as the first element of the output record.

Syntax:

R [,]

The random address consists of the following two elements:

1. Block number, a 5 digit block number at least containing the first character of the input record. Block size used is 512 words.

2. Byte number, a 4 digit number pointing to the beginning of the actual record within a given block.

This option may be useful for several purposes. It can be used to show where a record (group of records) exist within a file. Also, it may be used for more well defined functions. For example, FILE-EXTRACT may be run to produce an output file containing this random address together with subsets from input records to be used as SORT key. Then, this KEY file may be sorted. The resulting file may then be used as an INDEX file in order to process the input file in quite a different order, without actually having sorted the input file previously. Such an index file may be utilized by FILE-EXTRACT itself through the KEY file option (refer also to section 6.2.1.4).

This facility may have several advantages:

1. There may not be enough disk space to sort a bit input file itself.

2. A of a big input file may be very time consuming.

3. When an input file has to be accessed in many ways, this option will avoid the problems with keeping many copies of the same file.

Also, this option makes a limited input file sort possible by using the extract selection possiblity to output only those records interesting and then use the SORT utility to produce a suitable index file.

6.2.4.6 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. The default value is 24 for standard terminal screens.

,

is the specification delimiter in case of more parameters.

6.2.4.7 Line Printer or 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 6.2.4.8).

Syntax:

H [nn] [,]

where:

nn

is the number of lines per page. The default value is 24 which fits most terminal screens.

,

is the 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.
CUSTOMER ACCOUNT            N A M E
```

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 6.2.4.12), the heading should have all changes to be correctly specified.

6.2.4.8 Line Printer or 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. The default text is "PAGE".

Example:

PAGE = "SHEET:"

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

SHEET: 9999

HEADING LINE . . . . . .

DETAIL OUTPUT LINE 1 . . . . . .
DETAIL OUTPUT LINE 2 . . . . . .
. . . . . .
. . . . . .
. . . . . .

6.2.4.9 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 6.2.4.7) except it doesn't ask for heading input. Besides, the page numbering option (see section 6.2.4.8) will automatically be invoked.

6.2.4.10 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 6.2.4.9).

6.2.4.11 Split File Copy Option

Normally, the split file output (see section 6.2.2.2) 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 [,]

6.2.4.12 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 6.2.3.8). 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

6.2.4.13 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 6.2.1.2), the first line may be specified previously while the second one may be used for additional operator's choice at run time.

6.2.4.14 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 [,]

6.3 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 I= = = >----*-----I

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.

7 GENERAL PURPOSE MACRO GENERATOR - GPM

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 ouput 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 ND computers expects (and produces) characters with even parity. It may be called as a SINTRAN III subsystem. Program size is 1,5Kwords, while the rest of the virtual memory is used for a 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.

7.1 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 which have a special function in GPM:

↑ Precedes macro calls

; Ends macro calls

, Separates parameters in a macro call

\ 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.

7.2 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: ↑DEF, macro name, macro body;
Example:
↑DEF, A, <B\1>; defines macro A
to have B\1 (B and the first parameter) as its value. For instance, ↑A,5; yields the value B5.

Consider the definition of A in the following two examples:

1) ↑DEF, B, C; ↑DEF, A, ↑B; ; ↑DEF, B, D; ↑A;
2) ↑DEF, B, C; ↑DEF, A, <↑B;>; ↑DEF, B, D; ↑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: ↑A; yields C.

2) defines A equal to ↑B;. ↑A; is therefore equivalent to ↑B; which yields D. (Latest definition of B is valid!)
Hence: ↑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:
↑DEF, A, B; ↑A, ↑DEF, A, C; ; ↑A;

Temporary definition.

This string yields CB. Explanation:
↑DEF,A,B; defines A to have value B.
↑A, ↑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.
↑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: ↑VAL, macro name;
Example:
Suppose macro A has been defined by ↑DEF,A, <B\1>;
Then ↑VAL,A; yields B\1.

UPDATE

Updates macro definitions. Works in the same way as DEF. The new value must not be longer than the old value.
Format: ↑UPDATE, macro name, macro body;
Example:
Suppose A has been defined equal to B\1.
The call ↑UPDATE,A,<C\1>;
defines A equal to C\1.

BAR

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.
Format: ↑BAR, operator, binary number, binary number;

DECBIN

Performs decimal-to-binary conversion.
Format: ↑DECBIN, decimal number;

BINDEC

Performs binary-to-decimal conversion.
Format: ↑BINDEC, binary number;
Example:
↑DEF, SUM, <↑BINDEC, ↑BAR, +, ↑DECBIN, \1; , ↑DECBIN, \2; ; >;
defines a macro SUM which yields the decimal sum of its two parameters. For instance, ↑SUM,5,3; yields 8.

OCTBIN

Performs octal-to-binary conversion.
Format: ↑OCTBIN, octal number;
Example:↑DEF, CTR, <↑BAR, –, \1, ↑OCTBIN, 100; ; >;
defines a macro that yields control characters.
For instance, ↑CTR,A; yields A^c.

BINOCT

Performs binary-to-octal conversion.
Format: ↑BINOCT, binary number;

HD

Gives the first character of its argument (''head'').
Format: ↑HD, string;
Example: ↑HD, ABC; yields A.

TL — Gives all but the first character of its argument ("tail").
Format: ↑ TL, string;
Example: ↑TL, ABC; yields BC.

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

ICRMOD — Makes GPM ignore the characters "carriage return" and "line feed" in its *input* string. They may, however be used internally and be output.

CRMOD — Turns off the mode set by ICRMOD.

7.3 MACRO EVALUATION

According to rules a-e in Section 7.1, GPM works as follows:

Initially GPM is in *copying mode*.
When a macro call ↑N,P_1,P_2,---,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 \m, GPM enters the *parameter substitution mode* and replaces \m with p_M. The resulting string (the evaluated body with the actual parameters substituted for the formal ones) replaces the call ↑N,P_1,P_2, ---, 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 \2<LIC␣>↑$,\1;␣\0\13
\2 evaluates to PUB
<LIC␣> evaluates to LIC␣
↑$,\1; is equivalent to ↑$, MY; which evaluates to ENEMY
␣evaluates to␣
\0 evaluates to the evaluated macro name #
\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, ↑SUC,3;→ 4 ↑SUC,4;→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:

```
↑DEF, SUC, <↑1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ↑DEF, 1, <\>\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 ↑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 ↑ SUC,3;

7.4 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, ie.,

```
↑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.

7.5 RECURSIVE MACROS

```
↑DEF, A, <B↑A; >;
```

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.

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

Tells whether "general case" or "special case"

Covers the "general case"

Covers the "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:

```
↑DEF, RECUR,
<↑counter, ↑DEF, counter, <value X op ↑RECUR, counter-1; >; ↑DEF, 0, value Y; ; >;
```

"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.

↑FAC, n; → The value of 1. 2. 3. . . n =n!

Suppose that macros computing products and differences have been defined earlier and that their names are PROD and DIF. (For instance: ↑ PROD,2,3; →6 and ↑DIF,8,3;→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:

```
↑PROD,\1,↑FAC,↑DIF,\1,1;;;
```

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

```
↑DEF,\1,<↑PROD,>\1<,↑FAC,↑DIF,>\1<,1;;;>;
```

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 ↑FAC,0; → 0! = 1 the temporary definition that covers the special case simply is ↑DEF,0,1;

Now we may write the complete defintion of FAC:

```
↑DEF,FAC,<↑\1,↑DEF,\1,<↑PROD,>\1<,↑FAC,↑DIF,>\1<,1;;;>;↑DEF,0,1;;>;
L
```

n — "General case" expressing that n!=n.(n-1)! — "Special case" expressing that 0!=1

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 ↑DO,A,n; is equivalent to n calls of the parameterless macro A.

DO may be defined as follows:

```
↑DEF,DO,<↑\2,↑DEF,\2<↑>\ 1<;↑DO,>\1<,↑DIF,>\2<,1;;>;↑DEF,1,<>;;>;
```

↑DO,A,5; gives the same value as ↑A;↑A;↑A;↑A;↑A;
That a macro is parameterless does not necessarily mean that its value is constant, since it may call and redefine other macros.

7.6 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".
↑ADD,J,3; adds 3 to the value of "macro J" (which is updated) but the ADD macro leaves no value. ↑ 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.

↑&QUOTIENT, 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. $n \geqslant 0$.

↑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^c).

↑CHARACTER, octal, number;
Yields the corresponding character.
Example:
↑CHARACTER, 76; →>

↑ESC;
Yields an escape-character (33_8).

↑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)

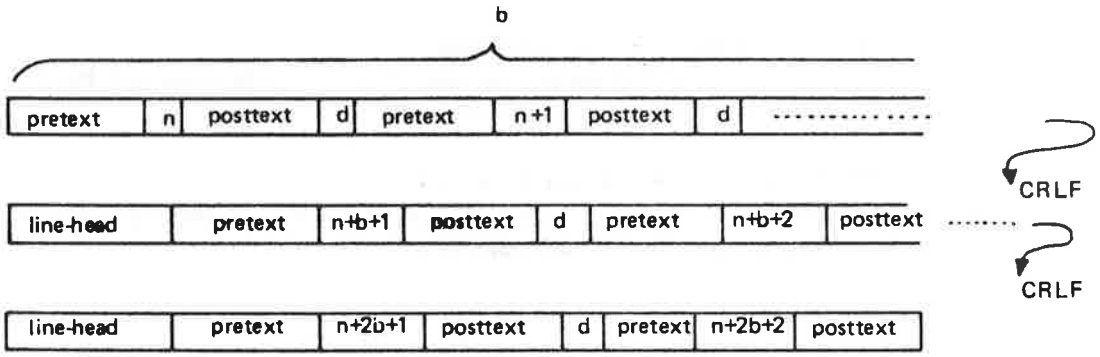
↑MASK, length, bitnumber;
Yields the bitmask. The length is given by the first decimal parameter, and the rightmost bit is given by the second, decimal parameter [0-15].
(Example: ↑MASK, 2, 1;→6)

↑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 [0-17$_8$].

↑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
```

7.7 GPM UNDER SINTRAN III

The GPM subsystem under SINTRAN III is called by writing:

```
@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_8 =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 accidently being specified as "output file" and consequently losing its contents completely.

7.8 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 preprocessor 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).

7.8.1 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 transformator 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.

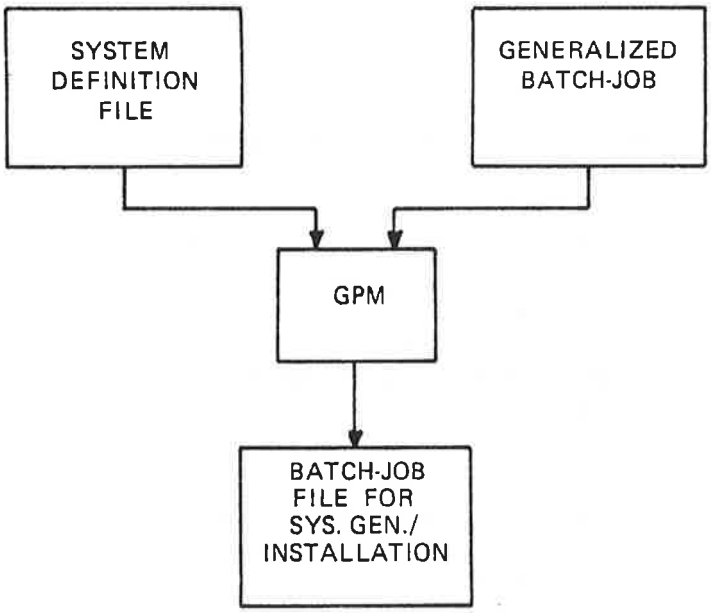
For further details, see the manual NORD PROCESS I/O, SOFTWARE GUIDE, (ND-60.093).

7.8.2 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;:BRF,,,,
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_8 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 get raving mad at all this GPM-stuff, and define NUPROG equal to 100 and run the system generation procedure. The inevitable 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
*)9END
*)9EOF
*"
*)LINE
```

7.9 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 macros 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
I
 ↑DEF,INTDEV1,01;
 ↑DEF,ASYNC,02;
 ↑DEF,PROCNR,03;
 ↑DEF,SEGNR,04;
L^C
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:

```
^CRMOD;
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
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
>
```

