SINTRAN III Utilities Manual

ND-60.151.3 EN

Norsk Data

SINTRAN III
Utilities Manual

ND-60.151.3 EN

Version 2	*June 1984*
Version 2A	*February 1985*
Version 3	*June 1988*

Send all documentation requests to:

*Norsk Data A.S
Graphic Centre
P.O. Box 25 – Bogerud
N-0621 Oslo 6
NORWAY*

Preface

THE PRODUCTS

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

PERFORM LOOK-FILE	part of Subsystem Package number ND 210005U
JEC	ND 210534B

PERFORM is a simple macro processing system to create mode and batch files; LOOK-FILE is used to inspect and modify files; JEC stands for Job Execution Control, and is used to control the execution of mode and batch jobs.

THE READER

This manual is intended for users of SINTRAN III. The subsystems documented here are not necessary for simple use of the operating system, but may be of considerable use for particular tasks. Familiarity with SINTRAN III on a public user level is the only previous knowledge assumed.

RELATED MANUALS

The SINTRAN III User Guide, ND-60.264, contains basic information about the SINTRAN operating system.

NOTATION USED IN THIS MANUAL

In the examples, user input is underlined. Most examples are written in uppercase, but lowercase is also accepted. When used as parameters, octal numbers are given in the form 377B, where B denotes octal.

In command parameter descriptions, parameters are enclosed in angular brackets, e.g. <parameter>. Parameters which have default values are enclosed in parentheses, e.g. (<parameter>). The default value is used if a parameter is omitted. Selections in parameter descriptions are separated by slashes, e.g. YES/NO.

CHANGES FROM THE PREVIOUS VERSION OF THIS MANUAL

This version of the SINTRAN III Utilities manual is considerably shorter than previous versions. The following products are no longer documented in this manual:

MAIL (see SINTRAN III COMMANDS Reference Manual, ND-60.128, and SINTRAN III System Supervisor, ND-30.003).

BACKUP-SYSTEM (see BACKUP User Guide, ND-60.250).

FILE-EXTRACT (see File Handler User's Manual, ND-60.175).

VTM-COMPOUND (see the Product Description sheets for VTM Terminal Tables (standard), product number ND 210455).

The remaining chapters contain only minor alterations from the previous version of the manual.

CHAPTER 1 PERFORM

Mode or batch files are used to execute sequences of commands that are used repeatedly. PERFORM gives you greater flexibility when using mode and batch files by allowing parameter substitution.

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 in a file. A predefined library of macros is stored in the file PERFORM-LIB:MCRO.

1.1 Creating Macros

A few simple directives, starting with a circumflex (^), are used to define a macro. All directives must end with a semicolon (;). A macro will have a macro head and a macro body in the following manner:

```
^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 uppercase letters, digits, or hyphen (-). The directive ^E; ends the macro. All user-defined macros are normally stored consecutively in one file.

The directive ^; separates the macro head from the macro body.

The other directives that may 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 for 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. 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 information will be displayed on the terminal when processed by PERFORM.
^C,<comment string>;	Comment. This will be ignored by PERFORM.

The numbers n must be consecutive and in the range 1 - 20. These numbers must be preceded by a reverse slash (\) in the macro body wherever a parameter from the terminal is to be inserted.

Here is a simple example:

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

When PERFORM processes the macro in this example, 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.

If you want to use the \ character to mean something other than a PERFORM parameter, you must indicate this by writing two consecutive reverse slashes. PERFORM will replace these with a single reverse slash and not make a parameter replacement.

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.

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 circumflex (^) is used.

1.2 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 for. The <macro file> is the file containing the macro with the specified <macro name>. The default <macro file> is PERFORM-LIB:MCRO and default file type is MCRO. The first macro in the specified file is the default <macro name>.

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

1.3 Example of Using Perform

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

```
(Other macros in the same file)

^B,FTNRUN;
^L,MACRO TO COMPILE, LOAD, AND EXECUTE A FORTRAN 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 in 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 in your user area until it is overwritten by another execution of PERFORM.

