SINTRAN III
Monitor Calls

ND-60.228.1 EN

Norsk Data

SINTRAN III
Monitor Calls

ND-60.228.1 EN

PRINTING RECORD	
PRINTING	NOTES
07/86	Version 1 EN

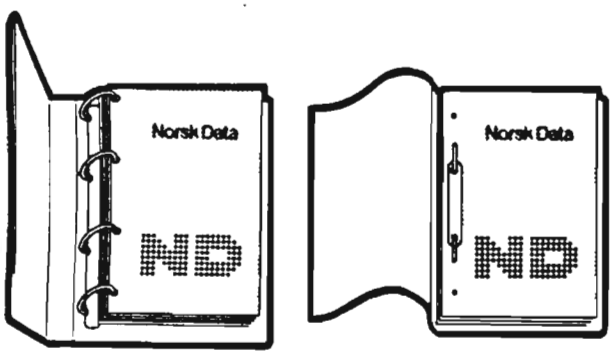
SINTRAN III Monitor Calls
Publ.No. ND –60.228.1 EN

UPDATING

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

New versions and revisions are announced in the ND Customer Support Information and can be ordered from the address below.

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



RING BINDER OR PLASTIC COVER

The manual can be placed in a ring binder for greater protection and convenience of use. Ring binders may be ordered at a price of NKr. 45.- per binder.

The manual may also be placed in a plastic cover. This cover is more suitable for manuals of less than 100 pages than for larger manuals.

Please send your order, as well as all types of inquiries and requests for documentation to the local ND office, or (in Norway) to:

Norsk Data A.S
Graphic Center
P.O.Box 25 BOGERUD
N-0621 OSLO 6 - Norway

I would like to order

..... Ring Binders, 40 mm, at NOK 45.- per binder

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

Name: .....

Company: .....

Address: .....

PREFACE

THE PRODUCT

This manual is related to the following products:

SINTRAN III version J	ND-210174, ND-210575, ND-210576
ND-500 MONITOR version F	ND-210333
MONITOR CALL PACKAGE version A	ND-210913, ND-210914

The products provide programs in various languages with operating system functions.

THE READER

This manual is intended for programmers.

PREREQUISITE KNOWLEDGE

You need knowledge of programming in general. Some of the monitor calls require knowledge of SINTRAN III and ND-500-MONITOR.

THIS MANUAL

The main part of this manual is an alphabetic list of SINTRAN III monitor call descriptions. The first chapter shows examples of how to use monitor calls from various programming languages. The second chapter helps you to find the correct monitor call for a particular task. The last appendix contains a glossary.

RELATED MANUALS

The following manuals may be of interest:

SINTRAN III TIMESHARING GUIDE (ND-60.132)
SINTRAN III REAL TIME GUIDE (ND-60.133)
ND-500 LOADER/MONITOR (ND-60.136)

You also need the manuals for the programming language and the loader you want to use.

CONTENTS

1 MONITOR CALLS

SINTRAN III is the operating system on all ND computers. It provides various services such as reading the current time or writing to files. Programs request such services through monitor calls. A monitor call looks like a routine call. See figure 1.

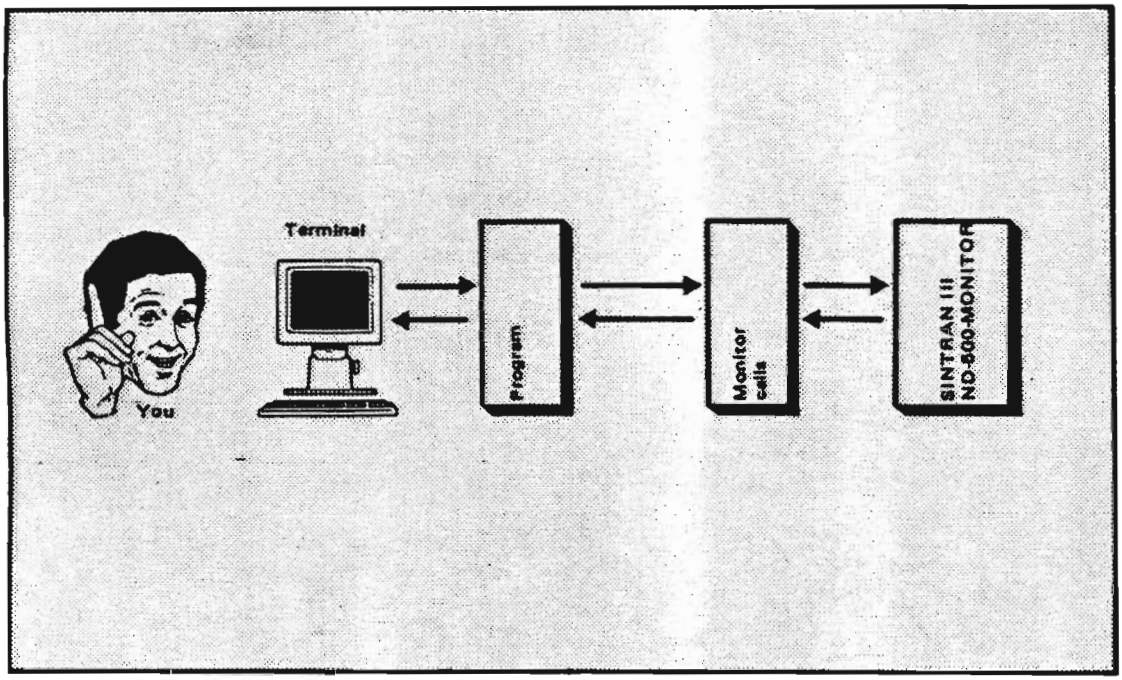


Figure 1. The use of monitor calls.

SINTRAN III provides monitor calls to programs running on the ND-100 computer or the ND-100 processor of an ND-500 computer. ND-500 MONITOR is a SINTRAN III subsystem on ND-500 computers. It provides monitor calls to programs running on the ND-500 processor. Almost every monitor call exists on both ND-100 and ND-500.

Some monitor calls are only available to user RT or user SYSTEM. Programs not started by these users will be rejected. Other monitor calls may be used only in RT programs or only in background programs.

You should use monitor calls when a programming language does not provide a particular function.

1.1 How to use this manual

Chapter 3 contains a detailed description of all monitor calls. They are sorted alphabetically. Each monitor call is described by two standard pages. Figure 2 shows the first of these pages.

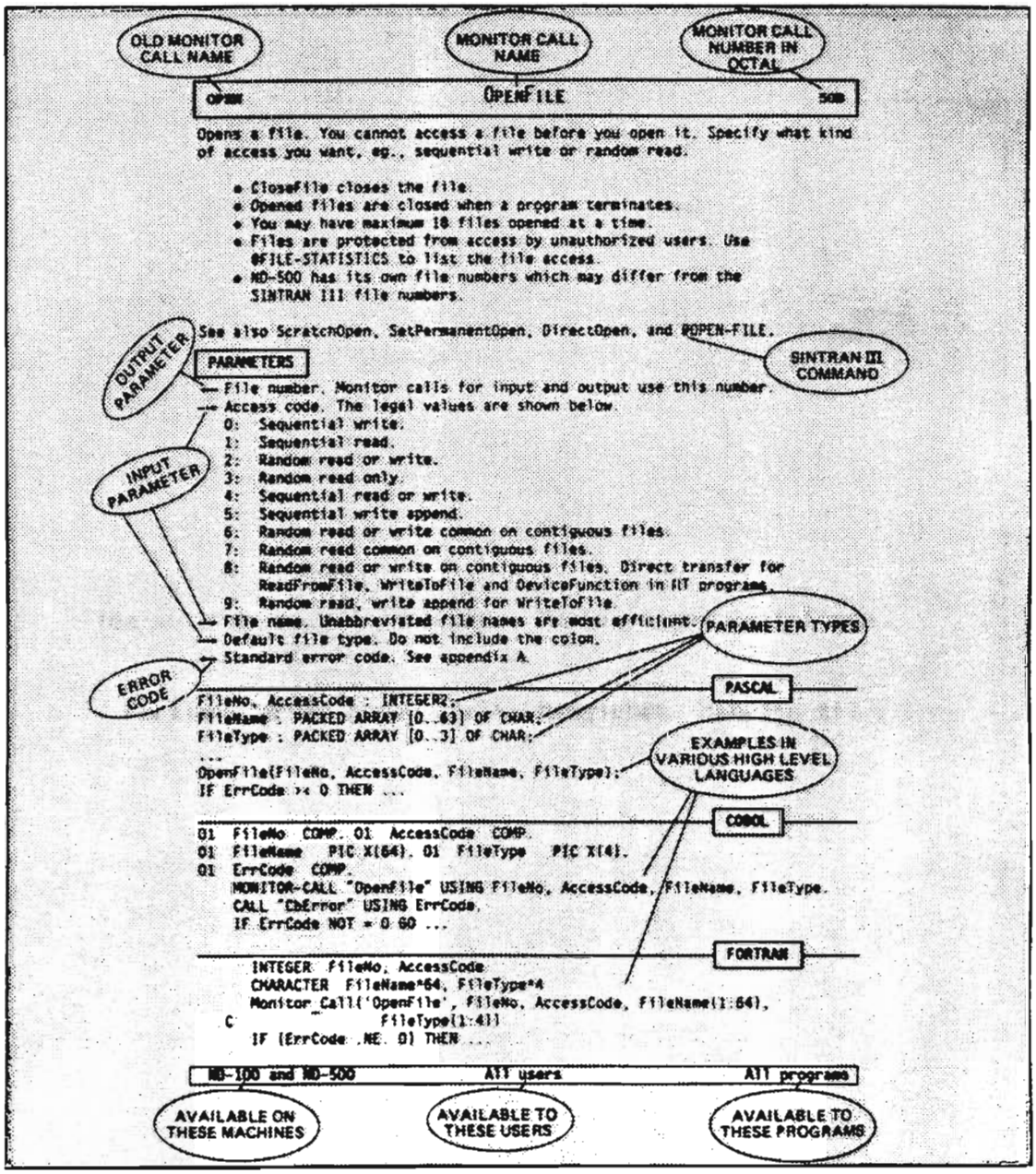


Figure 2. The first page of the monitor call description.

The second page describes monitor calls used from ND's special programming languages. See figure 3. Complicated monitor calls may use additional pages to describe more details.

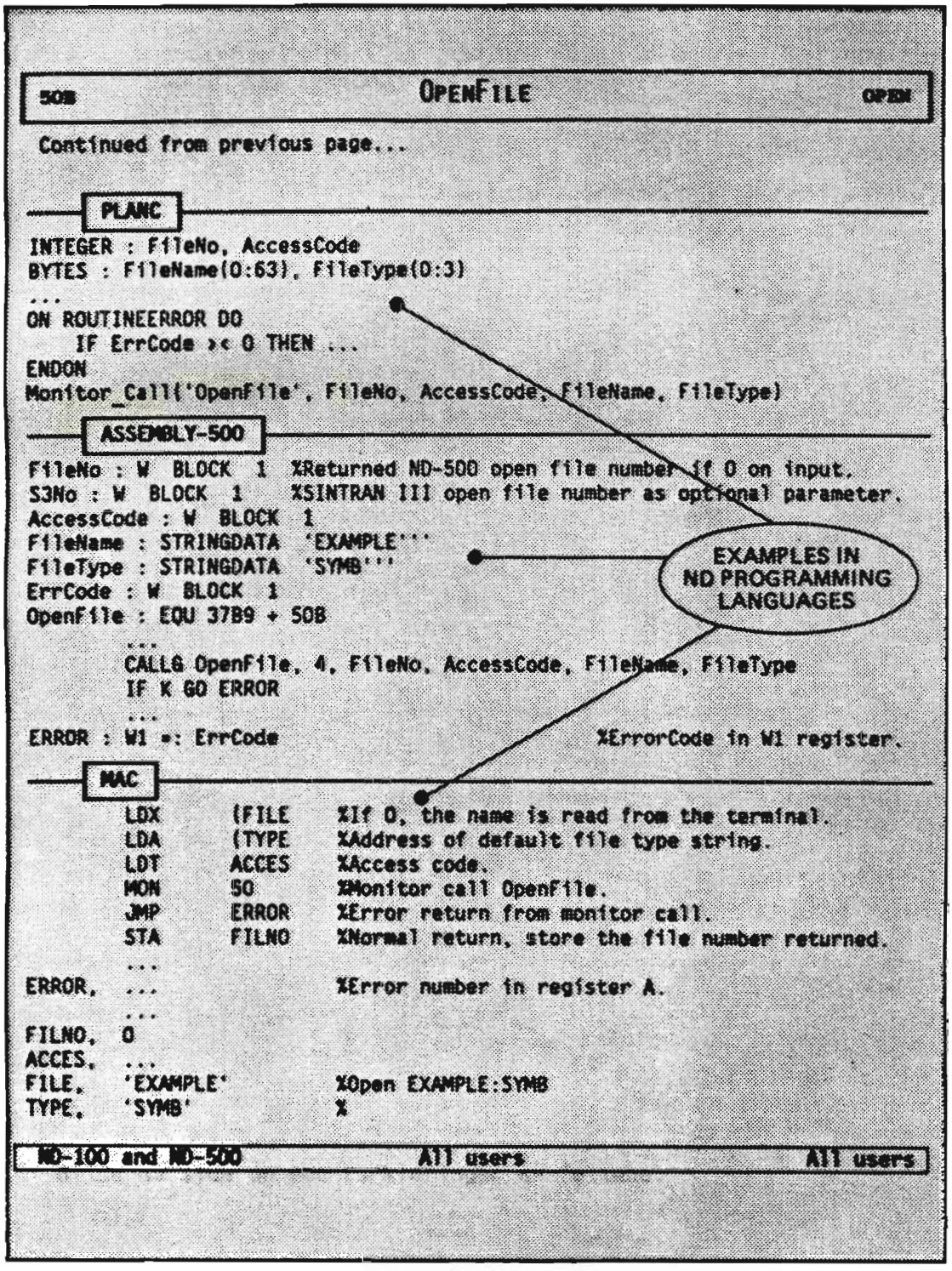


Figure 3. The second page of the monitor call description.

The following sections in this chapter show examples of how to compile and load programs. One example is provided for PASCAL, COBOL, FORTRAN, PLANC, MAC, and ASSEMBLY-500.

In chapter 2, monitor calls are grouped according to functions. Use it if you do not know the monitor call which provides a particular function. Lists sorted by monitor call numbers and short names are also provided.

The last appendix contains a glossary.

1.2 MONITOR CALLS IN PASCAL

The example below shows how to use monitor calls in a PASCAL program. The program checks the CPU number. Then it reads a name from a terminal and appends it to a file. The file is created if it does not exist.

```
PROGRAM StoreName (INPUT, OUTPUT);
VAR FileNo,Terminal,OutStat,I: INTEGER;
    InText: PACKED ARRAY [0..39] OF CHAR;     (* String parameter. *)
    Buffer: BitMap;       (* System defined type for monitor calls.*)
BEGIN
  Terminal := 1;          (* Logical device number of the terminal.*)
  GetSystemInfo(0,Buffer);
  IF Buffer[0] <> 678 THEN ExitFromProgram;    (* Test CPU number. *)
  OutString(1,'NAME: ',6,OutStat);    (* Output prompt on terminal.*)
  FOR I := 0 TO 39 DO InText[I] := ' ';         (*Spacefill InText.*)
  InString(Terminal,InText,40,141,OutStat);     (* Terminal input. *)
  OpenFile(FileNo,5,'NAME-STORAGE','DATA');       (* Append access.*)
  IF ErrCode = 46 THEN          (* ErrCode 46 means no such file. *)
  BEGIN
    CreateFile('NAME-STORAGE:DATA',0,0);
    OpenFile(FileNo,5,'NAME-STORAGE','DATA');
  END
  ELSE IF ErrCode <> 0 THEN ErrorMessage(ErrCode);
  FOR I := 0 TO 39 DO OutByte(FileNo,Ord(InText[I]));    (* Store. *)
  OutByte(FileNo,13);                          (* Out carriage return *)
  OutByte(FileNo,10);                               (* Out line feed *)
  CloseFile(FileNo);                                   (* Close file *)
  ExitFromProgram;                             (* Redundant statement.*)
END.
```

You should note the following with monitor calls in PASCAL:

- The function ErrCode returns the standard error code. You need not declare this function.
- The monitor calls sometimes use the types LONGINT, INTEGER2, BYTE, BYTE2 and BITMAP. BYTE2 is integers from 0 to 65535. BITMAP is defined as ARRAY [0..15] OF BYTE2. You need not define these types.
- Buffer parameters as in GetSystemInfo may be declared as records. This simplifies access and improves the readability.
- Some monitor call names are redefined in PASCAL to make the first 7 characters significant. This is commented in the examples.

This usage of monitor calls are available from version A of the CAT-PASCAL compilers for ND-100 and ND-500. Do not use the old PASCAL-100 and PASCAL-500 compilers.

Compile, load, and execute the program on an ND-100 computer as shown below. You need the library files CAT-2BANK:BRF and CAT-FREE:BRF. It provides the monitor call procedures. Your input is underlined.

```
@PASCAL
- ND-100 PASCAL COMPILER - VERSION A-
$OPTION B2
$COMPILE EX-PROG:SYMB, "EX-PROG:LIST", "EX-PROG:BRF"
*** NO ERRORS DETECTED ***
*EXIT
@BRF-LINKER
- BRF Linker -  10721A
Brl: PROGRAM-FILE "EX-PROG:PROG"
Brl: LOAD EX-PROG:BRF, CAT-2BANK:BRF, CAT-FREE:BRF
FREE: P 000726-177777
FREE: P 002456-177777
FREE: P 004215-177777
Brl: EXIT
@EX-PROG
```

The PASCAL REFERENCE MANUAL (ND-60.222) describes all compiler facilities. See the BRF-LINKER USER MANUAL (ND-60.196) for details of loading on the ND-100.

You compile, load, and execute a program on the ND-500 processor as shown below. See the manual ND-500 LOADER/MONITOR (ND-60.136) for details of loading.

```
@ND-500 PASCAL
- ND-500 PASCAL COMPILER  -  VERSION A -
*COMPILE EX-PROG:SYMB, "EX-PROG:LIST", "EX-PROG:NRF"
*** NO ERRORS DETECTED ***
*EXIT
@ND-500 LINKAGE-LOADER
- ND-500 LINKAGE LOADER,        VERSION F -
N11: SET-DOMAIN "EX-PROG"
N11: LOAD-SEGMENT EX-PROG:NRF, CAT-LIB:NRF
Program:.....400 P01     Data:..... 654 D01
CAT-LIB
Program:.....36422 P01     Data:.....16220 D01
N11: EXIT
@ND-500 EX-PROG
```

Use the RT-LOADER to load RT programs on the ND-100. See the manuals SINTRAN III REAL TIME GUIDE (ND-60.133) and SINTRAN III RT-LOADER (ND-60.051).

1.3 MONITOR CALLS IN FORTRAN

The example below shows how to use monitor calls in a FORTRAN program. The program reads a name from a terminal and appends it to a file. The file is created if it does not exist.

```
      PROGRAM StoreName
      INTEGER  FileNumber, NoOfBytes, Terminal, TextLength, I
      INTEGER*4 Zero              %General purpose parameter.
      CHARACTER InText*40;        %String to store terminal input.
      Terminal = 1; TextLength = 0; Zero = 0
      Monitor_Call('OutUpTo8Bytes',Terminal,'NAME: ')  %To terminal.
      DO FOR I = 1,40,8;      %Read 8 and 8 bytes from the terminal.
        Monitor_Call('InUpTo8Bytes',Terminal,InText(I:I+8),NoOfBytes)
        TextLength = TextLength + NoOfBytes;  %Count number of bytes.
        IF (NoOfBytes.NE.8) GO TO  100;  %Continue while more input.
      END DO
100   IF (TextLength.EQ.0) THEN Monitor_Call('ExitFromProgram') ENDIF
C     Open the mass storage file for append access, ie., type 5.
      Monitor_Call('OpenFile',FileNumber,5,'NAME-STORAGE','DATA')
      IF (ErrCode.EQ.46) THEN;    %ErrCode 46 means no such file.
        Monitor_Call('CreateFile','NAME-STORAGE:DATE',Zero,Zero)
        Monitor_Call('OpenFile',FileNumber,5,'NAME-STORAGE','DATA')
      ELSE IF (ErrCode.NE.0) THEN;    %Output error message.
        Monitor_Call('ErrorMessage',ErrCode)
      ENDIF
      DO FOR I = 1,40,8;        %Write 40 bytes to the file.
        Monitor_Call('OutUpTo8Bytes',Terminal,InText(I:I+8))
      END DO
      Monitor_Call('OutByte',13);              %Out carriage return.
      Monitor_Call('OutByte',12);              %Out line feed.
      Monitor_Call('CloseFile',FileNumber);  %Close file.
      END
```

With monitor calls in FORTRAN you should note the following:

- Error codes are automatically stored in the variable ErrCode. This integer variable ErrCode can be read as any other variable. It should not be declared. Use the compiler command STANDARD-CHECK OFF before compilation.
- Parameters declared as INTEGER*4 cannot be given as constants.
- You may use the monitor call numbers or the short names, eg., Monitor_Call('ERMSG',ErrCode) or Monitor_Call(64B,ErrCode).

This usage of monitor calls is available from FORTRAN-100 version E and FORTRAN-500 version I.

Compile, load, and execute the program on an ND-100 computer as shown below. You need the library file MON-CALL-1BANK:BRF. It provides the monitor call routines. Your input is underlined.

```
@FORTRAN-100
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053E
FTN: STANDARD-CHECK OFF
FTN: COMPILE EX-PROG:SYMB, EX-PROG:LIST, EX-PROG:BRF
- CPU TIME USED: 1.6 SECONDS.  26 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=3644 COMMON SIZE=0
FTN: EXIT
@BRF-LINKER
- BRF Linker -  10721A
Brl: PROGRAM-FILE EX-PROG:PROG
Brl: LOAD EX-PROG:BRF, MON-CALL-1BANK:BRF, FORTRAN-1BANK:BRF
FREE: P 004325-177777  etc.
Brl: EXIT
@EX-PROG
```

A large program may need more than 128 Kbyte memory space. In that case you should give the command SEPARATE-DATA ON before COMPILE. Then load the program with the libraries MON-CALL-2BANK:BRF and FORTRAN-2BANK:BRF instead of the ones shown. See the BRF-LINKER USER MANUAL (ND-60.196) for more details.

You compile, load, and execute a program on the ND-500 processor as shown below. The library MON-CALL-LIB:NRF provides the monitor call routines. See the manuals ND FORTRAN REFERENCE MANUAL (ND-60.145) and ND-500 LOADER/MONITOR (ND-60.136) for more details.

```
@ND-500 FORTRAN-500
- ND-500 ANSI 77 FORTRAN COMPILER - 203054I -
FTN: STANDARD-CHECK OFF
FTN: COMPILE EX-PROG:SYMB, "EX-PROG:LIST", "EX-PROG:NRF"
- CPU TIME USED: 1.1 SECONDS.  26 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=1642 COMMON SIZE=0
FTN: EXIT
@ND-500 LINKAGE-LOADER
- ND-500 LINKAGE LOADER        VERSION F -
N11: SET-DOMAIN "EX-PROG"
N11: LOAD-SEGMENT EX-PROG:NRF, MON-CALL-LIB, FORTRAN-LIB, EXCEPT-LIB
Program:.........1040 P01     Data:............ 502 D01  etc.
N11: EXIT
@ND-500 EX-PROG
```

Use the RT-LOADER to load RT programs on the ND-100. See the manuals SINTRAN III REAL TIME GUIDE (ND-60.133) and SINTRAN III RT-LOADER (ND-60.051).

1.4 MONITOR CALLS IN COBOL

The example below shows how to use monitor calls in a COBOL program. The program reads a name from a terminal and appends it to a file. The file is created if it does not exist.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXPROG.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 FileNo COMP. 01 NoOfBytes COMP. 01 Terminal COMP VALUE 1.
01 TextLength COMP VALUE 0. 01 OutStat COMP. 01 I COMP.
01 InText PIC X(100). 01 LastChar COMP.
01 FileName PIC X(22) VALUE "NAME-STORAGE'".
01 I4 COMP PIC 9(10) VALUE 0.
01 ErrCode COMP.
PROCEDURE DIVISION.
100.
    MONITOR-CALL 'OutString' USING Terminal 'NAME: ' 6 OutStat.
    MONITOR-CALL 'InString' USING 1 InText 40 141 OutStat.
    MONITOR-CALL 'OpenFile' USING FileNo 5 FileName "DATA'".
    CALL 'CbError' USING ErrCode.
    IF ErrCode = 46 GO 300.
    IF ErrCode > 0 MONITOR-CALL 'ErrorMessage' USING ErrCode.
200.
    MONITOR-CALL 'OutString' USING FileNo Intext 40 OutStat.
    MONITOR-CALL 'CloseFile' USING FileNo.
    MONITOR-CALL 'ExitFromProgram'.
300.
    MONITOR-CALL 'CreateFile' USING FileName I4 I4.
    MONITOR-CALL 'OpenFile' USING FileNo 5 FileName "DATA'".
    GO 200.
```

You should note the following with monitor calls in COBOL:

- The routine CBERROR returns the standard error code. The variable ErrCode must be declared.
- All strings input to monitor calls must be terminated with an apostrophe, eg., "DATA'".
- All strings output from monitor calls should be declared with length 100, eg., 01 InText PIC X(100).
- Parameters declared as PIC 9(10) cannot be given as constants.
- You may use monitor call numbers and short names, eg., MONITOR-CALL 'ERMSG' USING ErrCode or MONITOR-CALL 64B USING ErrCode.

This usage of monitor calls is available from version H of the COBOL-100 and COBOL-500 compilers.

Compile, load, and execute the program on an ND-100 computer as shown below. You need the library file MON-CALL-2BANK:BRF. It provides the monitor call routines. Your input is underlined.