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
```

8 VTM-COMPOUND

SINTRAN III has to handle different manufacturers' terminals individually, ie., according to the terminal type specified in the command @SET-TERMINAL-TYPE. Some subsystems, eg., NOTIS WP, use the VIRTUAL TERMINAL MANAGER, abbreviated VTM, to handle the terminal screen.

VTM needs a table with descriptions of all terminal types used in the computer system. In VTM version C and later versions, the terminal table is stored on the file DDBTABLES-n:VTM. The character "n" denotes the version, eg., DDBTABLES-D:VTM.

VTM-COMPOUND is a subsystem to add new terminal types to the terminal table when new terminals are connected to the computer system. The subsystem may also modify the terminal table in other ways.

All standard terminal types are initially described in the terminal table. Each nonstandard terminal type will be described by a separate file called DDBnnn:VTM where "nnn" denotes the terminal type. Such files should be compounded into the terminal table by VTM-COMPOUND.

VTM-COMPOUND may also produce a terminal table on a relocatable format file to be loaded together with an application program which uses VTM. The terminal type descriptions will then be fetched from this file.

8.1 STARTING VTM-COMPOUND

VTM-COMPOUND uses menus to show the available operations on the terminal table. You enter the first menu by giving:

@VTM-COMPOUND

You return to SINTRAN III by selecting the EXIT alternative in the menu, ie., by typing 9 followed by carriage return. Standard line editing characters like the a-key (or CTRL A), the EXPAND-key (or CTRL E), and the navigation keys to move the cursor forward and backward are available.

8.2 THE OPERATIONS AVAILABLE IN THE MENUS

The first menu you enter is the main menu. The available operations on the terminal table look like this:

```
DO YOU WANT TO:
  1:   GENERATE A NEW FILE
  2:   ADD TERMINAL TYPES
  3:   DELETE TERMINAL TYPES
  4:   GENERATE A FILE WITH BRF OR NRF FORMAT
  5:   LIST TERMINAL TYPES
  6:   LIST CPU-TYPE, CPU-NUMBER AND FILE VERSION NUMBER
  7:   CHANGE CPU-TYPE, CPU-NUMBER AND FILE VERSION NUMBER
  8:   EDIT THE CONTENTS IN DDB999:VTM
  9:   EXIT
  ANSWER: _
```

You select an alternative from this menu by entering one of the numbers followed by carriage return. In alternatives 1-7 you will be asked which file you want to use. You may select one of four alternatives as shown from the menu:

```
WHICH FILE DO YOU WANT TO USE:

  1: DDBARRAYS:VTM          (VTM-B)
  2: DDBTABLES-n:VTM        (VTM-n)
  3: User's choice          /:VTM/
  4: RETURN
  ANSWER: _
```

If you select alternative 2, "DDBTABLES-" will be output, and you have to fill in the correct version. The file DDBARRAYS:VTM contains the terminal table used by VTM, version B. Alternative 3 allows you to specify any file. A new file can be created by enclosing the file name in quotes ("..."). The default file type, :VTM, is shown between slanted lines.

You will then be asked for further information, or information will be ouput according to the selected alternative in the main menu. This is described in the next sections. The 4th alternative called RETURN redisplays the main menu.

8.2.1 Generate a New File

Alternative 1 in the main menu will create a new file containing a terminal table. VTM-COMPOUND will ask you to enter the CPU type, CPU number, and the file version number. Answer these questions with carriage return. These functions are reserved for future use.

The subsystem will then ask which terminal types the user wants to compound into the new terminal table. The terminal types can be given either one by one separated by carriage return, or as a range, eg., 2:5. You finish by typing 777 followed by carriage return. The corresponding DDBnnn:VTM files describing the terminal types must be available.

8.2.2 Add Terminal Types

Alternative 2 in the main menu will add new terminal types to the terminal table. Terminal types can be given one by one separated by carriage return, or as a range, eg., 2:5. You finish by typing 777. The corresponding DDBnnn:VTM files must be available.

8.2.3 Delete Terminal Types

Alternative 3 in the main menu will delete terminal types from the terminal table. Terminal types can be given one by one separated by carriage return, or as a range, eg., 2:5. You finish by typing 777.

8.2.4 Generate a New File with BRF or NRF Format

Alternative 4 in the main menu will create a file containing the terminal table in relocatable format, ie., in BRF or NRF. These files can be loaded together with an application program which uses VTM. VTM will then fetch the terminal type descriptions from this file. The relocatable format files may be generated for ND-100 one-bank programs, ND-100 two-bank programs, or ND-500 programs. The alternatives will be shown as in the menu below:

```
WHAT TYPE DO YOU WANT TO GENERATE:
  1:   ND-100  (1-BANK)  /:BRF/
  2:   ND-100  (2-BANK)  /:BRF/
  3:   ND-500            /:NRF/
  4:   RETURN
  ANSWER: _
```

When you select an alternative, the default file name will be displayed. You may edit this name, eg., enclose it in quotes (''...'') to create a new file, or overwrite it with another file name. The default file names are VTM-1B-ARRAY:BRF, VTM-2B-ARRAY:BRF, and VTM-ARRAY:NRF.

The information in the relocatable format files will be the same as in the ordinary terminal table. The file containing the ordinary terminal table will be used as input.

8.2.5 List Terminal Types

Alternative 5 in the main menu will display a list of the terminal types in the terminal table. The terminal types will be output both by number and the manufacturer's name, eg.,

```
 2:  TELETYPE-ASR-33
 3:  TANDBERG-TDV2115-STANDARD
 4:  INFOTON-200-1
 5:  INFOTON-400
11:  DEC-LA36 (DECWRITER-II)
36:  TANDBERG-TDV2215-EXTENDED
52:  TANDBERG- TDV-2215-SDS-V2
53:  TANDBERG-TDV2200/9-ND-NOTIS
57:  FACIT- 4420-ND-NOTIS
83:  TANDBERG-TDV2200/9-V2-ND-NOTIS
```

The terminal types listed above are the standard terminal types.

8.2.6 List CPU Type, CPU Number and File Version Number

Alternative 6 in the main menu should display a list of the CPU type, the CPU number, and the file version number in use. These functions are reserved for future extensions, and no values will be output.

8.2.7 Change CPU Type, CPU Number and File Version Number

Alternative 7 in the main menu allows you to change the CPU type, the CPU number, and the file version number. These functions are reserved for future use. The default values are no changes.

8.2.8 Edit the Contents of the File DDB999:VTM

The DDB999:VTM file is only used with VTM, version A. It contains the terminal types and the manufacturers' terminal names of all terminals in the computer system. The DDB999:VTM file is used by some subsystems, eg., NOTIS WP, to display the available terminal types if no terminal type is set. Section 8.3 explains how VTM version A functions.

The DDB999:VTM file should be updated when new terminal types are added to the system. Alternative 8 in the main menu will allow you to modify the contents of this file. The alternative operations will be shown as in the menu below:

```
DO YOU WANT TO:
  1:   MAKE A NEW DDB999 FILE
  2:   ADD TERMINAL TYPE DESCRIPTIONS
  3:   DELETE TERMINAL TYPE DESCRIPTIONS
  4:   LIST THE CONTENTS IN DDB999:VTM
  5:   EXIT
  ANSWER: _
```

The first alternative will create a new DDB999:VTM file. The second alternative will add new terminal types to the existing DDB999:VTM file. The 3rd alternative will delete terminal types from the file. The terminal types should be entered one by one separated by carriage return, or as a range, eg., 2:5. You terminate the input by typing 777.

The 4th alternative will list the terminal types in the file. The terminal type and the manufacturer's terminal name are output, eg., 53: TANDBERG-TDV2200/9-ND-NOTIS. The last alternative will display the new contents of the file DDB999:VTM and return you to SINTRAN III.

8.2.9 Exit

Alternative 9 in the main menu will return you to SINTRAN III. Information about the terminal types in the terminal table will be output if the table is modified.

8.3 VTM VERSIONS, FILE VERSIONS AND TERMINAL TYPES

To some extent the different versions of VTM should be handled individually. Version B of VTM uses the terminal table contained in the file called DDBARRAYS:VTM. Version C and later versions use the file DDBTABLES-n:VTM where "n" denotes the version.

VTM version A, uses one DDBnnn:VTM file for every terminal type. No compounded DDBTABLES-n:VTM or DDBARRAYS:VTM files exist. VTM-COMPOUND serves no purpose for VTM version A, except editing the file DDB999:VTM. This file contains a list of the terminal types in the computer system, but no terminal type descriptions are included.

The files DDBnnn:VTM describing each nonstandard terminal type are separate products. All terminal types are listed in appendix B in the SINTRAN III REFERENCE MANUAL (ND-60.128). VTM- COMPOUND should only be used from a standard terminal type.

8.4 AN EXAMPLE OF INCLUDING A NEW TERMINAL TYPE

In the following example, the nonstandard terminal type 78 is entered into the terminal table. The terminal table resides on the file DDBTABLES-D:VTM. User input is underlined.