1.4 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 for. Then type a "?", and all macros in 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:
```

After this, PERFORM will once more prompt for the <macro name> to be used.

1.5 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. 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>,

Default file type is :MODE. This is necessary if the mode job is 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 in 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>.

1.6 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.

1.7 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 for 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 reverse slash (\) 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 chapter 3 of this manual.

1.8 Predefined Macros

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

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

You can find more detailed information about each macro by inspecting the file, PERFORM-LIBRARY:MCRO, using an editor.

CHAPTER 2 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 words may be output as octal, decimal, or hexadecimal values.

2.1 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-CONTENT (<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 the other commands.

2.2 General Rules

The subsystem may be entered by:

@<u>LOOK-FILE</u>

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 (e.g. 129D), or octal numbers (e.g. 156B).

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

- The word number in decimal
- The word number in octal
- A single character indicating the mode being used for the current line, i.e. B for byte and W for word
- 5 words output in the mode being used
- 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 (&).

2.3 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, e.g., 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. If another file has already been opened by this command, this file will be closed. 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. 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, on the terminal. Use DEFINE-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. Default <block number> is 0, default value for <from word number> is 1, and 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 from the open file on the terminal. The command DEFINE-PRINT-FILE can be used to save a copy of the output on a file or write it to a printer. Each 16-bit word will be displayed as two octal bytes. This can be changed by the command SET-PRINT-FORMAT. Default <block size> is 0, default value for <from word number> is 1, and 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, on the terminal. The information may also be output to a print file using DEFINE-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 may also be output to a print file using DEFINE-PRINT-FILE.

DEFINE-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, PREVIOUS, SEARCH, and COMPARE. 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. Default is off.

ZERO (<block number>)

All words in the specified block of the open file will be filled with binary zeros. 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 using the DEFINE-PRINT-FILE command. Default <first block number> is 0; default number of blocks is 1.

MOVE <from file name>,<number of blocks to move>,
<first block number in source file>,
<first block number 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 print format for the output from the commands DUMP, BYTE-DUMP, NEXT and PREVIOUS to be octal, decimal, or hexadecimal. 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 <RETURN>. Just <RETURN> causes no change. The input value may be given as octal (B), decimal (D), or two characters ('AB'). Default is octal. The next words will be displayed until a period (.) is given. Default <block number> is 0 and 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.
```

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'). 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 to the terminal, or to a print file if you use the DEFINE-PRINT-FILE command. You will then be asked if you want to continue searching. Answer by YES or NO. Default <first block number> is 0, and default <number of blocks> is 1.

SET-BLOCK-CONTENT (<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 for, i.e., 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'). Default is octal.

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

The command is used to perform simple calculations on octal or decimal operands. 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.

CHAPTER 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 26.)
- You may execute nested mode files that have a return status showing whether they executed successfully or not. (See page 27.)
- You may use arithmetic. (See page 29.)
- 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 43. You may use your own variables in SINTRAN commands, as parameters to your own programs, as loop counters, or in arithmetic expressions. (See page 28.)
- Answer "questions" asked by the mode file. (See page 30.)
- You may make conditional tests, based on the values of the completion code, the SSI code, or the status code. (See page 33.)
- You may make conditional tests, based on the day, date, or month you execute your mode file. (See page 33.)
- Jump backwards and forwards to numeric labels defined in your