```
@COBOL-100
ND-100 COBOL COMPILER - VERSION H
*COMPILE EX-PROG:SYMB, "EX-PROG:LIST", "EX-PROG:BRF"
--- END OF COMPILATION ---------------
NUMBER OF ERRORS FOUND:              0
NUMBER OF WARNINGS GIVEN:            0
NUMBER OF SOURCE LINES:             26
--------------------------------------
*EXIT
@BRF-LINKER
- BRF Linker -  10721A
Brl: PROGRAM-FILE "EX-PROG:PROG"
Brl: LOAD EX-PROG:BRF, MON-CALL-2BANK:BRF, COBOL-2BANK:BRF
FREE: P 000533-177777      D 000531-177777
FREE: P 003613-177777      D 003015-177777
FREE: P 004325-177777      D 003621-177777
Brl: EXIT
@EX-PROG
```

You compile, load, and execute a program on the ND-500 processor as shown below. The library MON-CALL-LIB:NRF provides the monitor call routines. See the ND COBOL REFERENCE MANUAL (ND-60.144) and the ND-500 LOADER/MONITOR (ND-60.136) for more details.

```
@ND-500 COBOL-500
- ND-500 COBOL COMPILER       VERSION F -
*COMPILE EX-PROG:SYMB, "EX-PROG:LIST", "EX-PROG:NRF"
--- END OF COMPILATION ---------------
NUMBER OF ERRORS FOUND:              0
NUMBER OF WARNINGS GIVEN:            0
NUMBER OF SOURCE LINES:             26
--------------------------------------
*EXIT
@ND-500 LINKAGE-LOADER
- ND-500 LINKAGE LOADER       VERSION F -
N11: SET-DOMAIN "EX-PROG"
N11: LOAD-SEGMENT EX-PROG:NRF, MON-CALL-LIB:NRF, COBOL-LIB:NRF
Program:.........400 P01     Data:............ 654 D01
MON-CALL-LIB
Program:........29706 P01     Data:...........12232 D01
COBOL-LIB-H00
Program:........36422 P01     Data:...........16220 D01
N11: EXIT
@ND-500 EX-PROG
```

Use the RT-LOADER to load RT programs on the ND-100. See the manuals SINTRAN III REAL TIME GUIDE (ND-60.133) and SINTRAN III RT-LOADER (ND-60.051).

1.5 MONITOR CALLS IN PLANC

The example below shows how to use monitor calls in a PLANC program. The program reads a name from a terminal and appends it to a file. The file is created if it does not exist.

```
MODULE StoreName
INTEGER : FileNumber,Char,Terminal:=1,TextLength:=0,Status,I
INTEGER4 : Zero                    %Doble integer, ie., 32 bit.
INTEGER ARRAY : Stack(0:100)
BYTES : InText(0:39)               %String to store terminal input.
PROGRAM : StoreName
  INISTACK Stack
  ON ROUTINEERROR DO    %Exception handler for monitor call errors.
    %ErrCode = 46 means "No such file". Other errors cause exit.
    IF ErrCode >< 46 THEN Monitor_Call('ErrorMessage',ErrCode) ENDIF
  ENDON
  Monitor_Call('OutString',Terminal,'NAME: ',6, Status)  %Prompt.
  FOR I IN 0:39 DO   %Read characters from the terminal.
    Monitor_Call('InByte',Terminal,Char)
  WHILE Char >< 215   %Continue until carriage return (with parity).
    Char =: InText(TextLength); TextLength + 1 =: TextLength
  ENDFOR
  IF TextLength = 0 THEN Monitor_Call('ExitFromProgram') ENDIF
  %Open the mass storage file for append access, ie., type 5.
  Monitor_Call('OpenFile',FileNumber,5,'NAME-STORAGE','DATA')
  IF ErrCode = 46 THEN    %ErrCode 46 means no such file.
    Monitor_Call('CreateFile','NAME-STORAGE:DATA',Zero,Zero)
    Monitor_Call('OpenFile',FileNumber,5,'NAME-STORAGE','DATA')
  ENDIF
  FOR I IN 0:TextLength-1 DO       %Write the input to the file.
    InText(I) =: Char;  Monitor_Call('OutByte',FileNumber,Char)
  ENDFOR
  Monitor_Call('OutByte',FileNumber,13)  %Out carriage return.
  Monitor_Call('OutByte',FileNumber,12)  %Out line feed.
  Monitor_Call('CloseFile',FileNumber)   %Close file.
ENDROUTINE
ENDMODULE
```

You should note the following with monitor calls in PLANC:

- Error codes different from 0 cause ERRETURN from monitor calls. That is, the current ON - ENDON statement is executed.
- Error codes are automatically stored in the variable ErrCode. This integer variable ErrCode can be read as any other variable. It should not be declared.
- Parameters of type INTEGER4 cannot be given as constants.
- You may use monitor call numbers or short names, eg., Monitor_Call('ERMSG',ErrCode) or Monitor_Call(64B,ErrCode).

This usage of monitor calls is available from version F of the PLANC-100 and PLANC-500 compilers.

Compile, load, and execute the program on an ND-100 computer as shown below. You need the library file MON-CALL-1BANK:BRF. It provides the monitor call routines. Your input is underlined.

```
@PLANC-100
- ND-100 PLANC COMPILER - VERSION F
*COMPILE EX-PROG:SYMB, "EX-PROG:LIST", "EX-PROG:BRF"
     32 LINES COMPILED.          0 DIAGNOSTICS.
*EXIT
@BRF-LINKER
- BRF Linker -  10721A
Brl: PROGRAM-FILE "EX-PROG:PROG"
Brl: LOAD EX-PROG:BRF, MON-CALL-1BANK:BRF, PLANC-1BANK:BRF
FREE: P 000726-177777
FREE: P 003613-177777
FREE: P 004325-177777
Brl: EXIT
@EX-PROG
```

A large program may need more than 128 Kbyte memory space. In that case you should give the command SEPARATE-DATA ON before COMPILE. Then load the program with the libraries MON-CALL-2BANK:BRF and PLANC-2BANK:BRF instead of the ones shown. See the BRF-LINKER USER MANUAL (ND-60.196) for more details.

You compile, load, and execute a program on the ND-500 processor as shown below. The library MON-CALL-LIB:NRF provides the monitor call routines. See the manuals PASCAL USER GUIDE (ND-60.124). ND-500 LOADER/MONITOR (ND-60.136) for more details.

```
@ND-500 PLANC-500
- ND-500 PLANC COMPILER        VERSION F -
*COMPILE EX-PROG:SYMB, "EX-PROG:LIST", "EX-PROG:NRF"
     32 LINES COMPILED.          0 DIAGNOSTICS.
*EXIT
@ND-500 LINKAGE-LOADER
- ND-500 LINKAGE LOADER        VERSION F -
Nll: SET-DOMAIN "EX-PROG"
Nll: LOAD-SEGMENT EX-PROG:NRF, MON-CALL-LIB:NRF, PLANC-LIB:NRF
Program:.........400 P01      Data:............ 654 D01
MON-CALL-LIB
Program:.........29706 P01    Data:...........12232 D01
PLANC-LIB-F00
Program:.........36422 P01    Data:...........16220 D01
Nll: EXIT
@ND-500 EX-PROG
```

Use the RT-LOADER to load RT programs on the ND-100. See the manuals SINTRAN III REAL TIME GUIDE (ND-60.133) and SINTRAN III RT-LOADER (ND-60.051).

1.6 MONITOR CALLS IN ASSEMBLY-500

The example below shows how to use monitor calls in ASSEMBLY-500. ASSEMBLY-500 is the assembly language for the ND-500. The program asks for an abbreviated file name. It displays the full file name. If no matching file is found, a new file is created.

```
  MODULE ExProg     %Module name.
  MAIN   Start      %Declaration of the main entry point.
  %Declaration of constants and parameters.
  OutMessage : EQU 37B9 + 32B  %Constant for monitor call number 32B.
  Prompt : STRINGDATA 'File name: '''  %Input parameter to OutMessage.
  InString : EQU 37B9 + 161B %Constant for monitor call number 161B.
  Terminal : W DATA 54         %Logical device number of a terminal.
  AbrevName : STRING 60        %Output parameter from InString.
  Max : W DATA 64              %Maximum string length.
  CR : W DATA 141              %Carriage return with parity, 128 + 13.
  LF : W DATA 10               %Line feed.
  Zero : W DATA 0              %Utility parameter.
  OutStat : W DATA 0           %Output parameter from InString.
  OutByte : EQU 37B9 + 2B      %Constant for monitor call number 2B.
  FullFileName : EQU 37B9 + 256B  %Monitor call number 256B.
  FileType : STRINGDATA 'SYMB'''  %Input parameter to FullFileName.
  FullName : STRING 100        %Output parameter from FullFileName.
  OutString : EQU 37B9 + 162B     %Monitor call number 162B.
  CreateFile : EQU 37B9 + 221B    %Monitor call number 221B.
  ErrorMessage : EQU 37B9 + 65B   %Monitor call number 65B.
  ExitFromProgram : EQU 37B9 + 0B %Monitor call number 0B.
  ErrCode : W BLOCK 1          %Declare ErrCode as a variable.
          %The program starts here.
  START:  CALLG OutMessage,1,Prompt  %Monitor call OutMessage.
          %The 1 means that OutMessage has 1 parameter.
          CALLG InString,4,Terminal,AbrevName,Max,CR   %Status in W1.
          %The 5 means that InString has 5 parameters.
          W1 =: OutStat   %Store the OutStatus
          CALLG OutByte,2,Terminal,CR  %Carriage return, ie., 13.
          CALLG OutByte,2,Terminal,LF  %Line feed, ie., 10.
          CALLG FullFileName,2,AbrevName,FullName
          IF K GO ERROR   %Monitor call errors set the K-register.
  CONTI:  CALLG OutString,2,Terminal,FullName    %Status in W1.
          CALLG ExitFromProgram,0  %Monitor call without parameters.
  ERROR:  W1 =: ErrCode          %Register W1 contains error codes.
          W1 COMP 46             %ErrCode 46 = NO SUCH FILE NAME.
          IF >< GO EXIT
          CALLG CreateFile,3,AbrevName,Zero,Zero   %Create the file.
          %Below is an alternative way of calling FullFileName.
          CALLG (37B9 + 256B),2,AbrevName,FullName
          GO CONTI
  EXIT:   CALLG ErrorMessage,1,ErrCode    %Output error message.
  ENDMODULE
```

The ND-500 ASSEMBLER REFERENCE MANUAL (ND-60.113) describes the assembly language completely.

You should note the following with monitor calls in ASSEMBLY-500:

- Error codes different from 0 set the K register. The error code is returned in the W1 register.
- The second parameter to CALLG is the number of following parameters.
- Only the monitor call name declared as a constant or the monitor call number are allowed in monitor calls.
- Monitor calls only available on the ND-100 cannot be executed.
- All string parameters must be terminated with an '.
- You may allocate parameters in a stack. This is done as for ordinary routine calls.

Assemble, load, and execute the program on an ND-500 computer as shown below. No library files are needed. Your input is underlined.

```
@ND-500 ASSEMBLY-500
- ND-500 ASSEMBLER 2.15, 13 JANUARY 1982. -
$ASSEMBLE EX-PROG:SYMB, EX-PROG:LIST, EX-PROG:NRF
NO ERRORS DETECTED
TIME USED IS 3 SECS.
$EXIT
@ND-500 LINKAGE-LOADER
- ND-Linkage-Loader -  F          10. September  1983 Time: 00:07
N11: SET-DOMAIN EX-PROG
N11: LOAD EX-PROG:NRF
Program:.....276 P     Data:.....364 D01
N11: EXIT
@ND-500 ex-prog
...
```

See the manual ND-500 LOADER/MONITOR (ND-60.136) for more details.

1.7 MONITOR CALLS IN MAC

The example below shows how to use monitor calls in a MAC program. MAC is the assembly language for the ND-100. The program copies a file to the terminal, one byte at a time. It waits one second between each character.

```
)9BEG  XCOPY
%Open a file for sequential read.
XCOPY, SAT  1            %Sequential read access in T register.
       LDX  (FILNA       %File name pointer in X register.
       LDA  (FILTY       %File type in A register.
       MON  50           %Monitor call OpenFile.
       MON  65           %Monitor call ErrorMessage.
       STA  FILNO        %Successfull OpenFile returns here.
%Read one byte from the file and output on the terminal.
LOOP,  LDT  FILNO
       MON  1            %Monitor call InByte from file.
       MON  65           %Monitor call ErrorMessage.
       SAT  1            %Logical device number of own terminal.
       MON  2            %Monitor call OutByte to terminal.
       JMP  ERR          %Error return, eg., end of file.
%Make the program wait one second between each character.
       LDA  (PAR         %Address of param list to SuspendProgram.
       MON  104          %Monitor call SuspendProgram.
       JMP  LOOP         %Read next character.
ERR,   LDT  FILNO        %Close the file and terminate the program.
       MON  43           %Monitor call CloseFile.
       MON  0            %Monitor call ExitFromProgram.
PAR,   NOTIM             %Address list of SuspendProgram parameters.
       TUNIT
NOTIM, 1                 %Number of time units to wait.
TUNIT, 2                 %Unit to wait is seconds.
FILNA, 'EXAMPLE-FILE'    %File name.
FILTY, 'DATA'            %File type.
FILNO, 0                 %File number.
)FILL
)9END
)LINE
```

You should note the following with monitor calls in MAC:

- Error codes different from 0 cause skip return. That is, the instruction following the monitor call is not executed.
- Only the monitor call number can be used in the monitor calls.
- Monitor calls only available on the ND-500 cannot be executed.
- All string parameters must be terminated with an '.

Assemble, load, and execute the program on an ND-100 computer as shown below. No library files are needed. Your input is underlined. See the MAC USER GUIDE (ND-60.096) and the BRF-LINKER USER MANUAL (ND-60.196) for more details.