```
@VTM-COMPOUND
```

(main menu displayed and the choice ADD TERMINAL TYPES is selected)

```
ANSWER: 2
```

(a new menu asks which file you want to use)

```
ANSWER: 2
```

(the string "DDBTABLES-" is displayed to the right of alternative 2 in the menu)

```
 2:   DDBTABLES-n:VTM (VTM-n)      DDBTABLES-D
```

(the menu disappears and you are asked which terminal types you want to add)

```
                    --  WRITE TERMINAL TYPES  --
CARRIAGE RETURN AFTER EACH TYPE. WHEN FINISHED:  WRITE 777.

78
NEXT: 777
```

(you return to the main menu)

```
ANSWER: 9
```

(all terminal types are listed and you return to SINTRAN III).

The terminal type 78 can now be set for the new terminal by the SINTRAN III command @SET-TERMINAL-TYPE. The file DDB078:VTM containing the terminal type description must be available.

8.5 ERROR MESSAGES

The following error messages can be displayed by VTM-COMPOUND. Other possible messages are SINTRAN III error messages like "NO SUCH FILE", "ambiguous file name", etc.

THE FILE IS EMPTY

You tried to access terminal types in an empty terminal table or an empty DDB999:VTM file. VTM-COMPOUND will ask you to specify another file name.

TYPE > 255 IS ILLEGAL

You entered a terminal type greater than 255. VTM-COMPOUND will ask you for another terminal type.

FILE IS TOO BIG FOR BUFFER

The number of bytes in the DDB999:VTM is to great. VTM-COMPOUND will ask you to specify another terminal type.

ALREADY EXISTS IN THE FILE

You tried to add a terminal type that already exists in the terminal table or the file DDB999:VTM. VTM-COMPOUND will ask you to specify another terminal type.

DOES NOT EXIST IN THE FILE

You tried to delete a terminal type that does not exist in the file specified. VTM-COMPOUND will ask you to specify another terminal type.

THERE ARE NO MORE TERMINAL TYPES LEFT IN THE FILE

All terminal types have been deleted from the terminal table or the file DDB999:VTM. VTM-COMPOUND will return you to the main menu.

THE TYPE DOES NO EXIST

There is no DDBnnn:VTM file corresponding to the terminal type you chose. VTM-COMPOUND will ask you to specify another terminal type.

NOT A RANGE!

An improper range is given, eg., 36:34 instead of 34:36.

THE FILE "VTM-ALL-TYPES:VTM" DOES NOT EXIST ON THIS USER

No file by the name of VTM-ALL-TYPES:VTM exists. This file contains the manufacturers' names of all the defined terminal types in SINTRAN III. This file is used when terminal types are added to one of the files DDB999:VTM, DDBARRAYS:VTM or DDBTABLES-n:VTM. The file VTM-ALL-TYPES:VTM is supplied as part of the VTM-COMPOUND system. VTM-COMPOUND will terminate.

THE FILE "DDB999:VTM" DOES NOT EXIST ON THIS USER

No file by the name of DDB999:VTM exists. VTM-COMPOUND will terminate, and the user have to create this file before continuing.

THE FILE <file name> DOES NOT EXIST ON THIS USER

No file with the entered <file name> exists. VTM-COMPOUND will continue and ask for another file name.

********** 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: SINTRAN III Utilities Manual

Manual number: ND - 60.151.2 Rev. A

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 ? _____

_____

NOTE!
This form is primarily for documentation errors. Software and system errors should be reported on Customer System Reports.

Send to:
Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Norsk Data's answer will be found on reverse side

Answer from Norsk Data

Answered by Date

Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
0621 Oslo6, Norway

Systems that put people first