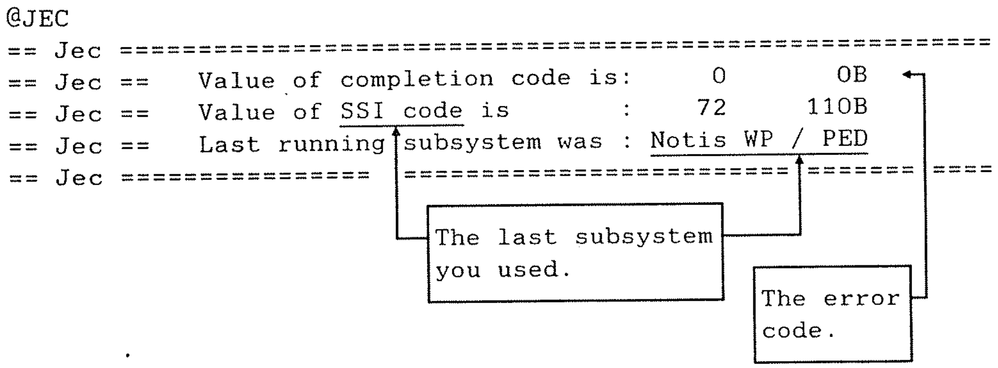
batch or mode file. (See page 32.)

- Create loops so that things can be repeated a certain number of times. (See page 36.)
- Give input from your terminal to programs you execute in mode jobs. (See page 37, Section 3.3.)
- You may turn communication with your terminal on and off in a mode job. (See page 37.)
- You may send output to your terminal, an output file, or both. (See page 37.)
- 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 41.

If you type your mode files in NOTIS-WP, make sure they are in 7-bit or 8-bit format, not in 16-bit format.

3.1 Interactive JEC and Error Codes

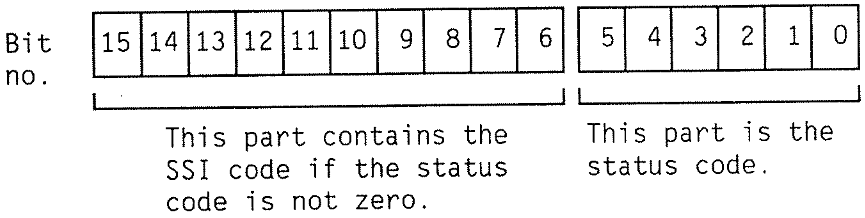
Type @JEC in SINTRAN and you should see something like this:



```
@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 numbers you get will most likely not be the same.)

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


Bit no.
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
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 which 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 III Commands 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	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". If you now type JEC, the following will appear:

```
== 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, 0, means that this is a SINTRAN File-System error. If you look in the SINTRAN III Commands 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.

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 can type a statement like this in a JEC mode file:

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

This will stop the mode file execution if any errors occur.

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	=	Probably informative messages
30-47B	=	Probably error conditions
50-76B	=	Error conditions
77B	=	Fatal error

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

The following problems may typically arise during a mode job:

- 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 skip 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 as few COBOL programs as you want to:

```
@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 MESSAGE                     %Sends messages to terminal even if
                                  %not chosen as output destination
@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:

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 the 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 33.

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.
```

DEFINE AND INQUIRE

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

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>
```

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, arithmetic expressions are allowed. Use +, -, *, and / to add, subtract, multiply, and divide. **NOTE - Always precede and follow the signs +, -, * or / with a blank.** It not only looks nicer, it is the only thing allowed! Extra blanks are allowed.

Do not multiply or divide by JEC variables such as DAY. DAY is explained on page 33. 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 the 50 files you have. You do this 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 deleted:

```
(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 could 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 use the á key to erase any typing errors. JEC accepts the same control characters for editing as SINTRAN.