```
@MAC
- MAC -
)9ASSM MAC-PROG,,MAC-PROG
 **** 000000 DIAGNOSTICS ****
)9EXIT
@BRF-LINKER
- BRF Linker -  10721A
Brl: PROG-FILE EX-PROG:PROG
Brl: LOAD MAC-PROG:BRF
FREE: P 000044-177777
Brl: EXIT
@EX-PROG
```

Use the RT-LOADER to load RT programs on the ND-100. See the manuals SINTRAN III REAL TIME GUIDE (ND-60.133) and SINTRAN III RT-LOADER (ND-60.051).

1.8 MONITOR CALLS IN OTHER PROGRAMMING LANGUAGES

The monitor call libraries are available in BASIC. BASIC programs use monitor calls in exactly the same way as FORTRAN.

Monitor calls may also be used from other languages. In such cases you have to write interface routines yourself. The interface routines are usually written in MAC or ASSEMBLY-500. Assemble the routines. Load them with the main program as any other routine library. The manual describing the language tells you how to transfer parameters.

You may also include MAC or ASSEMBLY-500 statements in PLANC programs.

2 OVERVIEW OF THE MONITOR CALLS

This chapter is a guide to the various monitor calls. It contains lists sorted by name, short name, and number. The monitor calls are also grouped according to function.

2.1 ALPHABETIC LIST OF MONITOR CALLS WITH PARAMETERS

This section contains an overview of monitor calls in the monitor call package. Experienced programmers may find this list useful to look up parameter sequences.

The parameter type differs from language to language. The type ARRAY means an integer array. The type STRING is a text. Marks in the 3 rightmost columns tell if the monitor call returns an error code, and if it is available on the ND-100 and the ND-500.

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
AccessRTCommon (RWRTC 406B)				•
1. Function	INTEGER			
2. RT common address	INTEGER			
3. Number of bytes	INTEGER			
4. Buffer	ARRAY			
AdjustClock (CLADJ 112B)			•	•
1. Number of time units	INTEGER			
2. Basic units	INTEGER			
AltPageTable (ALTON 33B)			•	
1. Page table number	INTEGER			
AppendSpooling (APSPF 240B)		•	•	•
1. File name	STRING			
2. Spooling file name	STRING			
3. Number of copyes	INTEGER			
4. User text	STRING			
AssignCamacLam (ASSIG 154B)		•	•	•
1. Logical device number	INTEGER			
2. Graded LAM number	INTEGER			
3. Crate number	INTEGER			
AttachSegment (REENT 167B)			•	
1. Segment number	INTEGER			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
AwaitFileTransfer (WAITF 121B)			•	•
1. File number	INTEGER			
2. Return flag	INTEGER			
3. Status	INTEGER			
AwaitRequest (WRQI 163B)		•	•	
1. Channel	INTEGER			
AwaitTransfer (MWAITF 431B)				•
1. File number	INTEGER			
2. Return flag	INTEGER			
3. Number of bytes read	INTEGER			
BackupClose (BCLOS 252B)		•	•	•
1. File number	INTEGER			
2. Flag	INTEGER			
BatchModeEcho (MBECH 325B)			•	•
1. Control bitmask	INTEGER			
BCNAF1Camac (BCNAF1 415B)				•
1. Function	INTEGER			
2. Address	INTEGER			
3. Data	INTEGER			
4. Status	INTEGER			
BCNAFCamac (BCNAF 414B)				•
1. Function	INTEGER			
2. Address	INTEGER			
3. Data	INTEGER			
4. Status	INTEGER			
CallCommand (COMND 70B)			•	•
1. Command	STRING			
CamacFunction (CAMAC 147B)			•	•
1. Data read/write	INTEGER			
2. Status returned	INTEGER			
3. Crate number	INTEGER			
4. Station number	INTEGER			
5. Subaddress	INTEGER			
6. Function	INTEGER			
CamacGLRegister (GL 150B)			•	•
1. Value input/output	INTEGER			
2. Crate number	INTEGER			
CamacIOInstruction (IOXN 153B)			•	•
1. Data input/output	INTEGER			
2. IOX instruction	INTEGER			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
CheckMonCall (MOINF 312B)		•	•	•
1. Monitor call number	INTEGER			
2. Monitor call entry	INTEGER			
ClearCapability (CAPCLE 424B)		•		•
1. Logical segment number	INTEGER			
2. Type	INTEGER			
ClearInBuffer (CIBUF 13B)		•	•	•
1. Logical device number	INTEGER			
ClearOutbuffer (COBUF 14B)		•	•	•
1. Logical device number	INTEGER			
CloseFile (CLOSE 43B)		•	•	•
1. File number	INTEGER			
CloseSpoolingFile (SPCLO 40B)		•	•	•
1. File number	INTEGER			
2. Text	STRING			
3. Number of copies	INTEGER			
4. Print flag	INTEGER			
CopyCapability (CAPCOP 423B)		•		•
1. Source logical segment number	INTEGER			
2. Source type	INTEGER			
3. Destination logical segment no	INTEGER			
4. Destination type	INTEGER			
5. Access	INTEGER			
6. Returned logical segment number	INTEGER			
CreateFile (CRALF 221B)		•	•	•
1. File name	STRING			
2. Start address of file	INTEGER4			
3. Number of pages in file	INTEGER4			
DataTransfer (ABSTR 131B)			•	•
1. Logical device number	INTEGER			
2. Function	INTEGER			
3. Memory address	INTEGER4			
4. Block address	INTEGER			
5. Number of blocks	INTEGER			
6. Status return (dummy ND-500)	INTEGER			
DefaultRemoteSystem (SRUSI 314B)		•	•	•
1. Remote system name	STRING			
2. Remote user name	STRING			
3. Remote user password	STRING			
4. Remote project password	STRING			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
DelayStart (SET 101B)			•	•
1. RT program	INTEGER			
2. Number of time units	INTEGER			
3. Basic units	INTEGER			
DeleteFile (MDLFI 54B)		•	•	•
1. File name	STRING			
DeletePage (DELPG 272B)		•	•	•
1. File number	INTEGER			
2. First page	INTEGER4			
3. Last page	INTEGER4			
4. Number of pages returned	INTEGER4			
DeviceControl (IOSET 141B)			•	•
1. Logical device number	INTEGER			
2. Input output flag	INTEGER			
3. RT program	INTEGER			
4. Control flag	INTEGER			
5. Return status	INTEGER			
DeviceFunction (MAGTP 144B)		•	•	•
1. Function	INTEGER			
2. Buffer	ARRAY			
3. Logical device number	INTEGER			
4. Parameter 1	INTEGER			
5. Parameter 2	INTEGER			
DirectOpen (DOPEN 220B)		•	•	•
1. File number	INTEGER			
2. Access code	INTEGER			
3. File name	STRING			
4. Default file type	STRING			
DisableEscape (DESCF 71B)			•	•
1. Logical device number	INTEGER			
DisableLocal (DLOFU 277B)		•	•	
DisableRTStart (RTOFF 137B)			•	•
1. RT program	INTEGER			
DisAssemble (DISASS 401B)				•
1. Program pointer	INTEGER			
2. Returned string	STRING			
3. Max number of characters	INTEGER			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
DMAFunction (UDMA 333B) 1. Logical device number 2. Function code 3. Logical memory address of data 4. Function dependent input par. 5. Function dependent output par.	 INTEGER INTEGER ARRAY INTEGER4 INTEGER4	•	•	•
EnableEscape (EESCF 72B) 1. Logical device number	 INTEGER		•	•
EnableLocal (ELOFU 276B) 1. Local handling	 INTEGER	•	•	
EnableRTStart (RTON 136B) 1. RT program	 INTEGER		•	•
ErrorMessage (QERMS 65B) 1. Error number	 INTEGER		•	•
ErrorReturn (MACROE 400B)				•
ExactDelayStart (DSET 126B) 1. RT program 2. Basic time units	 INTEGER INTEGER		•	•
ExactInterval (DINTV 130B) 1. RT program 2. Basic time units	 INTEGER INTEGER		•	•
ExactStartup (DABST 127B) 1. RT program 2. Basic time units	 INTEGER INTEGER		•	•
ExecuteCommand (UECOM 317B) 1. Command	 STRING		•	•
ExecutionInfo (RSIO 143B) 1. Execution mode 2. Input device 3. Output device 4. User index	 INTEGER INTEGER INTEGER INTEGER		•	•
ExitFromProgram (LEAVE 0B)			•	•
ExitRTProgram (RTEXT 134B)			•	•

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
ExpandFile (EXPFI 231B)		•	•	•
1. File name	STRING			
2. Number of pages	INTEGER4			
FileAsSegment (FSCNT 412B)		•		•
1. File number	INTEGER			
2. Logical segment number	INTEGER			
3. Type	INTEGER			
4. Segment number	INTEGER			
FileNotAsSegment (FSDCNT 413B)				•
1. File number	INTEGER			
2. Segment number	INTEGER			
FixContiguous (FIXC 160B)			•	•
1. Segment number	INTEGER			
2. Page number	INTEGER			
3. Status (dummy ND-500)	INTEGER			
FixInMemory (FIXMEM 410B)				•
1. Type	INTEGER			
2. First address	INTEGER			
3. Length in bytes	INTEGER			
4. Physical ND-100 address	INTEGER			
FixIOArea (IOFIX 404B)				•
1. First address	INTEGER			
2. Size of area in bytes	INTEGER			
FixScattered (FIX 115B)			•	•
1. Segment number	INTEGER			
ForceRelease (PRLS 125B)			•	•
1. Logical device number	INTEGER			
2. Input output flag	INTEGER			
ForceReserve (PRSRV 124B)			•	•
1. Logical device number	INTEGER			
2. Input output flag	INTEGER			
3. RT program	INTEGER			
4. Status (dummy ND-500)	INTEGER			
FullFileName (DEABF 256B)		•	•	•
1. Abbreviated file name	STRING			
2. Full file name	STRING			
3. Default file type	STRING			
GetActiveSegment (GASGM 421B)				•
1. Buffer, 2048 bytes	ARRAY			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
GetAddressArea (GBSIZ 222B)			•	
1. Background segment size	INTEGER			
GetAllFileIndexes (GUIOI 217B)		•	•	•
1. File number	INTEGER			
2. Directory index	INTEGER			
3. User index	INTEGER			
4. Object index	INTEGER			
5. Remote flag	INTEGER			
6. Remote system	STRING			
GetBasicTime (TIME 11B)			•	•
1. Time	INTEGER4			
GetBytesInFile (RMAX 62B)		•	•	•
1. File number	INTEGER			
2. Number of bytes	INTEGER4			
GetCurrentTime (CLOCK 113B)			•	•
1. Time buffer, 7 elements	ARRAY			
GetDefaultDir (FDFDI 250B)		•	•	•
1. User name	STRING			
2. Directory index	INTEGER			
3. User index	INTEGER			
GetDeviceType (GDEVT 263B)		•	•	•
1. Logical device type	INTEGER			
2. Input output flag	INTEGER			
3. Device type	INTEGER			
4. Device attribute	INTEGER4			
GetDirEntry (GDIEN 244B)		•	•	•
1. Directory index	INTEGER			
2. Directory entry, 24*16 bit	ARRAY			
3. Flag (dummy ND-100)	INTEGER			
4. Remote flag	INTEGER			
5. Remote system	STRING			
GetDirNameIndex (FDINA 243B)		•	•	•
1. Directory name	STRING			
2. Directory index	INTEGER			
3. Name index	INTEGER			
GetDirUserIndexes (MUIDI 213B)		•	•	•
1. User name	STRING			
2. Directory index	INTEGER			
3. User index	INTEGER			
GetErrorDevice (GERDV 254B)			•	•
1. Error device	INTEGER			
2. RT program	INTEGER			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
GetErrorInfo (RERRP 207B)			•	
1. Buffer	ARRAY			
2. Status output	INTEGER			
GetErrorMessage (GETXM 334B)		•	•	•
1. Error code	INTEGER			
2. Buffer for text, 128 bytes	STRING			
GetEscLocalChars (MGDAE 230B)			•	•
1. Logical device number	INTEGER			
2. Disconnect character	INTEGER			
3. Escape character	INTEGER			
GetFileIndexes (FOBJN 274B)		•	•	•
1. File name	STRING			
2. File type (dummy ND-100)	STRING			
3. Directory index	INTEGER			
4. User index	INTEGER			
5. Object index	INTEGER			
6. Next object index	INTEGER			
GetFileName (MGFIL 273B)		•	•	•
1. Directory index	INTEGER			
2. User index	INTEGER			
3. Object index	INTEGER			
4. File name	STRING			
5. Remote flag	INTEGER			
6. Remote system	STRING			
GetInputFlag (RFLAG 402B)				•
1. Value	INTEGER			
GetLastByte (LASTC 26B)			•	•
1. Logical device number	INTEGER			
2. Last character typed	INTEGER			
GetNameEntry (GNAEN 245B)		•	•	•
1. Name index	INTEGER			
2. Name table entry, 14*16 bit	ARRAY			
GetND500Param (5PAGET 437B)				•
1. Buffer, 5*32 bit	ARRAY			
GetObjectEntry (DROBJ 215B)		•	•	•
1. Buffer, 32*16 bit	ARRAY			
2. Directory index	INTEGER			
3. User index	INTEGER			
4. Object index	INTEGER			
5. Remote flag	INTEGER			
6. Remote system	STRING			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
GetOwnprocessInfo (GPRNAM 427B) 1. Process name, 34 bytes 2. Process number	 STRING INTEGER			•
GetOwnRTAddress (GETRT 30B) 1. RT description address	 INTEGER		•	•
GetProcessNo (GPRNUM 426B) 1. Process name 2. Process number	 STRING INTEGER			•
GetRTAddress (GRTDA 151B) 1. RT program name 2. RT program	 STRING INTEGER		•	•
GetRTDescr (RTDSC 27B) 1. RT program 2. RT descriptor, 26*16 bit 3. Number connected device	 INTEGER ARRAY INTEGER		•	•
GetRTName (GRTNA 152B) 1. RT program 2. RT program name	 INTEGER STRING		•	•
GetScratchSegment (GSWSP 422B) 1. Size in bytes 2. Logical segment number 3. Returned logical segment number	 INTEGER INTEGER INTEGER	•		•
GetSegmentEntry (RSEGM 53B) 1. Segment number 2. Buffer, 5*16 bit	 INTEGER ARRAY		•	•
GetSegmentNo (GSGNO 322B) 1. Segment name 2. Segment number return	 STRING INTEGER	•	•	•
GetSpoolingEntry (RSPQE 55B) 1. Spooling device number 2. Buffer, 136*16 bit	 INTEGER ARRAY	•	•	•
GetStartByte (REABT 75B) 1. File number 2. Byte pointer	 INTEGER INTEGER4	•	•	•
GetSystemInfo (CPUST 262B) 1. Number 2. Buffer, 12*16 bit	 INTEGER ARRAY	•	•	•

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
GetTerminalMode (GTMOD 306B)		•	•	
1. Logical device number	INTEGER			
2. Terminal mode	INTEGER			
GetTerminalType (MGTTY 16B)		•	•	•
1. Logical device number	INTEGER			
2. Terminal type	INTEGER			
GetTimeUsed (TUSED 114B)			•	•
1. Time used	INTEGER4			
GetTrapReason (GERRCO 505B)				•
1. Error code	INTEGER			
GetUserEntry (RUSER 44B)		•	•	•
1. User name	STRING			
2. Buffer, 32*16 bit	ARRAY			
GetUserName (GUSNA 214B)		•	•	•
1. User name	STRING			
2. Directory index	INTEGER			
3. User index	INTEGER			
4. Remote flag	INTEGER			
5. Remote system	STRING			
GetUserParam (PAGET 57B)			•	•
1. Buffer, 5*16 bit (ND-100) 5*32 bit (ND-500)	ARRAY			
GetUserRegister (GRBLK 420B)				•
1. Buffer, 39*32 bit	ARRAY			
GraphicFunction (GRAPHI 155B)			•	•
1. X coordinate	INTEGER			
2. Y coordinate	INTEGER			
3. Code	INTEGER			
4. Logical device number	INTEGER			
5. Function	INTEGER			
6. Return value (dummy ND-500)	INTEGER			
In4x2Bytes (B4INW 63B)		•	•	•
1. Logical device number	INTEGER			
2. Number of bytes read	INTEGER			
3. Bytes	STRING			
In8AndFlag (T8INB 310B)			•	•
1. Logical device number	INTEGER			
2. Number of bytes read	INTEGER			
3. Buffer	STRING			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
In8Bytes (B8INB 23B)		•	•	•
1. Logical device number	INTEGER			
2. Number of bytes read	INTEGER			
3. Bytes read	STRING			
InBufferSpace (ISIZE 66B)		•	•	•
1. Logical device number	INTEGER			
2. Number of bytes, return	INTEGER			
InBufferState (IBRSIZ 313B)		•	•	•
1. Logical device number	INTEGER			
2. Number in buffer	INTEGER			
3. Number until break	INTEGER			
InByte (INBT 1B)		•	•	•
1. Logical device number	INTEGER			
2. Return value	INTEGER			
InputString (DVINST 503B)				•
1. Device number	INTEGER			
2. Max number of bytes	INTEGER			
3. Number of bytes returned	INTEGER			
4. Buffer	ARRAY			
5. Break strategy	INTEGER			
6. Echo strategy	INTEGER			
7. Break table 1	INTEGER			
8. Break table 2	INTEGER			
9. Break table 3	INTEGER			
10. Break table 4	INTEGER			
11. Echo table 1	INTEGER			
12. Echo table 2	INTEGER			
13. Echo table 3	INTEGER			
14. Echo table 4	INTEGER			
InString (INSTR 161B)			•	•
1. Logical device number	INTEGER			
2. Text	STRING			
3. Number of bytes	INTEGER			
4. Terminator	INTEGER			
5. Status return	INTEGER			
InUpTo8Bytes (M8INB 21B)		•	•	•
1. Logical device number	INTEGER			
2. Number of bytes read	INTEGER			
3. Bytes read	STRING			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
IOInstruction (EXIOX 31B)			•	•
1. Register contents	INTEGER			
2. Device register address	INTEGER			
3. Register contents after	INTEGER			
LAMUFunction (MLAMU 315B)		•	•	
1. Function	INTEGER			
2. LAMU id	INTEGER			
3. Program	INTEGER			
4. Size	INTEGER			
5. Logical address	INTEGER			
6. Physical address	INTEGER			
LogInStart (MLOGI 326B)		•	•	
1. Terminal number	INTEGER			
2. User name	STRING			
3. Password	STRING			
4. Project password	STRING			
5. Subsystem	STRING			
6. User parameters	ARRAY			
MaxPagesInMemory (MXPISG 417B)				•
1. Logical segment number	INTEGER			
2. Segment type	INTEGER			
3. Number of pages	INTEGER			
MemoryAllocation (FIXC5 61B)		•	•	
1. Parameter1	INTEGER			
2. Parameter2	INTEGER			
3. Parameter3	INTEGER			
4. Parameter4	INTEGER			
5. Parameter5	INTEGER			
6. Parameter6	INTEGER			
MemoryUnFix (UNFIXM 411B)				•
1. Address	INTEGER			
ND500TimeOut (5TMOUT 514B)				•
1. Number of time units	INTEGER			
2. Time unit	INTEGER			
3. Status return	INTEGER			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
NewFileVersion (CRALN 253B) 1. File name 2. Page number first page 3. Number of pages	 STRING INTEGER4 INTEGER4	•	•	•
NewUser (SUSCN 241B) 1. User name 2. User password 3. Project name 4. Status return (dummy ND-500)	 STRING INTEGER STRING INTEGER	•	•	•
NoInterruptStart (DSCNT 107B) 1. RT program	 INTEGER		•	•
NormalPageTable (ALTOFF 34B)			•	
NoWaitSwitch (NOWT 36B) 1. Logical device number 2. Input output flag 3. Wait flag	 INTEGER INTEGER INTEGER	•	•	•
OffEscLocalFunction (ELOFF 303B)		•	•	
OldUser (RUSCN 242B)		•	•	•
OnEscLocalFunction (ELON 302B)		•	•	
OpenFile (OPEN 50B) 1. File number 2. Access code 3. File name 4. Default file type	 INTEGER INTEGER STRING STRING	•	•	•
OpenFileInfo (FOPFN 257B) 1. File name 2. File type 3. File number 4. Access code 5. Peripheral device number	 STRING STRING INTEGER INTEGER INTEGER	•	•	•
Out8Bytes (B8OUT 24B) 1. Logical device number 2. Bytes write	 INTEGER STRING	•	•	•
OutBufferSpace (OSIZE 67B) 1. Logical device number 2. Number of bytes, return	 INTEGER INTEGER	•	•	•

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
OutByte (OUTBT 2B)		•	•	•
1. Logical device number	INTEGER			
2. Output value	INTEGER			
OutMessage (MSG 32B)			•	•
1. Message	STRING			
OutNumber (IOUT 35B)			•	•
1. Format	INTEGER			
2. Number to be printed	INTEGER			
OutputString (DVOUTS 504B)				•
1. Device number	INTEGER			
2. Number of bytes	INTEGER			
3. Buffer	ARRAY			
OutString (OUTST 162B)			•	•
1. Logical device number	INTEGER			
2. Text	STRING			
3. Number of bytes	INTEGER			
4. Status return	INTEGER			
OutUpTo8Bytes (M8OUT 22B)		•	•	•
1. Logical device number	INTEGER			
2. Bytes write	STRING			
PrivInstruction (IPRIV 146B)			•	
1. Instruction	INTEGER			
ReadBlock (RPAGE 7B)		•	•	
1. File number	INTEGER			
2. Block number	INTEGER			
3. Data destination	ARRAY			
ReadDiskPage (RDPAG 270B)		•	•	•
1. Directory index	INTEGER			
2. Buffer	ARRAY			
3. Page	INTEGER4			
4. Number of pages	INTEGER			
ReadFromFile (RFILE 117B)		•	•	•
1. File number	INTEGER			
2. Return flag	INTEGER			
3. Buffer	ARRAY			
4. Block number	INTEGER			
5. Number of bytes to read	INTEGER4			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
ReadObjectEntry (ROBJE 41B) 1. File number 2. Buffer, 32*16 bit	 INTEGER ARRAY	•	•	•
ReadScratchFile (RDISK 5B) 1. Block number 2. Data destination	 INTEGER ARRAY	•	•	
ReentrantSegment (SREEN 212B) 1. Segment number	 INTEGER		•	
ReleaseDir (RLDIR 247B) 1. Directory index	 INTEGER	•	•	•
ReleaseResource (RELES 123B) 1. Logical device number 2. Input output flag	 INTEGER INTEGER		•	•
RenameFile (MRNFI 232B) 1. Old file name 2. New file name	 STRING STRING	•	•	•
ReservationInfo (WHDEV 140B) 1. Logical device number 2. Input output flag 3. Return value	 INTEGER INTEGER INTEGER		•	•
ReserveDir (REDIR 246B) 1. Directory index	 INTEGER	•	•	•
ReserveResource (RESRV 122B) 1. Logical device number 2. Input output flag 3. Return flag 4. Status	 INTEGER INTEGER INTEGER INTEGER		•	•
SaveND500Segment (WSEGN 416B) 1. Logical segment number 2. First logical page 3. Last logical page	 INTEGER INTEGER INTEGER	•		•
SaveSegment (WSEG 164B) 1. Segment number	 INTEGER		•	•
ScratchOpen (SCROP 235B) 1. File number 2. Access code 3. File name 4. Default file type	 INTEGER INTEGER STRING STRING	•	•	•

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
SegmentOverlay (SPLRE 323B)			•	
1. Segment number	INTEGER			
2. First page in area 1	INTEGER			
3. Number of pages area 1	INTEGER			
4. First page in area 2	INTEGER			
5. Number of pages area 2	INTEGER			
6. Clear flag	INTEGER			
SegmentToPageTable (ENTSG 157B)			•	•
1. Segment number	INTEGER			
2. Page table	INTEGER			
3. Interrupt level	INTEGER			
4. Start address	INTEGER			
SetBlockSize (SETBS 76B)		•	•	•
1. File number	INTEGER			
2. Block size in bytes	INTEGER4			
SetBreak (BRKM 4B)			•	•
1. Logical device number	INTEGER			
2. Break strategy	INTEGER			
3. Table, 8*16 bit/4*32 bit	ARRAY			
4. Number of characters	INTEGER			
SetClock (UPDAT 111B)			•	•
1. Minute	INTEGER			
2. Hour	INTEGER			
3. Day	INTEGER			
4. Month	INTEGER			
5. Year	INTEGER			
SetCommandBuffer (SETCM 12B)			•	•
1. Command string	STRING			
SetEcho (ECHOM 3B)			•	•
1. Logical device number	INTEGER			
2. Strategy	INTEGER			
3. Table, 8*16 bit/4*32 bit	ARRAY			
SetEscapeHandling (EUSEL 300B)		•	•	
1. Escape handling	INTEGER			
SetEscLocalChars (MSDAE 227B)			•	•
1. Logical device number	INTEGER			
2. Disconnect character	INTEGER			
3. Escape character	INTEGER			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
SetFileAccess (SFACC 237B)		•	•	•
1. File name	STRING			
2. Public access	STRING			
3. Friend access	STRING			
4. Own access	STRING			
SetMaxBytes (SMAX 73B)		•	•	•
1. File number	INTEGER			
2. Maximum bytes in the file	INTEGER4			
SetND500Param (5PASET 436B)				•
1. Buffer, 5*32 bit	ARRAY			
SetObjectEntry (DWOBJ 216B)		•	•	•
1. Buffer, 32*16 bit	ARRAY			
2. Directory index	INTEGER			
3. User index	INTEGER			
4. Object index	INTEGER			
5. Remote flag	INTEGER			
6. Remote system	STRING			
SetOutputFlags (WFLAG 403B)				•
1. Value	INTEGER			
SetPeripheralFile (SPEFI 234B)		•	•	•
1. File name	STRING			
2. Device number	INTEGER			
SetPermanentOpen (SPERD 236B)		•	•	•
1. File number	INTEGER			
SetProcessName (SPRNAM 425B)				•
1. Process name	STRING			
SetProcessPriority (SPRIO 507B)				•
1. New priority	INTEGER			
SetRemoteAccess (SRLMO 316B)			•	•
1. Mode	INTEGER			
SetRTPriorty (PRIOR 110B)			•	•
1. RT program	INTEGER			
2. Priority	INTEGER			
3. Old priority returned	INTEGER			
SetStartBlock (SETBL 77B)		•	•	•
1. File number	INTEGER			
2. Block number	INTEGER			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
SetStartByte (SETBT 74B) 1. File number 2. Byte pointer	 INTEGER INTEGER4	•	•	•
SetTemporaryFile (STEFI 233B) 1. File name	 STRING	•	•	•
SetTerminalType (MSTTY 17B) 1. Logical device number 2. Terminal type	 INTEGER INTEGER	•	•	•
SetUserParam (PASET 56B) 1. Buffer, 5*16 bit (ND-100) 5*32 bit (ND-500)	 ARRAY		•	•
StartOnInterrupt (CONCT 106B) 1. RT program 2. Logical device number	 INTEGER INTEGER		•	•
StartProcess (STARTP 500B) 1. Process number	 INTEGER	•		•
StartRTProgram (RT 100B) 1. RT program	 INTEGER		•	•
StartupInterval (INTV 103B) 1. RT program 2. Number of time units 3. Basic units	 INTEGER INTEGER INTEGER		•	•
StartupTime (ABSET 102B) 1. RT program 2. Seconds 3. Minutes 4. Hours	 INTEGER INTEGER INTEGER INTEGER		•	•
SuspendProgram (HOLD 104B) 1. Number of time units 2. Basic units	 INTEGER INTEGER		•	•
SwitchProcess (SWITCHP 502B) 1. Process number	 INTEGER			•

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
SwitchUserBreak (USTBRK 405B)				•
1. Function	INTEGER			
2. Address	INTEGER			
TerminalLineInfo (TREPP 332B)		•	•	•
1. Function code	INTEGER			
2. Logical device number	INTEGER			
3. Terminal status return	INTEGER			
TerminalMode (TERMO 52B)		•	•	•
1. Logical device number	INTEGER			
2. Mode	INTEGER			
TerminalNoWait (TNOWAI 307B)			•	•
1. Logical device number	INTEGER			
2. Input output flag	INTEGER			
3. No wait flag	INTEGER			
4. Status return (dummy ND-500)	INTEGER			
TerminalStatus (TERST 330B)		•	•	•
1. Logical device number	INTEGER			
2. Buffer, 22*16 bit	ARRAY			
TerminationHandling (EDTRM 206B)			•	
1. Enable disable flag	INTEGER			
2. Flag	INTEGER			
TimeOut (TMOUT 267B)		•	•	•
1. Number of time units	INTEGER			
2. Time unit	INTEGER			
3. Status return	INTEGER			
ToErrorDevice (ERMON 142B)			•	•
1. Error number	INTEGER			
2. Suberror number	INTEGER			
TransferData (EXABS 335B)			•	
1. Logical device number	INTEGER			
2. Function	INTEGER			
3. Memory address	INTEGER4			
4. Block address	INTEGER4			
5. Number of blocks	INTEGER4			
6. Status return	INTEGER			
TranslateAddress (ADR100 430B)				•
1. ND-500 array	INTEGER			
2. ND-100 physical word address	INTEGER			

NAME, SHORT NAME, NUMBER, AND PARAMETERS	TYPE	ERR	100	500
UnFixSegment (UNFIX 116B) 1. Segment number	INTEGER		•	•
WaitForRestart (RTWT 135B)			•	•
WarningMessage (ERMSG 64B) 1. Error number	INTEGER		•	•
WriteBlock (WPAGE 10B) 1. File number 2. Block number 3. Data	INTEGER INTEGER ARRAY	•	•	
WriteDirEntry (WDIEN 311B) 1. Directory index 2. Directory entry	INTEGER ARRAY	•	•	•
WriteDiskPage (WDPAG 271B) 1. Directory index 2. Buffer 3. Page 4. Number of pages	INTEGER ARRAY INTEGER4 INTEGER	•	•	•
WriteScratchFile (WDISK 6B) 1. Block number 2. Data	INTEGER ARRAY	•	•	
WriteToFile (WFILE 120B) 1. File number 2. Return flag 3. Buffer 4. Block number 5. Number of bytes to write	INTEGER INTEGER ARRAY INTEGER INTEGER4	•	•	•

2.2 COMMONLY-USED MONITOR CALLS

This section lists some commonly-used monitor calls.

CloseFile	43	Close file after access
CreateFile	221	Create or allocate a file
DeleteFile	54	Delete a file
ErrorMessage	65	Output error message and stop
ExecuteCommand	317	Execute a SINTRAN III command
ExitFromProgram	0	Terminate program
GetCurrentTime	113	Get the current time
OpenFile	50	Open a file
OpenFileInfo	257	Get file number
SuspendProgram	104	Suspend execution for a given time
WarningMessage	64	Output error message

2.3 FILE OPERATIONS

This section lists some common monitor calls for file operations. The advanced file system operations are not included.

AppendSpooling	240	Print a file
AwaitFileTransfer	121	Check data transfer
AwaitTransfer	431	Check data transfer
BackupClose	252	Close file for the BACKUP-SYSTEM
CloseFile	43	Close file after access
CreateFile	221	Create or allocate a file
DeleteFile	54	Delete a file
DeletePage	272	Delete pages of a file
DirectOpen	220	Open file with special access rights
ExpandFile	231	Increase the file size
FullFileName	256	Get file name in full
GetBytesInFile	62	Get the number of bytes in a file
GetStartByte	75	Get the next byte to access in a file
GetUserName	214	Get current user name
In8AndFlag	310	Read 8 bytes with break checking
In8Bytes	23	Read 8 bytes
InByte	1	Read one byte from a device or a file
InString	161	Read a string from a device
NewFileVersion	253	Create new file version
NewUser	241	Switch user name
OldUser	242	Reset old user name
OpenFile	50	Open a file
Out8Bytes	24	Write 8 bytes
OutByte	2	Write one byte to a device or a file
OutMessage	32	Write a message to a terminal
OutNumber	35	Write a number to a terminal
OutputString	504	Write a string to a file
OutString	162	Write to a peripheral file
OutUpTo8Bytes	22	Write up to 8 characters
ReadBlock	7	Read random blocks from a file
ReadDiskPage	270	Read a disk page
ReadFromFile	117	Read randomly from a file
ReadScratchFile	5	Read randomly from the scratch file
ReleaseResource	123	Release a device or a file
RenameFile	232	Rename a file
ReserveResource	122	Reserve a file or device
ScratchOpen	235	Open a file as a scratch file
SetBlockSize	76	Set block size of files