GO TO, IF, FOR, END-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 labels which are easier to understand, 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 43.
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>
```

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 characters, 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:

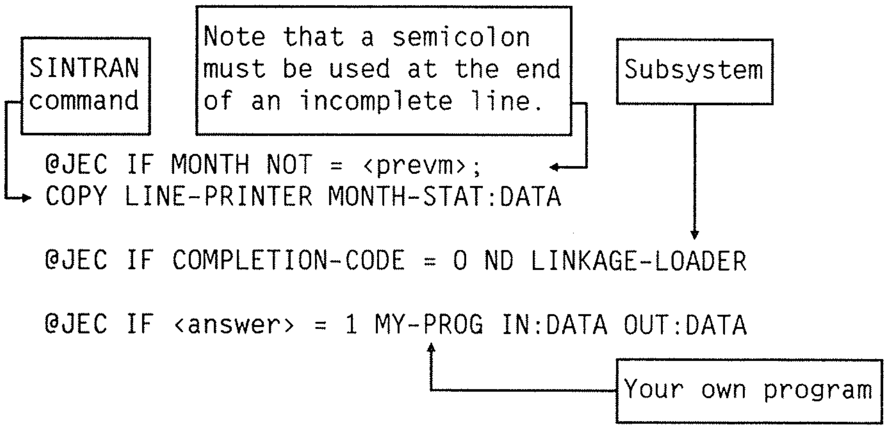
```
@JEC IF COMPLETION-CODE > OB TERMINATE
@JEC IF (SSI-CODE = 6B AND STATUS-CODE < 20B) GO TO 100
@JEC IF (DAY < 8 AND DAY = 'MONDAY') GO TO 100
@JEC IF (DAY = 20 AND (NOT DATE = 84.01.20)) GO TO 100
@JEC IF <answer> = 2 GO TO 2000
@JEC IF <answer> NOT > 0 GO TO 3000
@JEC IF COMPLETION-CODE = 0 BRF-LINKER

@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>.


SINTRAN command
Note that a semicolon must be used at the end of an incomplete line.
Subsystem
Your own program

```
@JEC IF MONTH NOT = <prevm>;
COPY LINE-PRINTER MONTH-STAT:DATA

@JEC IF COMPLETION-CODE = 0 ND LINKAGE-LOADER

@JEC IF <answer> = 1 MY-PROG IN:DATA OUT:DATA
```

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.

NOTE: 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 24.
Nested loops are also allowed.

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
          1987        12          24        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>'
```

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. Anything 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.
```

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.

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
```

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
```

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 > OB AND SSI-CODE < 4B);
DELETE-FILE OUTSALARY:DATA   %Split JEC command
@CC SSI code < 4B INDICATES FILE SYSTEM ERROR
@JEC END
<CTRL O> <ESCAPE> <CTRL O> <ESCAPE>
```

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
```

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.

<u>BEGIN</u>

<u>CLEAR-COMPLETION-CODE</u>

<u>DEFINE <identifier></u> % up to 40 ASCII characters long

<u>DEFINE <identifier></u>=numeric literal

<identifier>=<identifier>

```
END


                      ┌                            ┐
                      │<identifier>  <identifier>  │
FOR <identifier> IN   │             :              │ DO
                      │integer       integer       │
                      └                            ┘
      ┌                     ┐
      │ WHILE <condition>   │
      └                     ┘
END-FOR


GO TO ┌ numeric label ┐
      └               ┘

                      ┌                                                   ┐
                      │ GO TO ┌ numeric label ┐                           │
                      │       └               ┘                           │
                      │ TERMINATE                                         │
IF <condition> THEN   │                      ┌THRU   ┐                    │
                      │ PERFORM numeric label│THROUGH│ num. label         │
                      │                      └       ┘                    │
                      │ program name / SINTRAN III command                │
                      └                                                   ┘

INQUIRE <identifier>  ┌'string of ASCII characters
                      └
                                         and/or <identifier>'┐
                                                             ┘

MESSAGE 'ASCII string  and/or <identifier>'

MODE-INPUT

MODE-OUTPUT
```

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


            [                [ [THRU   ]                 ] ]
PERFORM     [ numeric label  [ [THROUGH]  numeric label  ] ]


PRINT-COMPLETION-CODE

PRINT-DATE

TERMINAL-INPUT

TERMINAL-OUTPUT

TERMINATE

WAIT-FOR-CR 'ASCII string'

% Comments in the mode/batch file
```

Table of contents