SetFileAccess	237	Set file access
SetMaxBytes	73	Set the size of an opened file
SetPermanentOpen	236	Set a file permanently open
SetStartBlock	77	Set start block in a file
SetStartByte	74	Set start byte of a file
SetTemporaryFile	233	Set file as temporary
WriteBlock	10	Write random blocks to a file
WriteDiskPage	271	Write to a disk page
WriteScratchFile	6	Write randomly to the scratch file
WriteToFile	120	Write randomly to a file

2.4 INPUT AND OUTPUT MONITOR CALLS

This section lists the monitor calls for input and output to files, printers, and terminals.

AppendSpooling	240	Print a file
AwaitFileTransfer	121	Check data transfer
AwaitTransfer	431	Check data transfer
BatchModeEcho	325	Set batch and mode job echo
ClearInBuffer	13	Clear a device's input buffer
ClearOutBuffer	14	Clear a device's output buffer
CloseFile	43	Close file after access
CloseSpoolingFile	40	Close and print a file
CopyPage	251	Copy page for the BACKUP-SYSTEM
DataTransfer	131	Data to and from mass storage
DisableEscape	71	Disable the ESCAPE key
EnableEscape	72	Enable the ESCAPE key
ErrorMessage	65	Output error message and stop
FixIOArea	404	Fix an area for input and output
GetLastByte	26	Get last byte typed on a terminal
GetSpoolingEntry	55	Get spooling queue information
GetStartByte	75	Get the next byte to access in a file
GetTerminalMode	306	Get terminal mode
GraphicFunction	155	Execute graphic function
In4x2Bytes	63	Read 4 words from a device
In8AndFlag	310	Read 8 bytes with break checking
In8Bytes	23	Read 8 bytes
InBufferSpace	66	Get number of bytes in input buffer
InBufferState	313	Get input buffer information
InByte	1	Read one byte from a device or a file
InString	161	Read a string from a device
InUpTo8Bytes	21	Read up to 8 bytes from a device
NoWaitSwitch	36	Switch No Wait on and off
Out8Bytes	24	Write 8 bytes
OutBufferSpace	67	Get number of bytes in output buffer
OutByte	2	Write one byte to a device or a file
OutMessage	32	Write a message to a terminal
OutNumber	35	Write a number to a terminal
OutputString	504	Write a string to a file
OutString	162	Write to a peripheral file
OutUpTo8Bytes	22	Write up to 8 characters
ReadADChannel	37	Read analog to digital channels
ReadBlock	7	Read random blocks from a file
ReadFromFile	117	Read randomly from a file
ReadScratchFile	5	Read randomly from the scratch file
ScratchOpen	235	Open a file as a scratch file
SetBlockSize	76	Set block size of files
SetBreak	4	Set break characters
SetCommandBuffer	12	Set command input buffer
SetEcho	3	Set echo for a terminal
SetEscLocalChars	227	Set escape and local characters
SetStartBlock	77	Set start block in a file
SetStartByte	74	Set start byte of a file

TerminalNoWait	307	Switch No Wait on and off
ToErrorDevice	142	Write to the error device
TransferData	335	Data to and from mass storage
WarningMessage	64	Output error message
WriteBlock	10	Write random blocks to a file
WriteScratchFile	6	Write randomly to the scratch file
WriteToFile	120	Write randomly to a file

2.5 MONITOR CALLS FOR TERMINAL HANDLING

This section lists the monitor calls which handle terminals. See also the monitor calls for input and output.

BatchModeEcho	325	Set batch and mode job echo
DisableEscape	71	Disable the ESCAPE key
DisableLocal	277	Disable the local function
EnableEscape	72	Enable the ESCAPE key
EnableLocal	276	Enable the local function
ExecutionInfo	143	Get execution information
GetErrorDevice	254	Find error device
GetLastByte	26	Get last byte typed on a terminal
GetOwnRTAddress	30	Get own RT description address
GetTerminalMode	306	Get terminal mode
GetTerminalType	16	Get the terminal type
In8AndFlag	310	Read 8 bytes with break checking
In8Bytes	23	Read 8 bytes
InUpTo8Bytes	21	Read up to 8 bytes from a device
Out8Bytes	24	Write 8 bytes
SetBreak	4	Set break characters
SetCommandBuffer	12	Set command input buffer
SetEcho	3	Set echo for a terminal
SetEscapeHandling	300	User defined escape handling
SetEscLocalChars	227	Set escape and local characters
SetTerminalName	275	Set file name for terminals
SetTerminalType	17	Set terminal type
StopEscapeHandling	301	Stop user defined escape handling
SwitchUserBreak	405	Control escape handling
TerminalLineInfo	332	Get terminal line information
TerminalMode	52	Set terminal mode
TerminalStatus	330	Get terminal status

2.6 MONITOR CALLS FOR PRINTER HANDLING

This section lists the monitor calls which handle printers.

AppendSpooling	240	Print a file
CloseSpoolingFile	40	Close and print a file
GetSpoolingEntry	55	Get spooling queue information
SetPeripheralName	234	Set a file as peripheral

2.7 MONITOR CALLS FOR ERROR HANDLING

This section lists the monitor calls which relate to error handling.

ErrorMessage	65	Output error message and stop
ErrorReturn	400	Error return from program
GetErrorDevice	254	Find error device
GetErrorInfo	207	Get error information
GetErrorMessage	334	Get error message text
GetND500Param	437	Get information about termination
GetTrapReason	505	Read swapper error
GetUserParam	57	Get information about termination
SetND500Param	436	Set information about termination
SetUserParam	56	Set information about termination
TerminationHandling	206	Set termination handling
ToErrorDevice	142	Write to the error device
WarningMessage	64	Output error message

2.8 FILE SYSTEM OPERATIONS

This section lists the monitor calls related to the file system structure. Section 2.3 describes simple file operations.

FileAsSegment	412	Connect file as segment
FileNotAsSegment	413	Disconnect file as segment
FileSystemFunction	327	Various file system functions
FullFileName	256	Get file name in full
GetAllFileIndexes	217	Get directory, user, and object index
GetDefaultDir	250	Get the default directory
GetDirEntry	244	Get directory entry
GetDirNameIndex	243	Get directory and name indexes
GetDirUserIndexes	213	Get user and directory indexes
GetFileIndexes	274	Get the file indexes
GetFileName	273	Get file name
GetObjectEntry	215	Get file information
GetSpoolingEntry	55	Get spooling queue information
GetUserEntry	44	Get information about a user
GetUserName	214	Get current user name
NewUser	241	Switch user name
OffEscLocalFunction	303	Disable escape and local characters
OldUser	242	Reset old user name
OnEscLocalFunction	302	Enable escape and local characters
OpenFileInfo	257	Get file number
ReadBlock	7	Read random blocks from a file
ReadFromFile	117	Read randomly from a file
ReadObjectEntry	41	Get information about opened files
ReleaseDir	247	Release a directory
ReserveDir	246	Reserve directory
SetMaxBytes	73	Set the size of an opened file
SetObjectEntry	216	Change file information
SetPeripheralName	234	Set a file as peripheral
SetRemoteAccess	316	Set remote file access
WriteBlock	10	Write random blocks to a file
WriteDirEntry	311	Change a directory entry
WriteToFile	120	Write randomly to a file

2.9 RT PROGRAM EXECUTION

This section lists the monitor calls which relate to RT program execution.

AccessRTCommon	406	Access RT common from ND-500
ChangeSegment	337	Change segment and page table
DelayStart	101	Start RT program after specified time
DisableRTStart	137	Disable start of RT programs
EnableRTStart	136	Enable start of RT programs
ExactDelayStart	126	Start RT program after given period
ExactInterval	130	Periodic execution of RT program
ExactStartup	127	Start RT program at a specific time
ExecutionInfo	143	Get execution information
ExitRTProgram	134	Exit from RT program
ForceRelease	125	Release another program's device
ForceReserve	124	Force reserve a device
GetBasicTime	11	Get the internal time
GetOwnRTAddress	30	Get own RT description address
GetProcessNo	426	Get process number
GetRTAddress	151	Get RT description address
GetRTDescr	27	Get RT description
GetRTName	152	Get the name of an RT program
NoInterruptStart	107	Disconnect program from interrupt
PrivInstruction	146	Execute privileged instruction
SegmentToPageTable	157	Enter a routine as a direct task
SetProcessPriority	507	Set ND-500 process priority
SetRTPriority	110	Set the priority of an RT program
StartOnInterrupt	106	Connect a program to an interrupt
StartRTProgram	100	Start RT program
StartupInterval	103	Periodic execution of RT program
StartupTime	102	Start RT program at a specified time
StopRTProgram	105	Abort an RT program
SuspendProgram	104	Suspend execution for a given time
WaitForRestart	135	Set RT program in wait state

2.10 DEVICE HANDLING

This section lists the monitor calls which relate to device handling.

AssignCamacLam	154	Assign Camac Lam
AwaitFileTransfer	121	Check data transfer
AwaitTransfer	431	Check data transfer
BatchModeEcho	325	Set batch and mode job echo
BCNAF1Camac	415	CAMAC function on ND-500
BCNAFCamac	414	CAMAC function on the ND-500
CamacFunction	147	Operate CAMAC
CamacGlRegister	150	Read the CAMAC GL register
CamacIOInstruction	153	Execute an IOX instruction
ClearInBuffer	13	Clear a device's input buffer
ClearOutBuffer	14	Clear a device's output buffer
CopyPage	251	Copy page for the BACKUP-SYSTEM
DataTransfer	131	Data to and from mass storage
DeviceControl	141	Set control information for a device
DeviceFunction	144	Various device functions
DisableEscape	71	Disable the ESCAPE key
DMAFunction	333	Various DMA functions
EnableEscape	72	Enable the ESCAPE key
ForceRelease	125	Release another program's device
ForceReserve	124	Force reserve a device
GetDeviceType	263	Get information about a device
GetDirNameIndex	243	Get directory and name indexes
GetErrorDevice	254	Find error device
GetLastByte	26	Get last byte typed on a terminal
GetNameEntry	245	Get disk information
GraphicFunction	155	Execute graphic function
In4x2Bytes	63	Read 4 words from a device
InBufferSpace	66	Get number of bytes in input buffer
InBufferState	313	Get input buffer information
InByte	1	Read one byte from a device or a file
InString	161	Read a string from a device
InUpTo8Bytes	21	Read up to 8 bytes from a device
IOInstruction	31	Execute an IOX machine instruction
NoInterruptStart	107	Disconnect program from interrupt
NoWaitSwitch	36	Switch No Wait on and off
OutBufferSpace	67	Get number of bytes in output buffer
OutByte	2	Write one byte to a device or a file
OutMessage	32	Write a message to a terminal
OutNumber	35	Write a number to a terminal
OutputString	504	Write a string to a file
OutString	162	Write to a peripheral file
OutUpTo8Bytes	22	Write up to 8 characters
ReadADChannel	37	Read analog to digital channels
ReadDiskPage	270	Read a disk page
ReleaseDir	247	Release a directory
ReleaseResource	123	Release a device or a file
ReservationInfo	140	Check device reservation
ReserveDir	246	Reserve directory
ReserveResource	122	Reserve a file or device

SetBreak	4	Set break characters
SetClock	111	Set new time and date
SetPeripheralName	234	Set a file as peripheral
StartOnInterrupt	106	Connect a program to an interrupt
TerminalNoWait	307	Switch No Wait on and off
TransferData	335	Data to and from mass storage
WriteDiskPage	271	Write to a disk page

2.11 SEGMENT ADMINISTRATION

This section lists the monitor calls which relate to segment administration. You should know the difference between segments on the ND-100 and the ND-500.

AltPageTable	33	Switch page table
AttachSegment	167	Attach a reentrant segment
ChangeSegment	337	Change segment and page table
ClearCapability	424	Clear capability
CopyCapability	423	Copy a capability
ExitFromSegment	133	Exchange current segments
FileAsSegment	412	Connect file as segment
FileNotAsSegment	413	Disconnect file as segment
FixContiguous	160	Fix a segment in memory
FixInMemory	410	Fix a shared segment
FixScattered	115	Fix a segment in memory
GetActiveSegment	421	Get names of active segments
GetAddressArea	222	Get address area size
GetScratchSegment	422	Get scratch segment
GetSegmentEntry	53	Get segment information
GetSegmentNo	322	Get segment number
JumpToSegment	132	Jump to another ND-100 segment
LAMUFunction	315	Operate the LAMU system
MaxPagesInMemory	417	Set maximum pages in memory
MemoryAllocation	61	Fix or unfix a segment
MemoryUnFix	411	Release fixed ND-500 segment
ReentrantSegment	212	Connect reentrant segment
SaveND500Segment	416	Save a segment in a domain
SaveSegment	164	Save a segment
SegmentOverlay	323	Create segment overlay systems
SegmentToPageTable	157	Enter a routine as a direct task
UnFixSegment	116	Release a fixed segment

2.12 DATA COMMUNICATION

This section lists the monitor calls which relate to data communication and networks.

DefaultRemoteSystem	314	Set default remote system
DisableLocal	277	Disable the local function
EnableLocal	276	Enable the local function
GetEscLocalChars	230	Get escape and local characters
HDLCFunction	201	Operate link to remote computer
OffEscLocalFunction	303	Disable escape and local characters
OnEscLocalFunction	302	Enable escape and local characters
SetEscapeHandling	300	User defined escape handling
SetRemoteAccess	316	Set remote file access
StopEscapeHandling	301	Stop user defined escape handling
XMSGFunction	200	Program to program communication

2.13 MONITOR CALLS FOR INTERNAL USE

This section lists the monitor calls intended for ND subsystems only. You ought not to use these monitor calls. However, it is not illegal. See the SINTRAN III SOURCE LISTING PART 1 and 2 for documentation. The source listing is sold as a product.

AnswerSIBAS	506	Handle request to SIBAS on ND-500
DMACBreakpoint	51	Assembly breakpoints
GetSIBASMessage	305	GetSIBASMessage
SendSIBASMessage	304	Send request to SIBAS
SIBASFunction	432	Various SIBAS functions
UEAdministrator	321	Handle User Environment
UELogin	320	User Environment login

2.14 MISCELLANEOUS MONITOR CALLS

This section lists the monitor calls which are not listed in any of the previous function groups.

AdjustClock	112	Adjust the computer's clock
CallCommand	70	Execute a SINTRAN III command
CheckMonCall	312	Check monitor call implementation
DefineBreakpoint	45	Define debug breakpoint
DisAssemble	401	Disassemble ND-500 programs
ForceTrap	435	Force a trap in any process
GetBreakpointInfo	46	Get debug breakpoint
GetInputFlags	402	Read process communication flags
GetInRegisters	165	Read process input registers
GetOwnProcessInfo	427	Get process name and number
GetSystemInfo	262	Get various system information
GetTimeUsed	114	Get CPU time used
GetUserRegisters	420	Get register block
LogInStart	326	Log in a user and start a subsystem
ND500Function	60	Control the ND-500 from the ND-100
ND500TimeOut	514	Suspend program execution
NormalPageTable	34	Reset alternative page table
OctobusFunction	324	Various Octobus functions
PIOCFunction	255	Various PIOC functions
SetBreakpoint	47	Start program to be debugged
SetOutputFlags	403	Set process communication flags
SetOutRegisters	166	Write to process input registers
SetProcessName	425	Define a name for a process
StartProcess	500	Start an ND-500 process
StopProcess	501	Stop ND-500 process
SwitchProcess	502	Switch ND-500 process
TimeOut	267	Suspend program execution
TranslateAddress	430	Translate ND-100 to ND-500 addresses
UserDef0	170	User defined monitor call

2.15 MONITOR CALLS SORTED ON SHORT NAMES

Here are the monitor calls sorted according to their short names. You will also find the short names in the index at the end of this manual.

500MT	ND500MagTape
500RF	ND500ReadFile
500WF	ND500WriteFile
5PAGET	GetND500Param
5PASET	SetND500Param
5TMOUT	ND500TimeOut
ABORT	StopRTProgram
ABSET	StartupTime
ABSTR	DataTransfer
ADR100	TranslateAddress
AIRDW	ReadADChannel
ALTOF	NormalPageTable
ALTON	AltPageTable
APSPF	AppendSpooling
ASSIG	AssignCamacLam
B4INW	In4x2Bytes
B8INB	In8Bytes
B8OUT	Out8Bytes
BCLOS	BackupClose
BCNAF	BCNAFCamac
BCNAF1	BCNAF1Camac
BRKM	SetBreak
CAMAC	CamacFunction
CAPCLE	ClearCapability
CAPCOP	CopyCapability
CIBUF	ClearInBuffer
CLADJ	AdjustClock
CLOCK	GetCurrentTime
CLOSE	CloseFile
COBUF	ClearOutBuffer
COMND	CallCommand
CONCT	StartOnInterrupt
COPAG	CopyPage
CPUST	GetSystemInfo
CRALF	CreateFile
CRALN	NewFileVersion
DABST	ExactStartup
DBRK	DefineBreakpoint
DEABF	FullFileName
DELPG	DeletePage
DESCF	DisableEscape
DINTV	ExactInterval
DISASS	DisAssemble
DIW	GetInRegisters
DLOFU	DisableLocal
DMAC	DMACBreakpoint
DOLW	SetOutRegisters
DOPEN	DirectOpen
DROBJ	GetObjectEntry
DSCNT	NoInterruptStart
DSET	ExactDelayStart
DUSEL	StopEscapeHandling
DVINST	InputString
DVOUTS	OutputString
DWOBJ	SetObjectEntry
ECHOM	SetEcho
EDTRM	TerminationHandling
EESCF	EnableEscape
ELOFF	OffEscLocalFunction
ELOFU	EnableLocal
ELON	OnEscLocalFunction
ENTSG	SegmentToPageTable
ERMON	ToErrorDevice
ERMSG	WarningMessage
EUSEL	SetEscapeHandling
EXABS	TransferData
EXIOX	IOInstruction
EXPFI	ExpandFile
FDFDI	GetDefaultDir
FDINA	GetDirNameIndex
FIX	FixScattered
FIXC	FixContiguous
FIXC5	MemoryAllocation
FIXMEM	FixInMemory
FOBJN	GetFileIndexes
FOPFN	OpenFileInfo
FSCNT	FileAsSegment
FSDCNT	FileNotAsSegment
FSMTY	FileSystemFunction
GASGM	GetActiveSegment
GBRK	GetBreakpointInfo
GBSIZ	GetAddressArea
GDEVT	GetDeviceType
GDIEN	GetDirEntry
GERDV	GetErrorDevice
GERRCOD	GetTrapReason
GETRT	GetOwnRTAddress
GETXM	GetErrorMessage
GL	CamacGlRegister
GNAEN	GetNameEntry
GPRNAME	GetOwnProcessInfo
GPRNUM	GetProcessNo

GRAPHIC	GraphicFunction
GRBLK	GetUserRegisters
GRTDA	GetRTAddress
GRTNA	GetRTName
GSGNO	GetSegmentNo
GSWSP	GetScratchSegment
GTMOD	GetTerminalMode
GUIOI	GetAllFileIndexes
GUSNA	GetUserName
HOLD	SuspendProgram
IBRSIZ	InBufferState
INBT	InByte
INSTR	InString
INTV	StartupInterval
IOFIX	FixIOArea
IOMTY	TerminalFunction
IOSET	DeviceControl
IOUT	OutNumber
IOXN	CamacIOInstruction
IPRIV	PrivInstruction
ISIZE	InBufferSpace
LASTC	GetLastByte
LEAVE	ExitFromProgram
M8INB	InUpTo8Bytes
M8OUT	OutUpTo8Bytes
MACROE	ErrorReturn
MAGTP	DeviceFunction
MAPSIB	SendSIBASMessage
MBECH	BatchModeEcho
MCALL	JumpToSegment
MDLFI	DeleteFile
MEXIT	ExitFromSegment
MGDAE	GetEscLocalChars
MGFIL	GetFileName
MGTTY	GetTerminalType
MHDLC	HDLCFunction
MLAMU	LAMUFunction
MLOGI	LogInStart
MOINF	CheckMonCall
MRNFI	RenameFile
MSDAE	SetEscLocalChars
MSG	OutMessage
MSIBB	GetSIBASMessage
MSTTY	SetTerminalType
MUIDI	GetDirUserIndexes
MWAITF	AwaitTransfer
MXPISG	MaxPagesInMemory
N500M	ND500Function
NOWT	NoWaitSwitch
OCTO	OctobusFunction
OPEN	OpenFile
OSIZE	OutBufferSpace
OUTBT	OutByte
OUTST	OutString
PAGET	GetUserParam
PASET	SetUserParam
PIOCM	PIOCFunction
PRIOR	SetRTPriority
PRLS	ForceRelease
PRSRV	ForceReserve
PRT	ForceTrap
QERMS	ErrorMessage
RDISK	ReadScratchFile
RDPAG	ReadDiskPage
REABT	GetStartByte
REDIR	ReserveDir
REENT	AttachSegment
RELES	ReleaseResource
RERRP	GetErrorInfo
RESRV	ReserveResource
RFILE	ReadFromFile
RFLAG	GetInputFlags
RLDIR	ReleaseDir
RMAX	GetBytesInFile
ROBJE	ReadObjectEntry
RPAGE	ReadBlock
RSEGM	GetSegmentEntry
RSIO	ExecutionInfo
RSPQE	GetSpoolingEntry
RT	StartRTProgram
RTDSC	GetRTDescr
RTEXT	ExitRTProgram
RTOFF	DisableRTStart
RTON	EnableRTStart
RTWT	WaitForRestart
RUSCN	OldUser
RUSER	GetUserEntry
RWRTC	AccessRTCommon
SBRK	SetBreakpoint
SCROP	ScratchOpen
SET	DelayStart
SETBL	SetStartBlock
SETBS	SetBlockSize
SETBT	SetStartByte
SETCM	SetCommandBuffer
SFACC	SetFileAccess
SIBFU	SIBASFunction
SIBSURV	AnswerSIBAS
SMAX	SetMaxBytes
SPCHG	ChangeSegment
SPCLO	CloseSpoolingFile
SPEFI	SetPeripheralName
SPERD	SetPermanentOpen
SPLRE	SegmentOverlay

SPRIO	SetProcessPriority
SPRNAME	SetProcessName
SREEN	ReentrantSegment
SRLMO	SetRemoteAccess
SRUSI	DefaultRemoteSystem
STEFI	SetTemporaryFile
STOPPR	StopProcess
STRATPR	StartProcess
STRFI	SetTerminalName
SUSCN	NewUser
SWAPMC	CallSwapper
SWITCHP	SwitchProcess
SYNCT	SystemControl
T8INB	In8AndFlag
TERMO	TerminalMode
TERST	TerminalStatus
TIME	GetBasicTime
TMOUT	TimeOut
TNOWAI	TerminalNoWait
TPSTRA	GetStopInfo
TREPP	TerminalLineInfo
TUSED	GetTimeUsed
UDMA	DMAFunction
UEADM	UEAdministrator
UECOM	ExecuteCommand
UELOG	UELogin
UNFIX	UnFixSegment
UNFIXMEM	MemoryUnFix
UPDAT	SetClock
US0	UserDef0
US1	UserDef1
US2	UserDef2
US3	UserDef3
US4	UserDef4
US5	UserDef5
US6	UserDef6
US7	UserDef7
USCNT	UserControl
USTBRK	SwitchUserBreak
WAITF	AwaitFileTransfer
WDIEN	WriteDirEntry
WDISK	WriteScratchFile
WDPAG	WriteDiskPage
WFILE	WriteToFile
WFLAG	SetOutputFlags
WHDEV	ReservationInfo
WPAGE	WriteBlock
WRQI	AwaitRequest
WSEG	SaveSegment
WSEGN	SaveND500Segment
XMSG	XMSGFunction

2.16 MONITOR CALLS IN NUMERIC ORDER

Here are the monitor calls sorted by numbers. The numbers are octal.

0B	ExitFromProgram
1B	InByte
2B	OutByte
3B	SetEcho
4B	SetBreak
5B	ReadScratchFile
6B	WriteScratchFile
7B	ReadBlock
10B	WriteBlock
11B	GetBasicTime
12B	SetCommandBuffer
13B	ClearInBuffer
14B	ClearOutBuffer
16B	GetTerminalType
17B	SetTerminalType
21B	InUpTo8Bytes
22B	OutUpTo8Bytes
23B	In8Bytes
24B	Out8Bytes
26B	GetLastByte
27B	GetRTDescr
30B	GetOwnRTAddress
31B	IOInstruction
32B	OutMessage
33B	AltPageTable
34B	NormalPageTable
35B	OutNumber
36B	NoWaitSwitch
37B	ReadADChannel
40B	CloseSpoolingFile
41B	ReadObjectEntry
43B	CloseFile
44B	GetUserEntry
45B	DefineBreakpoint
46B	GetBreakpointInfo
47B	SetBreakpoint
50B	OpenFile
51B	DMACBreakpoint
52B	TerminalMode
53B	GetSegmentEntry
54B	DeleteFile
55B	GetSpoolingEntry
56B	SetUserParam
57B	GetUserParam
60B	ND500Function
61B	MemoryAllocation
62B	GetBytesInFile
63B	In4x2Bytes
64B	WarningMessage
65B	ErrorMessage
66B	InBufferSpace
67B	OutBufferSpace
70B	CallCommand
71B	DisableEscape
72B	EnableEscape
73B	SetMaxBytes
74B	SetStartByte
75B	GetStartByte
76B	SetBlockSize
77B	SetStartBlock
100B	StartRTProgram
101B	DelayStart
102B	StartupTime
103B	StartupInterval
104B	SuspendProgram
105B	StopRTProgram
106B	StartOnInterrupt
107B	NoInterruptStart
110B	SetRTPriority
111B	SetClock
112B	AdjustClock
113B	GetCurrentTime
114B	GetTimeUsed
115B	FixScattered
116B	UnFixSegment
117B	ReadFromFile
120B	WriteToFile
121B	AwaitFileTransfer
122B	ReserveResource
123B	ReleaseResource
124B	ForceReserve
125B	ForceRelease
126B	ExactDelayStart
127B	ExactStartup
130B	ExactInterval
131B	DataTransfer
132B	JumpToSegment
133B	ExitFromSegment
134B	ExitRTProgram
135B	WaitForRestart

136B	EnableRTStart
137B	DisableRTStart
140B	ReservationInfo
141B	DeviceControl
142B	ToErrorDevice
143B	ExecutionInfo
144B	DeviceFunction
146B	PrivInstruction
147B	CamacFunction
150B	CamacGlRegister
151B	GetRTAddress
152B	GetRTName
153B	CamacIOInstruction
154B	AssignCamacLam
155B	GraphicFunction
157B	SegmentToPageTable
160B	FixContiguous
161B	InString
162B	OutString
163B	AwaitRequest
164B	SaveSegment
165B	GetInRegisters
166B	SetOutRegisters
167B	AttachSegment
170B	UserDef0
171B	UserDef1
172B	UserDef2
173B	UserDef3
174B	UserDef4
175B	UserDef5
176B	UserDef6
177B	UserDef7
200B	XMSGFunction
201B	HDLCFunction
206B	TerminationHandling
207B	GetErrorInfo
212B	ReentrantSegment
213B	GetDirUserIndexes
214B	GetUserName
215B	GetObjectEntry
216B	SetObjectEntry
217B	GetAllFileIndexes
220B	DirectOpen
221B	CreateFile
222B	GetAddressArea
227B	SetEscLocalChars
230B	GetEscLocalChars
231B	ExpandFile
232B	RenameFile
233B	SetTemporaryFile
234B	SetPeripheralName
235B	ScratchOpen
236B	SetPermanentOpen
237B	SetFileAccess
240B	AppendSpooling
241B	NewUser
242B	OldUser
243B	GetDirNameIndex
244B	GetDirEntry
245B	GetNameEntry
246B	ReserveDir
247B	ReleaseDir
250B	GetDefaultDir
251B	CopyPage
252B	BackupClose
253B	NewFileVersion
254B	GetErrorDevice
255B	PIOCFunction
256B	FullFileName
257B	OpenFileInfo
260B	UserControl
261B	SystemControl
262B	GetSystemInfo
263B	GetDeviceType
264B	ND500ReadFile
265B	ND500WriteFile
266B	ND500MagTape
267B	TimeOut
270B	ReadDiskPage
271B	WriteDiskPage
272B	DeletePage
273B	GetFileName
274B	GetFileIndexes
275B	SetTerminalName
276B	EnableLocal
277B	DisableLocal
300B	SetEscapeHandling
301B	StopEscapeHandling
302B	OnEscLocalFunction
303B	OffEscLocalFunction
304B	SendSIBASMessage
305B	GetSIBASMessage
306B	GetTerminalMode
307B	TerminalNoWait
310B	In8AndFlag
311B	WriteDirEntry
312B	CheckMonCall
313B	InBufferState
314B	DefaultRemoteSystem
315B	LAMUFunction
316B	SetRemoteAccess
317B	ExecuteCommand
320B	UELogin
321B	UEAdministrator
322B	GetSegmentNo
323B	SegmentOverlay

324B	OctobusFunction
325B	BatchModeEcho
326B	LogInStart
327B	FileSystemFunction
330B	TerminalStatus
332B	TerminalLineInfo
333B	DMAFunction
334B	GetErrorMessage
335B	TransferData
336B	TerminalFunction
337B	ChangeSegment
400B	ErrorReturn
401B	DisAssemble
402B	GetInputFlags
403B	SetOutputFlags
404B	FixIOArea
405B	SwitchUserBreak
406B	AccessRTCommon
407B	GetStopInfo
410B	FixInMemory
411B	MemoryUnFix
412B	FileAsSegment
413B	FileNotAsSegment
414B	BCNAFCamac
415B	BCNAF1Camac
416B	SaveND500Segment
417B	MaxPagesInMemory
420B	GetUserRegisters
421B	GetActiveSegment
422B	GetScratchSegment
423B	CopyCapability
424B	ClearCapability
425B	SetProcessName
426B	GetProcessNo
427B	GetOwnProcessInfo
430B	TranslateAddress
431B	AwaitTransfer
432B	SIBASFunction
435B	ForceTrap
436B	SetND500Param
437B	GetND500Param
500B	StartProcess
501B	StopProcess
502B	SwitchProcess
503B	InputString
504B	OutputString
505B	GetTrapReason
506B	AnswerSIBAS
507B	SetProcessPriority
510B	CallSwapper
514B	ND500TimeOut

2.17 MONITOR CALL NUMBERS NO LONGER IN USE

The following monitor call numbers are no longer in use. However, the monitor calls exist to allow old programs to be used.

37B	201B	321B
47B	251B	324B
51B	255B	407B
60B	260B	432B
132B	261B	435B
133B	264B	506B
145B	265B	510B
156B	266B	
165B	304B	
200B	320B	

These monitor calls are not available in the monitor call package. The monitor call package returns -2 as error code when these numbers are used.

2.18 NEW MONITOR CALL NUMBERS

Each version of SINTRAN III introduces new monitor calls. Their names are not known by old compilers. The monitor call package allows you to use such monitor calls. You must specify the monitor call number.

3 MONITOR CALL REFERENCE

This chapter describes each monitor call in detail. The calls appear in alphabetical order. Use the index if you only know the short name or the monitor call number.

Look at page 2 for an explanation of how to use this reference.

RTC AccessRTCommon 40GB

Reads from or write to RT common from an ND-500 program. RT common is an area in physical memory where RT programs may exchange data.

- Some SINTRAN III systems are generated without RT common. The size of RT common is defined in the generation.

See also GetInputFlags and SetOutputFlags.

PARAMETERS

→ Function. Use 0 to read, and 1 to write.
→ RT common address. This is a logical ND-100 address in the logical address area of RT common. ND-100 has 16-bit word addresses.
→ Buffer of data to be read or written.
→ Number of bytes to read or write.

PASCAL

```
Func, RTCommon, NoOfBytes : LONGINT;
Buffer : ARRAY [0..6] OF BITMAP;
...
AccessRTCommon(Func, RTCommon, NoOfBytes, Buffer);
```

COBOL

```
01  Func  COMP.
01  RTCommon  COMP.
01  NoOfBytes  COMP.
01  Buffer.
    02  array COMP OCCURS 100 TIMES.
    ...
    MONITOR-CALL "AccessRTCommon" USING Func, RTCommon, NoOfBytes, Buffer.
```

FORTRAN

```
      INTEGER  Func, RTCommon, NoOfBytes
      INTEGER  Buffer(100)
      ...
      Monitor_Call('AccessRTCommon', Func, RTCommon, NoOfBytes, Buffer(1))
```

ND-500 | User RT and user SYSTEM | RT programs

406B **AccessRTCommon** **RWRTC**

Continued from previous page...

PLANC

```
INTEGER : Func, RTCommon, NoOfBytes
BYTE ARRAY : Buffer(0:199)
...
Monitor_Call('AccessRTCommon', Func, RTCommon, NoOfBytes, Buffer(0))
```

ASSEMBLY-500

```
Func : W  BLOCK  1
RTCommon : W  BLOCK  1
NoOfBytes : W  BLOCK  1
Buffer : W  ARRAY 100
ErrCode : W  BLOCK  1
AccessRTCommon : EQU 37B9 + 406B
        ...
        CALLG AccessRTCommon, 4, Func, RTCommon, NoOfBytes, Buffer
        IF K GO ERROR
        ...
Error : W1 =: ErrCode     %Error code in the W1 register.
```

MAC

Not available.

ND-500 **User RT and user SYSTEM** **RT programs**

CLADJ — AdjustClock — 112B

Sets the computer's clock forward or back. If the computer panel has a clock, it is adjusted.

- The startup time for RT programs can be set by StartupTime. This time is modified. The next start of periodic programs started by StartupTime is also affected. Other sheduling times are not affected.

See also SetClock, GetCurrentTime, and GetBasicTime.

PARAMETERS

→ Number of time units the clock will be adjusted by. Negative values make the clock halt for the specified time.
→ The type of time units. 1 = basic time units, ie., 1/50th of a second, 2 = seconds, 3 = minutes, 4 = hours.

PASCAL

```
TimeUnits, UnitType : INTEGER2;
...
AdjustClock(TimeUnits, UnitType);
```

COBOL

```
01  TimeUnits  COMP.
01  UnitType   COMP.
    ...
    MONITOR-CALL "AdjustClock" USING TimeUnits, UnitType.
```

FORTRAN

```
      INTEGER  TimeUnits, UnitType
      ...
      Monitor_Call('AdjustClock', TimeUnits, UnitType)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

112B **ADJUSTCLOCK** **CLADJ**

Continued from previous page...

PLANC

```
INTEGER : TimeUnits, UnitType
...
Monitor_Call('AdjustClock', TimeUnits, UnitType)
```

ASSEMBLY-500

```
TimeUnits : W  BLOCK  1
UnitType : W  BLOCK  1
ErrCode : W  BLOCK  1
AdjustClock : EQU 37B9 + 112B
        ...
        CALLG AdjustClock, 2, TimeUnits, UnitType
        IF K GO ERROR
        ...
Error : W1 =: ErrCode
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     112     %Monitor call AdjustClock.
        ...
PAR,    TIME            %Number of time units the clock is adjusted by.
        BASE            %Time units.
        ...
TIME,   ...
BASE,   ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

ALTOM — ALTPAGETABLE — 33B

Switches page table. Each page table allows you to access 128 Kbyte memory. SINTRAN III has 4 page tables. SINTRAN III VSX, version K, has 16 page tables. They are numbered 0-15. RT programs may use any page table. Background programs will get page table 2. The parameter is ignored.

- When used from background, the background segment size must be 256 Kbyte. See @CHANGE-BACKGROUND-SEGMENT-SIZE.
- Do not call AltPageTable twice without calling NormalPageTable in between.
- If addressing via the alternative page table (LDA, STA), bit 0 in the status register (STS) must be set to 1. If this bit is 0, addressing is via the normal page table.
- Monitor calls are independent of status bit 0. Return parameters are always on the alternative page table.
- The specified page table is used for all later X register relative, B register relative, and indirect addresses. P register relative addresses go through the normal page table. See the ND-100 REFERENCE MANUAL (ND-06.014) for details on addressing.

See also related NormalPageTable.

PARAMETERS

→ Number of the page table to use.

PASCAL

```
PageTableNumber : INTEGER2;
...
AltPageTable(PageTableNumber);
```

COBOL

```
01  PageTableNumber  COMP.
    ...
    MONITOR-CALL "AltPageTable" USING PageTableNumber.
```

FORTRAN

```
      INTEGER  PageTableNumber
      ...
      Monitor_Call('AltPageTable', PageTableNumber)
```

ND-100 | All users | All programs

33B	ALTPAGETABLE	ALTOM

Continued from previous page...

PLANC

```
INTEGER : PageTableNumber
...
Monitor_Call('AltPageTable', PageTableNumber)
```

ASSEMBLY-500

Not available.

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     33      %Monitor call AltPageTable.
        ...
PAR,    PAGE
        ...
PAGE,   ...             %Number of the alternative page table.
```

ND-100	All users	All users

SIBSURV — ANSWERSIBAS — 506B

Used by the SIBAS database system on the ND-500. The monitor call answers requests from application programs run on the ND-100.

PARAMETERS

No parameters.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-500 | User RT and user SYSTEM | RT programs

506B	ANSWERSIBAS	SIBSUBV

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Not available.

MAC

Not available.

ND-500	User RT and user SYSTEM	RT programs

APSPF APPENDSPOOLING 240B

Prints a file. The printer has a queue of files waiting to be output. The file is appended to this queue. One or more copies can be printed.

- You may connect a text, eg., INVOICE, to the job. The operator uses the text to select all files to be printed on special paper.
- SINTRAN III, version K, allows both the file and the printer to be on a remote system.

See also CloseSpoolingFile and @APPEND-SPOOLING-FILE.

PARAMETERS

→ Name of file to be printed. The name may be abbreviated.
→ Peripheral file name of the printer, eg., LINE-PRINTER or PHILIPS.
→ Number of copies to be printed.
→ Optional message connected to the printout. Maximum 128 characters.
← Standard error code. See appendix A.

PASCAL

```
FileName, PrinterName : PACKED ARRAY [0..63] OF CHAR;
NoOfCopies : INTEGER2;
UserText : PACKED ARRAY [0..79] OF CHAR;
...
AppendSpooling(FileName, PrinterName, NoOfCopies, UserText);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  NoOfCopies  COMP.
01  FileName   PIC X(64).
01  PrinterName   PIC X(64).
01  UserText   PIC X(100).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "AppendSpooling" USING FileName, PrinterName,
                 NoOfCopies, UserText.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  NoOfCopies
      CHARACTER  FileName*64, PrinterName*64, UserText*80
      ...
      Monitor_Call('AppendSpooling', FileName(1:64), PrinterName(1:64),
     C                  NoOfCopies, UserText(1:80))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

240B **APPENDSPOOLING** **APSPF**

Continued from previous page...

PLANC

```
INTEGER : NoOfCopies
BYTES : FileName(0:63), PrinterName(0:63), UserText(0:79)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('AppendSpooling', FileName, PrinterName, NoOfCopies, UserText)
```

ASSEMBLY-500

```
NoOfCopies : W  BLOCK  1
FileName : STRINGDATA  'EXAMPLE:SYMB'''
PrinterName : STRINGDATA  'LINE-PRINTER'''
UserText : STRINGDATA  'File queued'''
ErrCode : W  BLOCK  1
AppendSpooling : EQU 37B9 + 240B
        ...
        CALLG AppendSpooling, 4, FileName, PrinterName, NoOfCopies, UserText
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (FILE    %Memory address of file name.
        LDT     NOCOP    %Number of copies to be printed in bit 0:14.
        LDA     (TEXT    %Memory address of message to the error device.
        COPY    SA  DD
        LDA     (DEV     %Memory address of name of spooling device.
        MON     240      %Monitor call AppendSpooling.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILE,   'EXAMPLE:SYMB'   %Send EXAMPLE:SYMB to a printer.
DEV,    'LINE-PRINTER'   %Print file on the device LINE-PRINTER.
NOCOP,  ...              %Bit 15 set to 1 means print message.
TEXT,   'GUMMED LABELS'  %Message to be send to error device.
```

ND-100 and ND-500 **All users** **All users**

ASSIG — AssignCamacLam — 154B

Assigns a graded LAM in the Camac identification table to a logical device number. See NORD Process I/O Software Guide (ND-60.093).

- Remove the LAM by specifying logical device number -1.

See also CamacFunction, BCNAFCamac, BCNAF1Camac, CamacIOInstruction, and CamacGlRegister.

PARAMETERS

→ Logical device number. See appendix B.
→ Graded LAM number. Use 0 for high priority on interrupt level 13.
→ Camac crate number in the range 0:15.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, GradedLAMNumber, CrateNumber : INTEGER2;
...
AssignCamacLam(DeviceNumber, GradedLAMNumber, CrateNumber);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNo  COMP.
01  GradedLAMNumber  COMP.
01  CrateNo  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "AssignCamacLam" USING DeviceNo, GradedLAMNumber, CrateNo.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNo, GradedLAMNumber, CrateNo
      ...
      Monitor_Call('AssignCamacLam', DeviceNo, GradedLAMNumber, CrateNo)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

154B **AssignCamacLam** **ASSIG**

Continued from previous page...

PLANC

```
INTEGER : DeviceNo, GradedLAMNumber, CrateNo
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('AssignCamacLam', DeviceNo, GradedLAMNumber, CrateNo)
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
GradedLAMNumber : W  BLOCK  1
CrateNo : W  BLOCK  1
ErrCode : W  BLOCK  1
AssignCamacLam : EQU 37B9 + 154B
        ...
        CALLG AssignCamacLam, 3, DeviceNo, GradedLAMNumber, CrateNo
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     GRLAM    %Graded LAM.
        LDA     CRATE    %Crate number.
        COPY    SA  DD
        LDA     DEVNO    %Logical device number.
        MON     154      %Monitor call AssignCamacLam.
        JAN     ERROR    %Error if register A is negative.
        ...
ERROR,  ...              %Error number in register A.
        ...
GRLAM,  ...
DEVNO,  ...
CRATE,  ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

REENT **ATTACHSEGMENT** 167B

Attaches a reentrant segment to your two current segments. The address areas of the segments may overlap. Pages in the reentrant segment may be accessed for reading, writing, or fetching instructions. When written to, a page loses its reentrancy. It is stored on one of your current overlapping segments.

- The segment is treated as reentrant only for the RT programs which have attached it.
- Parts of the reentrant segment may not be outside the address area of your current segments. OUTSIDE SEGMENT BOUNDS is output if these parts are written to.

See also ReentrantSegment.

PARAMETERS

→ Segment number of the reentrant segment. See @LIST-REENTRANT.

PASCAL

```
SegmentNumber : INTEGER2;
...
AttachSegment(SegmentNumber);
```

COBOL

```
01  SegmentNumber  COMP.
    ...
    MONITOR-CALL "AttachSegment" USING SegmentNumber.
```

FORTRAN

```
      INTEGER  SegmentNumber
      ...
      Monitor_Call('AttachSegment', SegmentNumber)
```

ND-100 All users All programs

167B **ATTACHSEGMENT** **RESET**

Continued from previous page...

PLANC

```
INTEGER : SegmentNumber
...
Monitor_Call('AttachSegment', SegmentNumber)
```

ASSEMBLY-500

Not available.

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     167     %Monitor call AttachSegment.
        ...
PAR,    SEGNO           %Segment number
        ...
SEGNO,  ...
```

ND-100 **All users** **All users**

WAITF — AWAITFILETRANSFER — 121B

Checks that a data transfer to or from a mass storage file is completed. The monitor call is relevant to ReadFromFile and WriteToFile operations. These data transfers are carried out independently of the CPU.

- You may specify that the program should wait if the transfer is not ready. It is set in the I/O wait state.

See also AwaitTransfer.

PARAMETERS

→ File number. See OpenFile.
→ Wait flag. If 0, the program waits until the data transfer is completed. Other values return a value showing the state of the transfer. Programs on the ND-500 do not wait.
← State of transfer. 0 means that transfer is finished. -1 means not finished. Values greater than 0 are standard error codes. See appendix A. The error codes relate to AwaitFileTransfer, not ReadFromDisk or WriteToDisk.

PASCAL

```
FileNumber, ReturnFlag, Status : INTEGER2;
...
AwaitFileTransfer(FileNumber, ReturnFlag, Status);
```

COBOL

```
01  FileNumber  COMP.
01  ReturnFlag  COMP.
01  Status  COMP.
    ...
    MONITOR-CALL "AwaitFileTransfer" USING FileNumber, ReturnFlag, Status.
```

FORTRAN

```
      INTEGER  FileNumber, ReturnFlag, Status
      ...
      Monitor_Call('AwaitFileTransfer', FileNumber, ReturnFlag, Status)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

121B AWAITFILETRANSFER WAITF

Continued from previous page...

PLANC

```
INTEGER : FileNumber, ReturnFlag, Status
...
Monitor_Call('AwaitFileTransfer', FileNumber, ReturnFlag, Status)
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
ReturnFlag : W  BLOCK  1
Status : W  BLOCK  1
AwaitFileTransfer : EQU 37B9 + 121B
        ...
        CALLG AwaitFileTransfer, 2, FileNumber, ReturnFlag
        IF K GO Error
        ...
Error,  W1 =: Status                %Status is returned in W1 register.
```

MAC

```
        LDA     (PAR        %Load register A with address of parameter list.
        MON     121         %Monitor call AwaitFileTransfer.
        STA     STAT        %Store returned status.
        JAP     ERROR       %Handle error if STAT is greater than 0.
        ...
ERROR,  ...                 %Handle the error with error number in STAT.
        ...
STAT,   0
PAR,    FILNO               %File number.
        FLAG                %Return flag.
        ...
FILNO,  ...
FLAG,   0
```

ND-100 and ND-500 User RT and user SYSTEM RT programs

AWAITF — AWAITTRANSFER — 431B

Checks that a data transfer to or from a mass storage file is completed. The monitor call is relevant to DeviceFunction, ReadFromFile and WriteToFile operations. These are carried out independently of the CPU. The number of bytes read or written is returned.

- You may specify that the program should wait if the transfer is not ready. It is set in the I/O wait state.

See also AwaitFileTransfer.

PARAMETERS

→ File number: See OpenFile.
→ Wait flag. If 0, the program waits until the data transfer is completed. Other values return a value showing the state of the transfer. ND-500 programs do not wait.
← Number of bytes transferred.

PASCAL

```
FileNumber, WaitFlag, NoOfBytes : LONGINT;
...
AwaitTransfer(FileNumber, WaitFlag, NoOfBytes);
```

COBOL

```
01  FileNumber  COMP.
01  WaitFlag  COMP.
01  NoOfBytes  COMP.
    ...
    MONITOR-CALL "AwaitTransfer" USING FileNumber, WaitFlag, NoOfBytes.
```

FORTRAN

```
      INTEGER  FileNumber, WaitFlag, NoOfBytes
      ...
      Monitor_Call('AwaitTransfer', FileNumber, WaitFlag, NoOfBytes)
```

ND-500 | All users | All programs

431B AWAITTRANSFER MWAITF

Continued from previous page...

PLANC

```
INTEGER : FileNumber, WaitFlag, NoOfBytes
...
Monitor_Call('AwaitTransfer', FileNumber, WaitFlag, NoOfBytes)
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
WaitFlag : W  BLOCK  1
NoOfBytes : W  BLOCK  1
ErrCode : W  BLOCK  1
AwaitTransfer : EQU 37B9 + 431B
        ...
        CALLG AwaitTransfer, 3, FileNumber, WaitFlag, NoOfBytes
        IF K GO Error
        ...
Error,  W1 =: ErrCode
```

MAC

Not available.

ND-500 All users All users

DCLOS — BACKUPCLOSE — 252B

Closes a file. The version number and the last date accessed are unchanged. The number of pages in temporary files and spooling files is not affected.

- This monitor call is mainly used by the BACKUP-SYSTEM.

See also CloseFile.

PARAMETERS

→ File number of the opened file. See OpenFile.
→ Modified flag. If 0, the file is not marked as modified.
← Standard error code. See appendix A.

PASCAL

```
FileNumber, Flag : INTEGER2;
...
BackupClose(FileNumber, Flag);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNumber  COMP.
01  Flag  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "BackupClose" USING FileNumber, Flag.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNumber, Flag
      ...
      Monitor_Call('BackupClose', FileNumber, Flag)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

252B **BACKUPCLOSE** **BCLOS**

Continued from previous page...

PLANC

```
INTEGER : FileNumber, Flag
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('BackupClose', FileNumber, Flag)
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
Flag : W  BLOCK  1
ErrCode : W  BLOCK  1
BackupClose : EQU 37B9 + 252B
        ...
        CALLG BackupClose, 2, FileNumber, Flag
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     FILNO    %File number returned from earlier open.
        LDA     FLAG     %Reset modified flag.
        MON     252      %Monitor call BackupClose.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
FLAG,   ...
```

ND-100 and ND-500 **All users** **All users**

BATCHMODEECHO

Controls echo of input and ouput if the program is executed in a batch or mode job. The purpose is to allow the program to communicate with the terminal in mode jobs.

- The state set by this monitor call is reset when the next batch or mode job starts.
- This monitor call is mainly used by the JEC system.

See also SetEcho.

PARAMETERS

→ Bit mask to set the echo.
Bit 0 set to 1 means no echo in batch and mode executions.
Bit 1 set to 1 means output on terminal from mode executions.
Bit 2 set to 1 means input from terminal in mode executions.
If the parameter is -1, the bit mark from the previous batch or mode job is returned in the A register.

PASCAL

```
ControlBitmask : INTEGER2;
...
BatchModeEcho(ControlBitmask);
```

COBOL

```
01  ControlBitmask  COMP.
    ...
    MONITOR-CALL "BatchModeEcho" USING ControlBitmask.
```

FORTRAN

```
      INTEGER  ControlBitmask
      ...
      Monitor_Call('BatchModeEcho', ControlBitmask)
```

ND-100 and ND-500 | All users | Background programs

325B **BATCHMODEECHO** NBSCH

Continued from previous page...

PLANC

```
INTEGER : ControlBitmask
...
Monitor_Call('BatchModeEcho', ControlBitmask)
```

ASSEMBLY-500

```
OldMask : W BLOCK 1          %The lower 16 bit of the I1 register holds the
                             % control bit mask from the previous BatchModeEcho
                             % if ControlBitMask was -1.
ControlBitmask : W  BLOCK  1
ErrCode : W  BLOCK  1
BatchModeEcho : EQU 37B9 + 325B
        ...
        CALLG BatchModeEcho, 1, ControlBitmask
        IF K GO Error
        W1=:OldMask
        ...
Error,
```

MAC

```
        LDA     CTRL     %Load register A with control bitmask.
        MON     325      %Monitor call BatchModeEcho.
        ...
CTRL,   ...
```

ND-100 and ND-500 | All users | Background programs

BCNAF1 BCNAF1Camac 415B

Special CAMAC monitor call for the ND-500.

- See the NORD PROCESS I/O SOFTWARE GUIDE (ND-60.093) for details.

See also CamacFunction, AssignCamacLam, BCNAFCamac, CamacIOInstruction, and CamacGlRegister.

PARAMETERS

→ Function
→ Address
→ Data
← Status

PASCAL

```
Funct, Address, Data, Status : LONGINT;
...
BCNAF1Camac(Funct, Address, Data, Status);
```

COBOL

```
01  Func  COMP.
01  Address  COMP.
01  Data  COMP.
01  Status  COMP.
    ...
    MONITOR-CALL "BCNAF1Camac" USING Func, Address, Data, Status.
```

FORTRAN

```
      INTEGER  Func, Address, Data, Status
      ...
      Monitor_Call('BCNAF1Camac', Func, Address, Data, Status)
```

ND-500 User RT and user SYSTEM RT programs

415B	BCNAF1Camac	BCNAF1

Continued from previous page...

PLANC

```
INTEGER : Func, Address, Data, Status
...
Monitor_Call('BCNAF1Camac', Func, Address, Data, Status)
```

ASSEMBLY-500

```
Func : W  BLOCK  1
Address : W  BLOCK  1
Data : W  BLOCK  1
Status : W  BLOCK  1
BCNAF1Camac : EQU 37B9 + 415B
        ...
        CALLG BCNAF1Camac, 4, Func, Address, Data, Status
```

MAC

Not available.

ND-500	User RT and user SYSTEM	RT programs

BCNAF BCNAFCamac 414B

Special CAMAC function on the ND-500.

- See the NORD PROCESS I/O SOFTWARE GUIDE (ND-60.093) for details.

See also CamacFunction, AssignCamacLam, BCNAF1Camac, CamacIOInstruction, and CamacGlRegister.

PARAMETERS

→ Function
→ Address
→ Data
← Status

PASCAL

```
Func, Address, Data, Status : LONGINT;
...
BCNAFCamac(Func, Address, Data, Status);
```

COBOL

```
01  Func   COMP.
01  Address  COMP.
01  Data   COMP.
01  Status   COMP.
    ...
    MONITOR-CALL "BCNAFCamac" USING Func, Address, Data, Status.
```

FORTRAN

```
      INTEGER  Func, Address, Data, Status
      ...
      Monitor_Call('BCNAFCamac', Func, Address, Data, Status)
```

ND-500 User RT and user SYSTEM RT programs

414B **BCNAFCamac** **BCNAF**

Continued from previous page...

PLANC

```
INTEGER : Func, Address, Data, Status
...
Monitor_Call('BCNAFCamac', Func, Address, Data, Status)
```

ASSEMBLY-500

```
Func : W  BLOCK  1
Address : W  BLOCK  1
Data : W  BLOCK  1
Status : W  BLOCK  1
BCNAFCamac : EQU 37B9 + 414B
        ...
        CALLG BCNAFCamac, 4, Func, Address, Data, Status
```

MAC

Not available.

ND-500 **User RT and user SYSTEM** **RT programs**

CALLCOMMAND 70B

Executes a SINTRAN III command from a program. The program terminates if an error occurs in the command.

- Some commands may destroy your program. Use care with commands which affect your program's memory area.
- Some commands have output, eg., @LIST-FILES. Background programs display the output on the terminals.
- Use SuspendProgram to wait a second between two CallCommands which depend on each other, eg., CreateFile and OpenFile.
- The following commands are allowed from ND-500 programs: @DATCL, @COPY, @COPY-FILE, @SCHEDULE, @HOLD, @TERMINAL-MODE, @OPERATOR, @WAIT-FOR-OPERATOR, @SET-TERMINAL-TYPE, @GET-TERMINAL-TYPE, @CREATE-FILE, @EXPAND-FILE, @DELETE-FILE, @RENAME-FILE, @LIST-FILE, @FILE-STATISTICS, @OPEN-FILE, @CONNECT-FILE, @SET-FILE-ACCESS, @CLOSE-FILE, @LIST-OPEN-FILE, @SET-BLOCK-SIZE, @SET-PERMANENT-OPEN, @SET-BYTE-POINTER, @SET-BLOCK-POINTER, @APPEND-SPOOLING-FILE, @DELETE-SPOOLING-FILE, @SET-TEMPORARY-FILE, and @SCRATCH-OPEN.

See also ExecuteCommand and SetCommandBuffer. ExecuteCommand does not terminate the program if an error occurs.

PARAMETERS

→ Command with parameters, eg., "LIST-FILES :TEXT,,". Do not include the @ character.

PASCAL

```
Command : PACKED ARRAY [0..79] OF CHAR;
...
CallCommand(Command);
```

COBOL

```
01  Command   PIC X(100).
    ...
    MONITOR-CALL "CallCommand" USING Command.
```

FORTRAN

```
      CHARACTER  Command*80
      ...
      Monitor_Call('CallCommand', Command(1:80))
```

ND-100 and ND-500 | All users | Background programs

70B **CALLCOMMAND** **COMND**

Continued from previous page...

PLANC

```
BYTES : Command(0:79)
...
Monitor_Call('CallCommand', Command)
```

ASSEMBLY-500

```
Command : STRINGDATA  'CLOSE-FILE 102'''
ErrCode : W BLOCK 1
CallCommand : EQU 37B9 + 70B
        ...
        CALLG CallCommand, 1, Command
        IF K GO Error
        ...
Error,  W1 =: ErrCode
```

MAC

```
        LDA     (CMND    %Address of string with command to be executed.
        MON     70       %Monitor call CallCommand.
        ...
CMND,   'CLOSE-FILE 102' %Execute @CLOSE-FILE 102
```

ND-100 and ND-500 **All users** **Background programs**

CAMAC — CamacFunction — 147B

Operate the CAMAC, ie., execute NAF. See the NORD PROCESS I/O SOFTWARE GUIDE (ND-60.093).

See also AssignCamacLam, BCNAFCamac, BCNAF1Camac, CamacIOInstruction, and CamacGlRegister.

PARAMETERS

↔ Input of date if write. Output if read.
← Return status.
→ Crate number.
→ Station number.
→ Subaddress.
→ Function.

PASCAL

```
DataWord, RetStatus, CrateNo, StationNo, Subaddress, Func : INTEGER2;
...
CamacFunction(DataWord, RetStatus, CrateNo, StationNo, Subaddress, Func);
```

COBOL

```
01  DataWord  COMP.
01  RetStatus  COMP.
01  CrateNo  COMP.
01  StationNo  COMP.
01  Subaddress  COMP.
01  Func  COMP.
    ...
    MONITOR-CALL "CamacFunction" USING DataWord, RetStatus,
                 CrateNo, StationNo, Subaddress, Func.
```

FORTRAN

```
      INTEGER  DataWord, RetStatus, CrateNo, StationNo, Subaddress, Func
      ...
      Monitor_Call('CamacFunction', DataWord, RetStatus, CrateNo,
     C                  StationNo, Subaddress, Func)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

147B **CAMACFUNCTION** **CAMAC**

Continued from previous page...

PLANC

```
INTEGER : DataWord, RetStatus, CrateNo, StationNo, Subaddress, Func
...
Monitor_Call('CamacFunction', DataWord, RetStatus, CrateNo, &
                StationNo, Subaddress, Func)
```

ASSEMBLY-500

```
DataWord : W  BLOCK  1
RetStatus : W  BLOCK  1
CrateNo : W  BLOCK  1
StationNo : W  BLOCK  1
Subaddress : W  BLOCK  1
Func : W  BLOCK  1
ErrCode W BLOCK 1
CamacFunction : EQU 37B9 + 147B
        ...
        CALLG CamacFunction, 6, DataWord, RetStatus, CrateNo, &
                                StationNo, Subaddress, Func
        IF K GO Error
        ...
Error,  W1 =: ErrCode
```

MAC

```
        LDT     DATA    %Data if write.
        LDA     CRATE   %Crate number.
        COPY    SA  DD
        LDA     CTRL    %Load register A.
        MON     147     %Monitor call CamacFunction.
        STT     DATA    %Store data if read.
        STX     STAT    %Store status.
        ...
STAT,   0
DATA,   ...
CRATE,  ...
CTRL,   ...             %CAMAC control: station number/subaddress/function.
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

GL CamacGlRegister 150B

Read the CAMAC GL register or the last CAMAC identification number. See also NORD PROCESS I/O SOFTWARE GUIDE (ND-60.093).

See also CamacFunction, AssignCamacLam, BCNAFCamac, BCNAF1Camac, and CamacIOInstruction.

PARAMETERS

→ Flag. -1 means read last ident. Other values means read GL register.
→ Create number.

PASCAL

```
Flag, CrateNo : INTEGER2;
...
CamacGlRegister(Flag, CrateNo);
```

COBOL

```
01  Flag   COMP.
01  CrateNo  COMP.
    ...
    MONITOR-CALL "CamacGlRegister" USING Flag, CrateNo.
```

FORTRAN

```
      INTEGER  Flag, CrateNo
      ...
      Monitor_Call('CamacGlRegister', Flag, CrateNo)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

150B CAMACGLREGISTER GL

Continued from previous page...

PLANC

```
INTEGER : Flag, CrateNo
...
Monitor_Call('CamacGlRegister', Flag, CrateNo)
```

ASSEMBLY-500

```
Flag : W  BLOCK  1
CrateNo : W  BLOCK  1
ErrCode : W  BLOCK  1
CamacGlRegister : EQU 37B9 + 150B
        ...
        CALLG CamacGlRegister, 2, Value, CrateNo
        IF K GO Error,
        ...
Error,  W! =: ErrCode
```

MAC

```
        LDA     CRATE    %Crate number.
        COPY    SA  DD
        LDA     FUNC     %Function.
        MON     150      %Monitor call CamacGlRegister.
        STA     RETUR    %Store result.
        ...
RETUR,  0
FUNC,   ...              %Input: function / Output: last identification
CRATE,  ...              %                      or GL register.
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

IOXN CamacIOInstruction 153B

Executes a single IOX instruction for CAMAC. See also the NORD PROCESS I/O SOFTWARE GUIDE (ND-60.093)

See also CamacFunction, AssignCamacLam, BCNAFCamac, BCNAF1Camac, and CamacGlRegister.

PARAMETERS

↔ Input of data if write. Output if read.
⟶ Physical device number in the range 2000B-4000B.

PASCAL

```
DataWord, IOXCode : INTEGER2;
...
CamacIOInstruction(DataWord, IOXCode);
```

COBOL

```
01  DataWord  COMP.
01  IOXCode  COMP.
    ...
    MONITOR-CALL "CamacIOInstruction" USING DataWord, IOXCode.
```

FORTRAN

```
      INTEGER  DataWord, IOXCode
      ...
      Monitor_Call('CamacIOInstruction', DataWord, IOXCode)
```

ND-100 and ND-500 | **User RT and user SYSTEM** | **RT programs**

153B **CAMACIOINSTRUCTION** **IOXM**

Continued from previous page...

PLANC

```
INTEGER : DataWord, IOXCode
...
Monitor_Call('CamacIOInstruction', DataWord, IOXCode)
```

ASSEMBLY-500

```
DataWord : W  BLOCK  1
IOXCode : W  BLOCK  1
ErrCode : W  BLOCK  1
CamacIOInstruction : EQU 37B9 + 153B
        ...
        CALLG CamacIOInstruction, 2, DataWord, IOXCode
        ...
Error,  W1 =: ErrCode
```

MAC

```
        LDA     DEVNO    %Hardware device number.
        COPY    SA  DD
        LDA     DATA     %Data if write.
        MON     153      %Monitor call CamacIOInstruction.
        STA     DATA     %Store data if read.
        ...
DATA,   ...
DEVNO,  ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

SPCHG ChangeSegment 337B

Changes the segment and the page table your program uses. The monitor call is similar to JumpToSegment and ExitFromSegment. In addition, you may change the two page tables in use.

- You cannot change reentrant segments.

See also JumpToSegment and ExitFromSegment.

PARAMETERS

→ Function.

Bit 15 in the D register equal to 0 means JumpToSegment. Bit 0:1 should contain the new alternative page table and bit 2:3 the new normal page table. The T register contains the address of a parameter list. The parameter list consists of the start address and the segment numbers. The execution continues after this monitor call when an ExitFromSegment function is performed. The D register then contains the old page tables.

Bit 15 in the D register equal to 1 means ExitFromSegment. Bit 0:1 should contain the old alternative page table and bit 2:3 the old normal page table.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 User RT and user SYSTEM RT programs

337B CHANGESEGMENT SPCHG

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Not available.

MAC

```
%Code on initial segment. JumpToSegment is performed.
        SAA     3       %Page table 1 and 2 in A register.
        COPY    SA  DD  %Copy to D register.
        LDT     (PAR    %Address of parameter list in T register.
        MON     337     %ChangeSegment with bit 15 in D register = 0.
CONT,   ...             %Execution continues here after ExitFromSegment.
        ...
PAR,    SUBRO           %Start address on the new segment.
        100201          %Segment 200B and 201B.

%Code on new segment. ExitFromSegment is performed.
SUBRO,  STT     SAVET   %Save T, L, and D registers.
        COPY    SL  DT
        STT     SAVEL
        COPY    SD  DT
        STT     SAVED
        ...
        LDT     SAVED   %Restore D register
        COPY    ST  DD
        BSET ONE DD 170 %Set bit 15 for ExitFromSegment function.
        LDT     SAVEL   %Restore L and T registers.
        COPY    ST  DL
        LDT     SAVET
        MON     337     %ChangeSegment to return to original segment.
```

MONRP — CHECKMONCALL — 312B

Some monitor calls are optional or only available in later versions of SINTRAN III. This monitor call checks if a monitor call exists in your particular SINTRAN III system. Optional monitor calls are included or left out when SINTRAN III is generated.

PARAMETERS

→ Monitor call number.
← Address of the monitor call entry. 0 means not implemented.

PASCAL

```
MonCallNumber, MonCallEntry : INTEGER2;
...
CheckMonCall(MonCallNumber, MonCallEntry);
```

COBOL

```
01  MonCallNumber  COMP.
01  MonCallEntry  COMP.
    ...
    MONITOR-CALL "CheckMonCall" USING MonCallNumber, MonCallEntry.
```

FORTRAN

```
      INTEGER  MonCallNumber, MonCallEntry
      ...
      Monitor_Call('CheckMonCall', MonCallNumber, MonCallEntry)
```

ND-100 and ND-500 | All users | All programs

312B **CHECKMONCALL** MONF

Continued from previous page...

PLANC

```
INTEGER : MonCallNumber, MonCallEntry
...
Monitor_Call('CheckMonCall', MonCallNumber, MonCallEntry)
```

ASSEMBLY-500

```
MonCallNumber : W  BLOCK  1
MonCallEntry : W  BLOCK  1
CheckMonCall : EQU 37B9 + 312B
        ...
        CALLG CheckMonCall, 2, MonCallNumber, MonCallEntry
        ...
```

MAC

```
        LDA     MONNO    %Load register A with monitor call number.
        MON     312      %Monitor call CheckMonCall.
        ...              %Return: Monitor call not implemented in system.
        STA     ENTRY    %Skipreturn: Monitor call is implemented.
        ...
MONNO,  ...              %Monitor call number.
ENTRY,  0                %Monitor call entry returned if implemented.
```

ND-100 and ND-500 All users All users

CAPCLE — ClearCapability — 4248

Clears a capability. A capability describes each logical segment in a domain. The protection of the segment is removed. See the ND-500 LOADER/MONITOR (ND-60.136).

- The logical segment is available for other physical segments.

See also CopyCapability.

PARAMETERS

→ Logical segment number in your domain.
→ Segment type. Use 0 for data segments, and 1 for program segments.

PASCAL

```
LogicalSegmentNo, SegType : LONGINT;
...
ClearCapability(LogicalSegmentNo, SegType);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  LogicalSegmentNo  COMP.
01  SegType  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "ClearCapability" USING LogicalSegmentNo, SegType.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ... .
```

FORTRAN

```
      INTEGER  LogicalSegmentNo, SegType
      ...
      Monitor_Call('ClearCapability', LogicalSegmentNo, SegType)
      IF (ErrCode .NE. 0) THEN ...
```

ND-500 | All users | All programs

424B ClearCapability CAPCLR

Continued from previous page...

PLANC

```
INTEGER : LogicalSegmentNo, SegType
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('ClearCapability', LogicalSegmentNo, SegType)
```

ASSEMBLY-500

```
LogicalSegmentNo : W  BLOCK  1
SegType : W  BLOCK  1
ErrCode : W  BLOCK  1
ClearCapability : EQU 37B9 + 424B
        ...
        CALLG ClearCapability, 2, LogicalSegmentNo, SegType
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

Not available.

ND-500 All users All users

CIBUF — CLEARINBUFFER — 13B

Clears a device input buffer. Input from character devices, eg., terminals, are temorarily stored in this buffer.

- You can use logical device number 1 for your own terminal in background programs.

See also @CLEAR-DEVICE, ClearOutBuffer, and DeviceControl.

PARAMETERS

→ Logical device number. See appendix B.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber : INTEGER2;
...
ClearInBuffer(DeviceNumber);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "ClearInBuffer" USING DeviceNumber.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber
      ...
      Monitor_Call('ClearInBuffer', DeviceNumber)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

13B ClearInBuffer CIBUF

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('ClearInBuffer', DeviceNumber)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
ErrCode : W  BLOCK  1
ClearInBuffer : EQU 37B9 + 13B
        ...
        CALLG ClearInBuffer, 1, DeviceNumber
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Load register T with logical device number.
        MON     13       %Monitor call ClearInBuffer.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
```

ND-100 and ND-500	All users	All users

CIBUF — CLEAROUTBUFFER — 14B

Clears a device output buffer. Output to character devices, eg., terminals, are temorarily stored in this buffer.

- You can use logical device number 1 for your own terminal in background programs.

See also @CLEAR-DEVICE, ClearInBuffer, and DeviceControl.

PARAMETERS

→ Logical device number. See appendix B.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber : INTEGER2;
...
ClearOutBuffer(DeviceNumber);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "ClearOutBuffer" USING DeviceNumber.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber
      ...
      Monitor_Call('ClearOutBuffer', DeviceNumber)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

14B CLEAROUTBUFFER CIBUF

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('ClearOutBuffer', DeviceNumber)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
ErrCode : W  BLOCK  1
ClearOutBuffer : EQU 37B9 + 14B
        ...
        CALLG ClearOutBuffer, 1, DeviceNumber
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Load register T with logical device number.
        MON     14       %Monitor call ClearOutBuffer.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
```

ND-100 and ND-500 | All users | All users

CLOSE — CloseFile — 43B

Closes one or more files. Files must be opened before they are accessed. Afterwards they should be closed.

- CloseFile also resets peripheral files. This is similar to DeviceControl with control flag -1.
- Files are closed when your program terminates.

See also OpenFile, CloseSpoolingFile, BackupClose, and @CLOSE-FILE.

PARAMETERS

→ File number returned when the file was opened. You close all your files which are not set permanently open, with -1. The block size of all scratch files which are permanently opened is set to 400B. Use -2 to close all your files, including those set permanently open.

← Standard error code. See appendix A.

PASCAL

```
FileNumber : INTEGER2;
...
CloseFile(FileNumber);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNumber  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "CloseFile" USING FileNumber.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNumber
      ...
      Monitor_Call('CloseFile', FileNumber)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

43B CLOSEFILE CLOSE

Continued from previous page...

PLANC

```
INTEGER : FileNumber
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('CloseFile', FileNumber)
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
ErrCode : W  BLOCK  1
CloseFile : EQU 37B9 + 43B
        ...
        CALLG CloseFile, 1, FileNumber
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     FILNO    %Load register T with file number.
        MON     43       %Monitor call CloseFile.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
```

ND-100 and ND-500 | All users | All users

SPCLO CLOSESPOOLINGFILE 40B

Appends an opened file to a spooling queue. You specify a text to be printed on the error device when the file is to be printed.

- If the file is not a spooling file, a normal close is performed.

See also CloseFile, AppendSpooling and @DEFINE-SPOOLING-CONDITIONS.

PARAMETERS

→ File number given when the file was opened.
→ Text to be output on the error device.
→ Number of print copies.
→ Print flag. If 0, the the text is only output if required by @DEFINE-SPOOLING-CONDITIONS. If not 0, the file is printed unconditionally. A stop print condition occurs before printing.
← Standard error code. See appendix A.

PASCAL

```
FileNo, NoOfCopies, PrintFlag : INTEGER2;
UserText : PACKED ARRAY [0..79] OF CHAR;
...
CloseSpoolingFile(FileNo, UserText, NoOfCopies, PrintFlag);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNo  COMP.
01  NoOfCopies  COMP.
01  PrintFlag  COMP.
01  UserText   PIC X(100).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "CloseSpoolingFile" USING FileNo, UserText,
                                           NoOfCopies, PrintFlag.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNo, NoOfCopies, PrintFlag
      CHARACTER  UserText*80
      ...
      Monitor_Call('CloseSpoolingFile', FileNo, UserText(1:80),
     C                  NoOfCopies, PrintFlag)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

40B **CLOSESPOOLINGFILE** **SPCLO**

Continued from previous page...

PLANC

```
INTEGER : FileNo, NoOfCopies, PrintFlag
BYTES : UserText(0:79)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('CloseSpoolingFile', FileNo, UserText, NoOfCopies, PrintFlag)
```

ASSEMBLY-500

```
FileNo : W  BLOCK  1
NoOfCopies : W  BLOCK  1
PrintFlag : W  BLOCK  1
UserText : STRINGDATA  'Printing file.'''
ErrCode : W  BLOCK  1
CloseSpoolingFile : EQU 37B9 + 40B
        ...
        CALLG CloseSpoolingFile, 4, FileNo, UserText, NoOfCopies, PrintFlag
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     FILNO    %File number returned from earlier open.
        LDA     NOCOP    %Number of copies.
        COPY    SA  DD
        LDA     FLAG     %Condition flag.
        LDX     (TEXT    %Address of text to be sent to error device.
        MON     40       %Monitor call CloseSpoolingFile.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
TEXT,   'GUMMED LABELS'  %Message when the file is to be printed.
NOCOP,   ...
FLAG,   0                %Text is only written if required by
                         %    @DEFINE-SPOOLING-CONDITIONS.
```

ND-100 and ND-500 **All users** **All users**

CAPCOP CopyCapability 423B

Copies a capability for a segment. The segment itself is also copied. A capability describes each logical segment in a domain.

- The destination segment number must be unused.

See also ClearCapability.

PARAMETERS

→ Source logical segment number.
→ Type of source segment. Use 0 for data segments, and 1 for program segments.
→ Destination logical segment number. Use 0 to get the first unused segment.
→ Type of destination segment. Use 0 for data segments, and 1 for program segments.
→ Access. Use 0 not to change the access. Use 1 to set read access only. Read and write access is set by 2.
← Returned logical segment number if 0 was specified as destination.

PASCAL

```
SourceSegNo, SourceType, DestSegNo, DestType, AccCode, RetSegNo : LONGINT;
...
CopyCapability(SourceSegNo, SourceType, DestSegNo,
               DestType, AccCode, RetSegNo)
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  SourceSegNo  COMP.
01  SourceType  COMP.
01  DestSegNo  COMP.
01  DestType  COMP.
01  AccCode  COMP.
01  RetSegNo  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "CopyCapability" USING SourceSegNo, SourceType,
                 DestSegNo, DestType, AccCode, RetSegNo.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  SourceSegNo, SourceType, DestSegNo, DestType
      INTEGER  AccCode, RetSegNo
      ...
      Monitor_Call('CopyCapability', SourceSegNo, SourceType,
     C                 DestSegNo, DestType, AccCode, RetSegNo)
      IF (ErrCode .NE. 0) THEN ...
```

ND-500 All users All programs

423B **CopyCapability** **CAPCOP**

Continued from previous page...

PLANC

```
INTEGER : SourceSegNo, SourceType, DestSegNo, DestType, AccCode, RetSegNo
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('CopyCapability', SourceSegNo, SourceType, DestSegNo, &
                DestType, AccCode, RetSegNo)
```

ASSEMBLY-500

```
SourceSegNo : W  BLOCK  1
SourceType : W  BLOCK  1
DestSegNo : W  BLOCK  1
DestType : W  BLOCK  1
AccCode : W  BLOCK  1
RetSegNo : W  BLOCK  1
ErrCode : W  BLOCK  1
CopyCapability : EQU 37B9 + 423B
        ...
        CALLG CopyCapability, 6, SourceSegNo, SourceType, DestSegNo, &
                                 DestType, AccCode, RetSegNo
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

Not available.

ND-500 **All users** **All users**

COPAG — CopyPage — 251B

Copies file pages between two opened files. One of the files may be a magnetic tape or floppy disk with volume.

- This is a special monitor call used by the BACKUP-SYSTEM. No high level language interface exists. The use of the X and D registers is tailored for reading labels on magnetic tape.
- Copying stops at end-of-file, a non-existent page, or if a short magnetic tape record is found.
- CopyPage is only used for sequential copying, ie., a sequence of CopyPage calls must start with page number zero.
- Files should be opened for random read or write.

PARAMETERS

→ File number of source file in the T register. Use logical device number for magnetic tape and floppy disks.
→ File number of destination file in the A register. Use logical device number for magnetic tapes and floppy disks.
→ Address of 32-bit word with page address of first page to copy. The page address is the same for the source and destination files.
→ Address of buffer to receive short magnetic tape record if source file is magnetic tape. Use -1 if you do not want the short record returned.

Output parameters if normal return:
← Error number relating to the destination file in the A register. See appendix A.
Output parameters if skip return:
← Error number relating to the source file in the A register. If finished, end-of-file is returned. See appendix A.
Output parameters if double skip return. Unless the source is magnetic tape, a page is missing.
← Page number of the missing page in the A&D register.
← A set of contiguous pages can be missing. The T and X register contains the last page number missing in such a hole. This is only returned if the source is a directory.
← Number of 16-bit integers returned if a short magnetic tape record is found. Only for magnetic tape as source and the D register different from -1.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 and ND-500 | All users | All programs

251B CopyPage COPAG

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
SourceFile : W  BLOCK  1
DestFile : W  BLOCK  1
FirstPage : W  BLOCK  1
DestBuffer : W  BLOCK  1
FirstPageMiss : W  BLOCK  1
LastPageMiss : W  BLOCK  1
NoOfWord : W BLOCK 1                    %Dummy for mass storage files.
ErrCode : W  BLOCK  1
CopyPage : EQU 37B9 + 251B
        ...
        CALLG CopyPage, 7, SourceFile, DestFile, FirstPage, DestBuffer, &
                           FirstPageMiss, LastPageMiss, NoOfWords
        IF K GO ERROR ...
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     FL1NO    %Load file number of source file.
        LDA     (BUFF    %Memory address of buffer to receive record.
        COPY    SA  DD
        LDA     FL2NO    %Load file number of destination file.
        LDX     (PAGE    %Address of double word with page address.
        MON     251      %Monitor call CopyPage.
        JMP     ERROR    %Destination error, error number in register A.
        JMP     ERROR    %Source error, EOF if finished, error number in A.
        STD     PAGNO    %Normal return, store page number of missing page.
        STT     LAST     %Store page number of last missing page.
        STX     LAST + 1
        ...
ERROR,  ...
        ...
FL1NO,  ...
FL2NO,  ...
PAGE,   ...              %A double word.
        ...              %
BUFF,   0
        0                %Buffer to receive a short magtape record.
        ...
PAGNO,  0                %A double word.
        0                %
LAST,   0                %A double word.
        0                %
```

ND-100 and ND-500 | All users | All users

CRALF — CREATEFILE — 221B

Creates a file. The file may be indexed, contiguous, or allocated. Most files are indexed. The size of indexed files expands automatically when written to. Contiguous and allocated files have shorter access time.

- You need directory access to the user who owns the file.
- User SYSTEM and RT always have the owner's access rights.
- An indexed page not yet written to may be converted to a contiguous file. Use ExpandFile or @EXPAND-FILE.

See also @CREATE-FILE, @ALLOCATE-FILE, NewFileVersion, and ExpandFile.

PARAMETERS

→ File name. Default file type is :DATA.
→ Start address in the directory. Use 0 if you want to create a contiguous or indexed file.
→ Length of the file in pages. Use 0 if you want to create an indexed file.
← Standard error code. See appendix A.

PASCAL

```
StartAddress, NoOfPages : LONGINT;
FileName : PACKED ARRAY [0..63] OF CHAR;
...
CreateFile(FileName, StartAddress, NoOfPages);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  StartAddress  COMP PIC S9(10).
01  NoOfPages  COMP PIC S9(10).
01  FileName   PIC X(64).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "CreateFile" USING FileName, StartAddress, NoOfPages.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER*4  StartAddress, NoOfPages
      CHARACTER  FileName*64
      ...
      Monitor_Call('CreateFile', FileName(1:64), StartAddress, NoOfPages)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

221B **CREATEFILE** **CBALP**

Continued from previous page...

PLANC

```
INTEGER4 : StartAddress, NoOfPages
BYTES : FileName(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('CreateFile', FileName, StartAddress, NoOfPages)
```

ASSEMBLY-500

```
StartAddress : W  BLOCK  1
NoOfPages : W  BLOCK  1
FileName : STRINGDATA  'EXAMPLE:SYMB'''
ErrCode : W  BLOCK  1
CreateFile : EQU 37B9 + 221B
        ...
        CALLG CreateFile, 3, FileName, StartAddress, NoOfPages
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (FILE    %Address of file name string.
        LDD     START    %Load register AD with start address.
        LDT     (SIZE    %Address of  double word with number of pages.
        MON     221      %Monitor call CreateFile.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILE,   'EXAMPLE:SYMB'   %Create EXAMPLE:SYMB.
START,  ...              %Start address, ie. page address in directory.
        ...              %A double word.
SIZE,   ...              %File size as a double word.
        ...              %
```

ND-100 and ND-500 **All users** **All users**

ABSTR **DATATRANSFER** 131B

Transfers data between physical memory and a mass storage device, eg., a disk or magnetic tape. You may perform various device control functions. This monitor call is mainly used by the operating system itself.

- The monitor call and the parameters must reside in resident memory or on a fixed segment on protection ring 2, page table 0.
- For calls to magnetic tape or other devices where the last parameter contains number of bytes or words read or other return parameters, the parameter list must be in resident memory, ie., an area with fixed page index table contents.
- Parameters are fetched via the normal page table. In SINTRAN III, version K, they are fetched via the alternative page table.
- The physical memory area must be contiguous. Older versions of magnetic tapes or disk controllers cannot cross physical memory bank boundaries of 128 Kbytes. These magnetic tapes have ND numbers less than ND-537. The disks have ND numbers less than ND-559.

See also TransferData, ReadFromFile and WriteToFile. TransferData allows you to use any page table.

PARAMETERS

→ Logical device number. See appendix B.
→ Function code. See the tables on the following pages.
→ Physical memory address.
→ Sector address on the disk. See the tables on the following pages.
→ Number of sectors to transfer.
← Error code. Negative if error. Contains a hardware status.

PASCAL

```
DeviceNo, Func, BlockAddr, NoOfBlocks, Stat : INTEGER2;
SectorAddr : LONGINT;
...
DataTransfer(DeviceNo, Func, SectorAddr, BlockAddr, NoOfBlocks, Stat);
```

COBOL

```
01  DeviceNo  COMP.
01  Func  COMP.
01  BlockAddr  COMP.
01  NoOfBlocks  COMP.
01  Stat  COMP.
01  SectorAddr  COMP PIC S9(10).
    ...
    MONITOR-CALL "DataTransfer" USING DeviceNo, Func, SectorAddr,
                 BlockAddr, NoOfBlocks, Stat.
```

FORTRAN

```
      INTEGER  DeviceNo, Func, BlockAddr, NoOfBlocks, Stat
      INTEGER*4  SectorAddr
      ...
      Monitor_Call('DataTransfer', DeviceNo, Func, SectorAddr,
     C                  BlockAddr, NoOfBlocks, Stat)
```

ND-100 and ND-500 User RT and user SYSTEM RT programs

131B **DATATRANSFER** **ABSTR**

Continued from previous page...

PLANC

```
INTEGER : DeviceNo, Func, BlockAddr, NoOfBlocks, Stat
INTEGER4 : SectorAddr
...
Monitor_Call('DataTransfer', DeviceNo, Func, SectorAddr, &
                BlockAddr, NoOfBlocks, Stat)
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
Func : W  BLOCK  1
MemoryAddr : W  BLOCK  1     %ND-100 memory address.
BlockAddr : W  BLOCK  1
NoOfBlocks : W  BLOCK  1
Stat : W  BLOCK  1
DataTransfer : EQU 37B9 + 131B
        ...
        CALLG DataTransfer, 5, DeviceNo, Func, MemoryAddr, &
                               BlockAddr, NoOfBlocks
        IF K GO Error
        ...
Error,  W1 =: Stat
```

MAC

```
        LDT     DEVNO    %Logical device number.
        LDA     (PAR     %Load register A with address of parameter list.
        MON     131      %Monitor call DataTransfer.
        JAN     ERROR    %Error if register A is negative.
        ...              %Continue with processing.
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
PAR,    FUNC             %Function code etc.
        DMEM             %Memory address.
        BLOCK            %Block address.
        NOBLK            %Number of blocks to transfer.
        ...
FUNC,   ...
DMEM,   ...              %A double word.
        ...              %
BLOCK,  ...              %Double word if function 50, 51, or 60:63.
        ...
NOBLK,  ...
```

Function code for disks and floppy disks with 32 bits sector addresses:

60 - read with 32 bits disk address
61 - write with 32 bits disk address
63 - compare with 32 bits disk address. The function code for ECC and HAWK disks:

Bits	Value	Explanation
0-5	0B	Read
	1B	Write
	2B	Read block for parity check
	3B	Compare block
6-10	(0-3B)	Mass storage unit number.
11-13		Most significant 3 bits of the sector address. Not PHOENIX.
11-13	(0-4B)	Subunit number. PHOENIX only.
16	1B	PHOENIX
	0B	NOT PHOENIX.

The sector address for ECC and HAWK disks:

Bit	Value	Meaning
17	0	Removable surface.
	1	Fixed surface. HAWK and PHOENIX only.

The function code for floppy disk controllers. The asterisk (*) indicates that this applies to new controllers only.

Bits	Value	Explanation
0-5	0B	Read.
	1B	Write.
	2B	Read without transfering data, ie., find end-of-file mark.
	3B	Compare block.
	5B	Write end-of-file mark, (deleted record). *
	10B	Advance to end-of-file mark.
	11B	Reverse to end-of-file mark.
	13B	Rewind.
	15B	Backspace one record.
	16B	Advance one record.
	20B	Read status.
	24B	Read last status.
	40B	Select format.
	41B	Format floppy
	42B	Read format.
	43B	Read deleted record.
	44B	Write deleted record.
	54B	Copy floppy disk. *
	55B	Format floppy disk track. *
	56B	Check floppy disk. *
6-10	(0-3B)	Unit number.

The sector address should contain the number of blocks to transfer in read or write functions. The select format function requires the format number in the sector address.

Function code for VERSATEC:

Bits	Value	Explanation
0-5	20B	Read status.
	21B	Clear VERSATEC.
	24B	Read last status.
	30B	Set alphanumeric mode.
	31B	Set graphic mode.
	32B	Give form feed.

SBUSI DefaultRemoteSystem 314B

Sets default values for COSMOS remote file access. You can specify the default remote system, the remote user, and the remote user's passwords. The specified values are used when you omit values in a remote file access.

- Empty parameters remove previous default values. Default values are then the local user's name and passwords.
- SetRemoteMode switches the remote search on and off.

See also @SET-DEFAULT-REMOTE-SYSTEM and @RESET-DEFAULT-REMOTE-SYSTEM.

PARAMETERS

→ Remote system name.
→ User owning the files in the remote system.
→ The user's password.
→ The user's project password.
← Standard error code. See appendix A.

PASCAL

```
SystemName: PACKED ARRAY [0..15] OF CHAR;
UserName, Password, ProjPassword : PACKED ARRAY [0..15] OF CHAR;
...
DefaultRemoteSystem(SystemName, UserName, Password, ProjPassword);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  SystemName    PIC X(16).
01  UserName    PIC X(16).
01  Password    PIC X(16).
01  ProjPassword    PIC X(16).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "DefaultRemoteSystem" USING SystemName, UserName,
                 Password, ProjPassword.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      CHARACTER  SystemName*16
      CHARACTER  UserName*16, Password*16, ProjPassword*16
      ...
      Monitor_Call('DefaultRemoteSystem', SystemName(1:16),
     C                  UserName(1:16), Password(1:16), ProjPassword(1:16))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

314B DEFAULTREMOTESYSTEM SBUSI

Continued from previous page...

PLANC

```
BYTES : SystemName(0:15), UserName(0:15), Password(0:15), ProjPassword(0:15)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('DefaultRemoteSystem', SystemName, UserName, &
                Password, ProjPassword)
```

ASSEMBLY-500

```
SystemName : STRINGDATA  'SNORRE'''
UserName : STRINGDATA  'A-HANSEN'''
Password : STRINGDATA  'MAY'''
ProjPassword : STRINGDATA  'CHEESE'''
ErrCode : W  BLOCK  1
DefaultRemoteSystem : EQU 37B9 + 314B
        ...
        CALLG DefaultRemoteSystem, 4, SystemName, UserName, &
                                      Password, ProjPassword
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (SYS     %Address of remote system name.
        LDT     (USER    %Address of remote user identifier string.
        LDA     (PROJP   %Address of remote project password string.
        COPY    SA  DD
        LDA     (PASSW   %Address of remote user password string.
        MON     314      %Monitor call DefaultRemoteSystem.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
SYS,    'SNORRE'         %Set up SNORRE as default remote system.
USER,   'A-HANSEN'       %Set up A-HANSEN as default remote user.
PASSW,  'MAY'            %Set up MAY as default remote user password.
PROJP,  'CHEESE'         %Set up CHEESE as default remote project password.
```

ND-100 and ND-500 All users All users

DEBUG **DEFINEBREAKPOINT** 45B

Defines a breakpoint for debugging a program. The program stops at this address. The register contents are saved. Use SetBreakpoint to start the program to be debugged.

- Both the program executing this monitor call and the debug program must be loaded to the same address area.
- The program must place GetBreakpointInfo (153046B) in the breakpoint address before calling SetBreakPoint.
- This monitor call is mainly used by the MAC assembler.

See also GetBreakpointInfo and SetBreakpoint.

PARAMETERS

→ Program address to start execution when GetBreakpointInfo is called.
→ Buffer to receive the register contents.
← Standard error code. See appendix A.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 All users All programs

45B DEFINEBREAKPOINT DB00K

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
        LDT     (REGBL
        LDX     BRADR
        MON     45          %Monitor call DefineBreakpoint.
        ...
BRADR,  2345                %Breakpoint handler in 2345B.
REGBL,     0                %Buffer where register block of the debugged
*+10/                       %program is saved when GetBreakpointInfo is
                            %executed.
```

MAC

ND-100 All users All users

SET	DELAYSTART	101B

Starts an RT program after a specified time. The RT program is put in the time queue. It is moved to the execution queue after the specified period.

- RT programs already in the time queue are reinserted according to the new specifications.
- AdjustClock and @CLADJ do not affect the specified period.
- A period less than or equal to 0 moves the RT program to the execution queue the next time the basic time unit counter is incremented.

See also @SET and StartupTime.

PARAMETERS

→ Address of the RT description. Use 0 for the calling program.
→ The number of time units to stay in the time queue.
→ The type of time units. 1 = basic time units, ie., 1/50th of a second, 2 = seconds, 3 = minutes, 4 = hours.

PASCAL

```
RTProgram, TimeUnits, UnitType : INTEGER2;
...
DelayStart(RTProgram, TimeUnits, UnitType);
```

COBOL

```
01  RTProgram  COMP.
01  TimeUnits  COMP.
01  UnitType   COMP.
    ...
    MONITOR-CALL "DelayStart" USING RTProgram, TimeUnits, UnitType.
```

FORTRAN

```
      INTEGER  RTProgram, TimeUnits, UnitType
      ...
      Monitor_Call('DelayStart', RTProgram, TimeUnits, UnitType)
```

ND-100 and ND-500	User RT and user SYSTEM	RT programs

101B **DELAYSTART** **SET**

Continued from previous page...

PLANC

```
INTEGER : RTProgram, TimeUnits, UnitType
...
Monitor_Call('DelayStart', RTProgram, TimeUnits, UnitType)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
TimeUnits : W  BLOCK  1
UnitType : W  BLOCK  1
DelayStart : EQU 37B9 + 101B
        ...
        CALLG DelayStart, 3, RTProgram, TimeUnits, UnitType
```

MAC

```
        LDA     (PAR      %Load register A with address of parameter list.
        MON     101       %Monitor call DelayStart.
        ...
PAR,    RTPRO
        TIME
        BASE
        ...
RTPRO,  ...               %Address of RT description.
TIME,   ...               %Number of time units.
BASE,   ...               %Base time units.
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

MDLFI DELETEFILE 54B

Deletes a file. The pages of the file are released.

- You must have directory access to the file in order to delete it. RT programs can delete a file if user RT has directory access to it.
- Include a version number in the file name to delete specific versions of a file. Otherwise, all versions are deleted.

See also @DELETE-FILE, @DELETE-USERS-FILES, and CreateFile.

PARAMETERS

→ File name.
← Standard error code. See appendix A.

PASCAL

```
FileName : PACKED ARRAY [0..63] OF CHAR;
...
DeleteFile(FileName);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileName   PIC X(64).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "DeleteFile" USING FileName.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      CHARACTER  FileName*64
      ...
      Monitor Call('DeleteFile', FileName(1:64))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

54B **DELETEFILE** MDLP1

Continued from previous page...

PLANC

```
BYTES : FileName(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('DeleteFile', FileName)
```

ASSEMBLY-500

```
FileName : STRINGDATA  'EXAMPLE:SYMB'''
ErrCode : W  BLOCK  1
DeleteFile : EQU 37B9 + 54B
        ...
        CALLG DeleteFile, 1, FileName
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (FILE    %Load register X with address of file name.
        MON     54       %Monitor call DeleteFile.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILE,   'EXAMPLE:SYMB'   %Delete file EXAMPLE:SYMB.
```

ND-100 and ND-500 All users All users

DELPG **DELETEPAGE** 272B

Deletes pages from a file. Pages between two page numbers are removed.

- The file must be opened.

See also DeleteFile.

PARAMETERS

→ File number.
→ First page to be deleted.
→ Last page to be deleted.
← Number of pages deleted.
← Standard error code. See appendix A.

PASCAL

```
FileNo : INTEGER2;
FirstPage, LastPage, NoOfPages : LONGINT;
...
DeletePage(FileNo, FirstPage, LastPage, NoOfPages);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNo  COMP.
01  FirstPage  COMP PIC S9(10).
01  LastPage  COMP PIC S9(10).
01  NoOfPages  COMP PIC S9(10).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "DeletePage" USING FileNo, FirstPage, LastPage, NoOfPages.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNo
      INTEGER*4  FirstPage, LastPage, NoOfPages
      ...
      Monitor_Call('DeletePage', FileNo, FirstPage, LastPage, NoOfPages)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 **All users** **All programs**

272B **DELETEPAGE** **DELPG**

Continued from previous page...

PLANC

```
INTEGER : FileNo
INTEGER4 : FirstPage, LastPage, NoOfPages
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('DeletePage', FileNo, FirstPage, LastPage, NoOfPages)
```

ASSEMBLY-500

```
FileNo : W  BLOCK  1
FirstPage : W  BLOCK  1
LastPage : W  BLOCK  1
NoOfPages : W  BLOCK  1
ErrCode : W  BLOCK  1
DeletePage : EQU 37B9 + 272B
        ...
        CALLG DeletePage, 4, FileNo, FirstPage, LastPage, NoOfPages
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     FILNO    %File number.
        LDA     (FIRST   %Address of double word with first page.
        LDX     (LAST    %Address of double word with last page.
        MON     272      %Monitor call DeletePage.
        JMP     ERROR    %Error return from monitor call.
        STD     NODEL    %Normal return, store the number of pages deleted.
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
FIRST,  ...              %A double word.
        ...              %
LAST,   ...              %A double word.
        ...              %
NODEL,  0                %A double word.
        0                %
```

ND-100 and ND-500 **All users** **All users**

IOSET — DEVICECONTROL — 141B

Sets control information for a character device, eg., a terminal or a printer. The control information depends on the device.

- The device must be reserved. See ReserveResource.

See also DeviceFunction and @IOSET.

PARAMETERS

→ Logical device number. See appendix B. You cannot use 1 for your own terminal. Use ExecutionInfo to get its logical device number instead. File numbers are illegal.
→ Input or output part of the device. Use 0 for input and 1 for output.
→ Reserving RT program description. Use 0 for the calling program.
→ Control flag. Use -1 to reset the device. This sets card readers in ASCII mode. 0 sets ASCII mode without resetting the device. 1 sets binary mode. InByte will then return a 12 bit card column image.
← Error code. 0 means no errors. Illegal RT program description returns -1.

PASCAL

```
DeviceNo, IOFlag, RTProgram, CtrlFlag : INTEGER2;
...
DeviceControl(DeviceNo, IOFlag, RTProgram, CtrlFlag);
```

COBOL

```
01  DevNo  COMP.
01  IOFlag  COMP.
01  RTProgram  COMP.
01  CtrlFlag  COMP.
    ...
    MONITOR-CALL "DeviceControl" USING DevNo, IOFlag, RTProgram, CtrlFlag.
```

FORTRAN

```
      INTEGER  DeviceNo, IOFlag, RTProgram, CtrlFlag
      ...
      Monitor_Call('DeviceControl', DeviceNo, IOFlag, RTProgram,
                   CtrlFlag)
```

ND-100 and ND-500 | All users | All programs

141B **DEVICECONTROL** **IOSET**

Continued from previous page...

PLANC

```
INTEGER : DeviceNo, IOFlag, RTProgram, CtrlFlag
...
Monitor_Call('DeviceControl', DeviceNo, IOFlag, RTProgram,
                CtrlFlag)
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
IOFlag : W  BLOCK  1
RTProgram : W  BLOCK  1
CtrlFlag : W  BLOCK  1
ReturnStatus : W  BLOCK  1
DeviceControl : EQU 37B9 + 141B
        ...
        CALLG DeviceControl, 4, DeviceNo, IOFlag, RTProgram, CtrlFlag
        W1 =: ReturnStatus              %Status is returned in W1 register.
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     141     %Monitor call DeviceControl.
        JAN     ERROR   %Error if register A is negative.
        ...
ERROR,  ...             %Handle the error.
        ...
PAR,    DEVNO           %Logical device number.
        IOF             %Input or Output flag.
        PROG            %RealTime (program) description.
        CTRL            %Control flag.
        ...
DEVNO,  ...
IOF,    ...
PROG,   ...
CTRL,   ...
```

ND-100 and ND-500 **All users** **All users**

MAGTP — DEVICEFUNCTION — 144B

Performs various operations on floppy disks, magnetic tapes, cassette tapes, Versatec plotters, and ND-NET channels.

- The parameter values depends on the device.
- If the function code is in the range 5B to 24B, except 23B, the parameter buffer and the two device dependent parameters are dummies.
- If the function code is in the range 20B to 24B, except 23B, the hardware status is returned.

See also DeviceControl and @DEVICE-FUNCTION.

PARAMETERS

→ Function code. See the following pages.
→ Buffer used for data transfer to and from the device.
→ Logical device number. See appendix B.
→ First device dependant parameter. See the following pages.
→ Second device dependant parameter. See the following pages.
← Device dependant status code. See the following pages.

PASCAL

```
DevNo, Func, Param1, Param2 : INTEGER2;
Buff : ARRAY [0..63] OF BITMAP;
...
DeviceFunction(Func, Buff, DevNo, Param1, Param2);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DevNo  COMP.
01  Func  COMP.
01  Param1  COMP.
01  Param2  COMP.
01  Buff.
    02  array COMP OCCURS 1024 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "DeviceFunction" USING Func, Buff, DevNo, Param1, Param2.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DevNo, Func, Param1, Param2
      INTEGER  Buff(1024)
      ...
      Monitor_Call('DeviceFunction', Func, Buff(1), DevNo, Param1, Param2)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

144B **DEVICEFUNCTION** **MAGTP**

Continued from previous page...

PLANC

```
INTEGER : DevNo, Func, Param1, Param2
BYTE ARRAY : Buff(0:2047)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('DeviceFunction', Func, Buff(0), DevNo, Param1, Param2)
```

ASSEMBLY-500

```
DevNo : W  BLOCK  1
Func : W  BLOCK  1
Param1 : W  BLOCK  1
Param2 : W  BLOCK  1
Buff : W  BLOCK 1024
ErrCode : W  BLOCK  1
DeviceFunction : EQU 37B9 + 144B
        ...
        CALLG DeviceFunction, 5, Func, Buff, DevNo, Param1, Param2
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     144     %Monitor call DeviceFunction.
        STA     STAT    %Store status returned.
        ...
STAT,   0
PAR,    FUNC            %Function to be performed.
        BUFF            %Address of buffer used for data transfer.
        DEVNO           %Logical device number.
        PARA1           %Device dependent parameter.
        PARA2           %Device dependent parameter.
        ...
FUNC,   ...
BUFF,   0
*+120/                  %Make a buffer of 80 words.
DEVNO,  ...
PARA1,  ...
PARA2,  ...
```

ND-100 and ND-500 **All users** **All users**

DOPEN — DIRECTOPEN — 220B

Files must be opened before they can be accessed. For public users this monitor call is identical to OpenFile. User SYSTEM and user RT are given the same access rights as the owner of a file.

See also OpenFile and CloseFile.

PARAMETERS

← File number.
→ Access code. The legal values are shown below.
0: Sequential write.
1: Sequential read.
2: Random read or write.
3: Random read only.
4: Sequential read or write.
5: Sequential write append.
6: Random read or write common on contiguous files.
7: Random read common on contiguous files.
8: Random read or write on contiguous files. Direct transfer for ReadFromFile, WriteToFile and DeviceFunction in RT programs.
9: Random read, write append for WriteToFile.
→ File name.
→ File type. Do not include the colon. Default is 'SYMB'.
← Standard error code. See appendix A.

PASCAL

```
FileNumber, AccCode : INTEGER2;
FileName : PACKED ARRAY [0..63] OF CHAR;
FileType : PACKED ARRAY [0..3] OF CHAR;
...
DirectOpen(FileNumber, AccCode, FileName, FileType);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNumber  COMP.
01  AccCode  COMP.
01  FileName   PIC X(64).
01  FileType   PIC X(4).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "DirectOpen" USING FileNumber, AccCode, FileName, FileType.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNumber, AccCode
      CHARACTER  FileName*64, FileType*4
      ...
      Monitor_Call('DirectOpen', FileNumber, AccCode, FileName(1:64),
     C                  FileType(1:4))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

220B **DIRECTOPEN** **DOPEN**

Continued from previous page...

PLANC

```
INTEGER : FileNumber, AccCode
BYTES : FileName(0:63), FileType(0:3)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('DirectOpen', FileNumber, AccCode, FileName, FileType)
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
AccCode : W  BLOCK  1
FileName : STRINGDATA  'EXAMPLE'''
FileType : STRINGDATA  'SYMB'''
ErrCode : W  BLOCK  1
DirectOpen : EQU 37B9 + 220B
        ...
        CALLG DirectOpen, 4, FileNumber, AccCode, FileName, FileType
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (FILE    %Address of file name string.
        LDA     (TYPE    %Address of file type string.
        LDT     ACCES    %Access code.
        MON     220      %Monitor call DirectOpen.
        JMP     ERROR    %Error return from monitor call.
        STA     FILNO    %Normal return, store the filenumber returned.
        ...
ERROR,  ...              %Error number in register A.
        ...
FILNO,  0
ACCES,  ...
FILE,   'EXAMPLE'        %Open EXAMPLE:SYMB
TYPE,   'SYMB'           %
```

ND-100 and ND-500 **All users** **All users**

DESCF DISABLEESCAPE 71B

The ESCAPE key on the terminal normally terminates a program. This is called user break. This monitor call disables the escape function.

- The escape function is enabled again by EnableEscape.
- The escape function is enabled when you log out.
- When escape function is disabled, the escape character is treated as any other character.

See also EnableEscape, SetEscapeHandling, and @DISABLE-ESCAPE-FUNCTION.

PARAMETERS

→ The terminal's logical device number. This parameter is ignored for background programs. Your own terminal is always selected.

PASCAL

```
DeviceNumber : INTEGER2;
...
EscapeDisable(DeviceNumber);  [Note routine name.]
```

COBOL

```
01  DeviceNumber  COMP.
    ...
    MONITOR-CALL "DisableEscape" USING DeviceNumber.
```

FORTRAN

```
      INTEGER  DeviceNumber
      ...
      Monitor_Call('DisableEscape', DeviceNumber)
```

ND-100 and ND-500 | All users | All programs

71B DISABLEESCAPE DESCP

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber
...
Monitor_Call('DisableEscape', DeviceNumber)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
DisableEscape : EQU 37B9 + 71B
        ...
        CALLG DisableEscape, 1, DeviceNumber
```

MAC

```
        LDT     DEVNO    %Logical device number.
        MON     71       %Monitor call DisableEscape.
        ...
DEVNO,  ...
```

ND-100 and ND-500	All users	All users

DLOFU DISABLELOCAL 277B

You may log in on remote computers through the COSMOS data network. A key on the terminal returns you to your local computer. This monitor call disables the function of this key.

- You enable the key with EnableLocal.
- The key is not enabled when a program terminates.
- The COSMOS CONNECT-TO program tells you which key to use as the LOCAL key.

See also EnableLocal.

PARAMETERS

← Standard error code. See appendix A.

PASCAL

```
LocalDisable;    [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  ErrCode  COMP.
    ...
    MONITOR-CALL "DisableLocal".
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      Monitor_Call('DisableLocal')
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 All users All programs

277B **DISABLELOCAL** **DLOFU**

Continued from previous page...

PLANC

```
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('DisableLocal')
```

ASSEMBLY-500

Not available.

MAC

```
        MON     277     %Monitor call DisableLocal.
        JMP     ERROR   %Error return from monitor call.
        ...             %Normal return.
ERROR,  ...             %Error number in register A.
        ...
```

ND-100 **All users** **All users**

RTOFF — DISABLERTSTART — 137B

Disables start of RT programs. No RT program can be started before EnableRTStart is executed.

- RT programs in the time queue will not start. Other active RT programs are not affected.

See also EnableRTStart, and @RTOFF.

PARAMETERS

→ Address of the RT description. 0 means calling program.
← Standard error code. See appendix A.

PASCAL

```
RTProgram : INTEGER2;
...
DisableRTStart(RTProgram);
```

COBOL

```
01  RTProgram  COMP.
    ...
    MONITOR-CALL "DisableRTStart" USING RTProgram.
```

FORTRAN

```
      INTEGER  RTProgram
      ...
      Monitor_Call('DisableRTStart', RTProgram)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

137B **DISABLERTSTART** **RTOFF**

Continued from previous page...

PLANC

```
INTEGER : RTProgram
...
Monitor_Call('DisableRTStart', RTProgram)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
DisableRTStart : EQU 37B9 + 137B
        ...
        CALLG DisableRTStart, 1, RTProgram
```

MAC

```
        LDA     (PAR     %Load register A with address of parameter list.
        MON     137      %Monitor call DisableRTStart.
        ...
PAR,    RTPRO            %Address of RT description.
        ...
RTPRO,  ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

DISASS — DISASSEMBLE — 401B

Disassembles one machine instruction on the ND-500. Output is the instruction in ASSEMBLY-500 language. See the manual ND-500 ASSEMBLER REFERENCE MANUAL (ND-60.113).

- The returned string is truncated if it contains more characters than given in the last parameter.
- Causes an error return in ND-500 if called from a standard domain started directly from SINTRAN III.

PARAMETERS

→ Program address.
← The assembly instruction.
→ Maximum number of characters in the assembly instruction.

PASCAL

```
ProgPointer, MaxNoOfChar : LONGINT;
ReturnString : PACKED ARRAY [0..79] OF CHAR;
...
DisAssemble(ProgPointer, ReturnString, MaxNoOfChar);
```

COBOL

```
01  ProgPointer  COMP.
01  MaxNoOfChar  COMP.
01  ReturnString   PIC X(100).
    ...
    MONITOR-CALL "DisAssemble" USING ProgPointer, ReturnString, MaxNoOfChar.
```

FORTRAN

```
      INTEGER  ProgPointer, MaxChar
      CHARACTER  RetString*80
      ...
      Monitor_Call('DisAssemble', ProgPointer, RetString(1:80), MaxChar)
```

ND-500 | All users | All programs

401B	DISASSEMBLE	DISASS

Continued from previous page...

PLANC

```
INTEGER : ProgPointer, MaxNoOfChar
BYTES : ReturnString(0:79)
...
Monitor_Call('DisAssemble', ProgPointer, ReturnString, MaxNoOfChar)
```

ASSEMBLY-500

```
ProgPointer : W  BLOCK  1
MaxNoOfChar : W  BLOCK  1
ReturnString : STRINGDATA
DisAssemble : EQU 37B9 + 401B
        ...
        CALLG DisAssemble, 3, ProgPointer, ReturnString, MaxNoOfChar
```

MAC

Not available.

ND-500	All users	All users

DMAC Breakpoint	DMACBREAKPOINT	51B

Special monitor call used by the DMAC assembler. Not used in ordinary programs. The program executing this monitor call is put in the I/O wait state. A message is sent to the error device.

PARAMETERS

This monitor call has no parameters

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100	User RT and user SYSTEM	RT programs

51B	DMACBREAKPOINT	DMAC Breakpoint

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Not available.

MAC

```
MON     51      %Monitor call DMACBreakpoint.
```

ND-100	User RT and user SYSTEM	RT programs

UDMA DMAFUNCTION 333B

Performs various DMA functions. Most functions transfer data between your memory area and a DMA channel.

- The output part of the DMA channel must be reserved. See ReserveResource. Do not reserve the input part.
- RT programs may fix the segment which holds the memory buffer contiguously. See FixContigously. Then the data are transferred directly into the memory buffer.
- No Wait can only be used from ND-500 or if the segment is fixed contiguously.

See also DeviceFunction.

PARAMETERS

→ Logical device number of a DMA channel.
→ Function code. See the next page.
→ Memory address of data to send or receive for function code 0:3. Function codes 54:57 use programmed input/output devices. Then this parameter contains the logical device number of the device. The parameter is ignored for other function codes.
→ Input parameter. Function dependent. See the next page.
← Output parameter. Function dependent. See the next page. Some functions return a status value. Bits 0:7 and bit 15 are copied from the device status word. Bits 8:14 are user status lines. Bit 16 means that an DMA interrupt has ocurred. Bit 17 means timeout.
← Standard error code. See appendix A.

PASCAL

```
DeviceNo, FuncCode : INTEGER2; InPara, OutPara : LONGINT;
DataAddress : ARRAY [0..16] OF BITMAP;
...
DMAFunction(DeviceNo, FuncCode, DataAddress, InPara, OutPara);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNo  COMP. 01  FuncCode  COMP.
01  InPara  COMP PIC S9(10). 01  OutPara  COMP PIC S9(10).
01  DataAddress.
    02  array COMP OCCURS 256 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "DMAFunction" USING DeviceNo, FuncCode, DataAddress,
                 InPara, OutPara.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNo, FuncCode
      INTEGER*4  InPara, OutPara
      INTEGER  DataAddress(256)
      Monitor_Call('DMAFunction', DeviceNo, FuncCode,
     C                 DataAddress(1), InPara, OutPara)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

333B **DMAFUNCTION** **UDMA**

Continued from previous page...

PLANC

```
INTEGER : DeviceNo, FuncCode
INTEGER4 : InPar, OutPar
BYTE ARRAY : DataAdr(0:511)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('DMAFunction', DeviceNo, FuncCode, DataAdr(0), InPar, OutPar)
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
FuncCode : W  BLOCK  1
InPara : W  BLOCK  1      %Length must be an even number of bytes.
OutPara : W  BLOCK  1
Buffer : W  BLOCK 256
ErrCode : W  BLOCK  1
DMAFunction : EQU 37B9 + 333B
        ...
        CALLG DMAFunction, 5, FuncCode, Buffer, DeviceNo, InPara, OutPara
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     LDN     %Load register T with the logical device number.
        LDA     (PAR    %Load register A with address of parameter list.
        MON     333     %Monitor call DMAFunction.
        STA     STAT
        ...
STAT,   0
PAR,    FUNC            %Parameter list.
        ADR
        IPAR
        OPAR
FUNC,   ...
ADR,    ...             %Address in your memory area.
IPAR,   0;0             %A 32-bit input parameter.
OPAR,   0;0             %A 32-bit output parameter.
```

Function code	Input parameter	Output parameter
1 Receive DMA data	length in bytes	actual length
2 Send DMA data	length in bytes	not used
3 Receive DMA data with No Wait	length in bytes	not used
3 Send DMA data with No Wait	length in bytes	not used
7 Test mode	not used	status
20 Read DMA status	not used	status
21 Clear DMA device	not used	status
24 Read last status	not used	not used
54 Programmed input without interrupt		not used
55 Programmed output without interrupt		not used
56 Programmed input with interrupt		not used
57 Programmed input with interrupt		not used
62 Wait for DMA finished interrupt	1 to 4	status
64 Enable DMA interrupt	not used	not used
65 Disable DMA interrupt	not used	not used
70 Write user control lines		not used

ND-100 and ND-500	All users	All users

RESCP EnableEscape 72B

The ESCAPE key on the terminal normally terminates a program. This is called user break. You can disable this key with DisableEscape. To enable it again you should use EnableEscape.

- The escape function is enabled when you log out.

See also DisableEscape, SetEscapeHandling, and @ENABLE-ESCAPE-FUNCTION.

PARAMETERS

→ The terminal's logical device number. This parameter is ignored for background programs. Your own terminal is always selected.

PASCAL

```
DeviceNumber : INTEGER2;
...
EscapeEnable(DeviceNumber);     [Note routine name.]
```

COBOL

```
01  DeviceNumber  COMP.
    ...
    MONITOR-CALL "EnableEscape" USING DeviceNumber.
```

FORTRAN

```
      INTEGER  DeviceNumber
      ...
      Monitor_Call('EnableEscape', DeviceNumber)
```

ND-100 and ND-500 | All users | All programs

72B **ENABLEESCAPE** **RESCP**

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber
...
Monitor_Call('EnableEscape', DeviceNumber)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
EnableEscape : EQU 37B9 + 72B
        ...
        CALLG EnableEscape, 1, DeviceNumber
        IF K GO Error
        ...
Error,
```

MAC

```
        LDT     DEVNO    %Logical device number.
        MON     72       %Monitor call EnableEscape.
        ...
DEVNO,  ...
```

ND-100 and ND-500 **All users** **All users**

ELOPU **ENABLELOCAL** 2765

You may log in on remote computers through the COSMOS data network. A key on the terminal returns you to your local computer. This local function can be disabled. You enable it again with EnableLocal.

- You disable the key with DisableLocal.
- The key is disabled when a program terminates.
- The COSMOS CONNECT-TO program tells you which key to use as the LOCAL key.

See also DisableLocal.

PARAMETERS

← Standard error code. See appendix A.

PASCAL

```
LocalEnable;     [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  ErrCode  COMP.
    ...
    MONITOR-CALL "EnableLocal".
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      Monitor_Call('EnableLocal')
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 All users All programs

276B **ENABLELOCAL** **ELOPU**

Continued from previous page...

PLANC

```
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('EnableLocal')
```

ASSEMBLY-500

Not available.

MAC

```
        MON     276     %Monitor call EnableLocal.
        JMP     ERROR   %Error return from monitor call.
        ...             %Normal return.
ERROR,  ...             %Error number in register A.
        ...
```

ND-100 **All users** **All users**

RTON — EnableRTStart — 136B

RT programs cannot be started after DisableRTStart is executed. You have to use EnableRTStart.

See also DisableRTStart, and @RTON.

PARAMETERS

→ Address of the RT description. 0 means calling program.

PASCAL

```
RTProgram : INTEGER2;
...
EnableRTStart(RTProgram);
```

COBOL

```
01  RTProgram  COMP.
    ...
    MONITOR-CALL "EnableRTStart" USING RTProgram.
```

FORTRAN

```
      INTEGER  RTProgram
      ...
      Monitor_Call('EnableRTStart', RTProgram)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

136B **ENABLERTSTART** **RTOM**

Continued from previous page...

PLANC

```
INTEGER : RTProgram
...
Monitor_Call('EnableRTStart', RTProgram)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
EnableRTStart : EQU 37B9 + 136B
        ...
        CALLG EnableRTStart, 1, RTProgram
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     136     %Monitor call EnableRTStart.
        ...
PAR,    RTPRO           %Address of RT description.
        ...
RTPRO,  ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

QERMS ERRORMESSAGE 65B

Displays a file system error message. Appendix A shows the messages connected to each error code. The error code is input. The program terminates.

- The error message is displayed on the terminal. RT programs write it to the error device. The error device is normally the console.
- Do not input error code 0.

See also GetErrorMessage and WarningMessage. WarningMessage writes out the error message without terminating the program.

PARAMETERS

→ Error code of the message to be printed. Use octal numbers.

PASCAL

```
ErrNumber : INTEGER2;
...
ErrorMessage(ErrNumber);
```

COBOL

```
01  ErrNumber  COMP.
    ...
    MONITOR-CALL "ErrorMessage" USING ErrNumber.
```

FORTRAN

```
      INTEGER  ErrNumber
      ...
      Monitor_Call('ErrorMessage', ErrNumber)
```

ND-100 and ND-500 | All users | All programs

65B **ERRORMESSAGE** QRRMS

Continued from previous page...

PLANC

```
INTEGER : ErrNumber
...
Monitor_Call('ErrorMessage', ErrNumber)
```

ASSEMBLY-500

```
ErrNumber : W  BLOCK  1
ErrorMessage : EQU 37B9 + 65B
        ...
        CALLG ErrorMessage, 1, ErrNumber
        IF K GO Error
        ...
Error,  ...
```

MAC

```
        LDA     ERRNO    %Error number of message to be printed.
        MON     65       %Monitor call ErrorMessage.
        ...
ERRNO,  ...
```

ND-100 and ND-500 All users All users

MACRO	ERRORRETURN	400B

Terminates the program and set an error code. The error code can be tested by the commands IF-ERROR-MACRO-STOP and IF-ERROR-FULL-STOP commands in the ND-500-MONITOR. See the ND-500 LOADER/MONITOR (ND-60.136) for details about macros.

PARAMETERS

This monitor call has no parameters.

PASCAL

```
ErrorReturn;
```

COBOL

```
    MONITOR-CALL "ErrorReturn".
```

FORTRAN

```
      Monitor_Call('ErrorReturn')
```

ND-500	All users	All programs

400B	ERRORRETURN	MACROS

Continued from previous page...

PLANC

```
Monitor_Call('ErrorReturn')
```

ASSEMBLY-500

```
ErrorReturn : EQU 37B9 + 400B
        ...
        CALLG ErrorReturn, 0
```

MAC

Not available.

ND-500	All users	All users

DSET — EXACTDELAYSTART — 126B

Sets an RT program to start after a given period. It is then moved from the time queue to the execution queue. The period is specified in basic time units. A basic time unit is 1/50th of a second. The period may be from 1 to 4294967647 basic time units.

- The program may already be in the time queue. It is then reinserted according to the new specifications.
- A period less than or equal to 0 transfers the RT program to the execution queue the next time the basic time unit counter is incremented.
- SetClock, AdjustClock and @CLADJ do not affect the interval.
- StopRTProgram removes the RT program from the time queue.

See also DelayStart and StartupTime. DelayStart allows you to specify the period in seconds, minutes, and hours.

PARAMETERS

→ Address of RT description. Use 0 for the calling program.
→ Number of basic time units before start.

PASCAL

```
RTProgram, BasicTimeUnits : INTEGER2;
...
ExactDelayStart(RTProgram, BasicTimeUnits);
```

COBOL

```
01  RTProgram  COMP.
01  BasicTimeUnits  COMP.
    ...
    MONITOR-CALL "ExactDelayStart" USING RTProgram, BasicTimeUnits.
```

FORTRAN

```
      INTEGER  RTProgram, BasicTimeUnits
      ...
      Monitor_Call('ExactDelayStart', RTProgram, BasicTimeUnits)
```

ND-100 and ND-500 | **User RT and user SYSTEM** | **RT programs**

126B **EXACTDELAYSTART** **DSET**

Continued from previous page...

PLANC

```
INTEGER : RTProgram, BasicTimeUnits
...
Monitor_Call('ExactDelayStart', RTProgram, BasicTimeUnits)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
BasicTimeUnits : W  BLOCK  1
ExactDelayStart : EQU 37B9 + 126B
        ...
        CALLG ExactDelayStart, 2, RTProgram, BasicTimeUnits
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     126     %Monitor call ExactDelayStart.
        ...
PAR,    RTPRO           %Address of RT description.
        TIME            %Number of basic time units the program
        ...             % is to stay in the time queue.
RTPRO,  ...
TIME,   ...             %A double word.
        ...             %
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

DINTV — EXACTINTERVAL — 130B

Prepares an RT program for periodic execution. The interval between the executions may be from 1 to 4294967647 basic time units. A basic time unit is 1/50th of a second.

- The RT program is not started. Use, for example, StartRTProgram or @RT to start it.
- StopRTProgram, Disconnect, @ABORT or @DSCNT cancel this monitor call.
- One execution may be unfinished when it is time for the next execution. In this case, the program's restart flag is set. If the delay becomes as long as two intervals, one execution is lost.
- The interval replaces any earlier specified intervals.
- AdjustClock and @CLADJ do not affect the interval.

See also StartupInterval and @DINTV. StartupInterval allows you to specify intervals in seconds, minutes or hours.

PARAMETERS

→ Address of an RT description. 0 means calling program. GetRtAddress gives RT description addresses.
→ Period between executions in basic time units.

PASCAL

```
RTProgram, BasicTimeUnits : INTEGER2;
...
ExactInterval(RTProgram, BasicTimeUnits);
```

COBOL

```
01  RTProgram  COMP.
01  BasicTimeUnits  COMP.
    ...
    MONITOR-CALL "ExactInterval" USING RTProgram, BasicTimeUnits.
```

FORTRAN

```
      INTEGER  RTProgram, BasicTimeUnits
      ...
      Monitor_Call('ExactInterval', RTProgram, BasicTimeUnits)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

130B **EXACTINTERVAL** **DIMTV**

Continued from previous page...

PLANC

```
INTEGER : RTProgram, BasicTimeUnits
...
Monitor_Call('ExactInterval', RTProgram, BasicTimeUnits)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
BasicTimeUnits : W  BLOCK  1
ExactInterval : EQU 37B9 + 130B
        ...
        CALLG ExactInterval, 2, RTProgram, BasicTimeUnits
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     130     %Monitor call ExactInterval.
        ...
PAR,    RTPRO           %Address of RT description.
        TIME            %Time of interval between each execution.
        ...
RTPRO,  ...
TIME,   ...             %A double word giving number of basic time units.
        ...             %
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

DABST — EXACTSTARTUP — 127B

Starts an RT program at a specific time. The time is given in basic time units. A basic time unit is 1/50th of a second. The RT program is moved from the time queue to the execution queue at the specified time.

- It may already be later than the time specified. The RT program is then scheduled for the next day.
- The RT program may already be in the time queue. It is then reinserted according to the new time specifications.
- AdjustClock and @CLADJ affect the startup. The RT program starts according to the new time.
- Use GetBasicTime to read the internal time in basic time units.

See also StartupTime. StartupTime allows you to specify the time in hours, minutes or seconds.

PARAMETERS

→ Address of an RT description. 0 means calling program. GetRtAddress gives RT description addresses.
→ StartupTime in basic time units.

PASCAL

```
RTProgram, BasicTimeUnits : INTEGER2;
...
ExactStartup(RTProgram, BasicTimeUnits);
```

COBOL

```
01  RTProgram  COMP.
01  BasicTimeUnits  COMP.
    ...
    MONITOR-CALL "ExactStartup" USING RTProgram, BasicTimeUnits.
```

FORTRAN

```
      INTEGER  RTProgram, BasicTimeUnits
      ...
      Monitor_Call('ExactStartup', RTProgram, BasicTimeUnits)
```

ND-100 and ND-500 | **User RT and user SYSTEM** | **RT programs**

127B	EXACTSTARTUP	DABST

Continued from previous page...

PLANC

```
INTEGER : RTProgram, BasicTimeUnits
...
Monitor_Call('ExactStartup', RTProgram, BasicTimeUnits)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
BasicTimeUnits : W  BLOCK  1
ExactStartup : EQU 37B9 + 127B
        ...
        CALLG ExactStartup, 2, RTProgram, BasicTimeUnits
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     127     %Monitor call ExactStartup.
        ...
PAR,    RTPRO           %Address of RT description.
        TIME            %Time when the program is to be executed.
        ...
RTPRO,  ...
TIME,   ...             %A double word giving time in basic time units.,
        ...             %
```

ND-100 and ND-500	User RT and user SYSTEM	RT programs

UBCOM	EXECUTECOMMAND	317B

Executes a SINTRAN III command. Specify the command name and the parameters as a text string.

- An error message is output if an error occurs. The program does not terminate.
- Some commands may destroy your program. Commands which affect your program's memory area should be used with care.
- Some commands have output, eg., @LIST-FILES. This is displayed on the terminal.
- Use SuspendProgram to wait a second between two ExecuteCommands which depend on each other, eg., CreateFile and OpenFile.

See also CallCommand and SetCommandBuffer. CallCommand terminates the program if an error occurs.

PARAMETERS

→ Command with parameters.

PASCAL

```
Command : PACKED ARRAY [0..34] OF CHAR;
...
ExecuteCommand(Command);
```

COBOL

```
01  Command   PIC X(35).
    ...
    MONITOR-CALL "ExecuteCommand" USING Command.
```

FORTRAN

```
      CHARACTER  Command*35
      ...
      Monitor_Call('ExecuteCommand', Command(1:35))
```

ND-100 and ND-500	All users	Background programs

317B **EXECUTECOMMAND** **UBCOM**

Continued from previous page...

PLANC

```
BYTES : Command(0:35)
...
Monitor_Call('ExecuteCommand', Command)
```

ASSEMBLY-500

```
Command : STRINGARRAY 35
ExecuteCommand : EQU 37B9 + 317B
        ...
        CALLG ExecuteCommand, 1, Command
```

MAC

```
        LDA     (CMND    %Address of string with SINTRAN command.
        MON     317      %Monitor call ExecuteCommand.
        ...
CMND,   'CLOSE-FILE 102' %Execute CLOSE-FILE 102.
```

ND-100 and ND-500 **All users** **Background programs**

RSIO EXECUTIONINFO 143B

Gets information about the execution of a program. You are told whether the program executes interactively, as a batch or mode job, or as an RT program. The monitor call returns some additional information for non-RT programs, consisting of the command input file, the command output file, and the directory index and user index of the program's owner.

- Your terminal number is returned as the command input and output files in interactive programs.

PARAMETERS

← Execution. 0 means interactive program. 1 means batch job. 2 means mode job. 3 means RT program.
← Logical device number for command input. This is your terminal number for interactive programs. Batch and mode jobs return the file number of the command input file. Not used for RT programs.
← Logical device number for command output. This is your terminal number for interactive programs. Batch and mode jobs return the file number of the command output file. Not used for RT programs.
← Directory and user index of the program's owner. The first byte contains the directory index. The second byte contains the user index. Not used for RT programs.
← Standard error code. See appendix A.

PASCAL

```
ExecutionMode, InputDev, OutputDev, UserIndex : INTEGER2;
...
ExecutionInfo(ExecutionMode, InputDev, OutputDev, UserIndex);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  ExecutionMode  COMP.
01  InputDev  COMP.
01  OutputDev  COMP.
01  UserIndex  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "ExecutionInfo" USING ExecutionMode, InputDev,
                 OutputDev, UserIndex.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ....
```

FORTRAN

```
      INTEGER  ExecutionMode, InputDev, OutputDev, UserIndex
      ...
      Monitor_Call('ExecutionInfo', ExecutionMode, InputDev,
     C                  OutputDev, UserIndex)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

143B	EXECUTIONINFO	RSIO

Continued from previous page...

PLANC

```
INTEGER : ExecutMode, InputDev, OutputDev, UserIndex
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('ExecutionInfo', ExecutMode, InputDev, OutputDev, UserIndex)
```

ASSEMBLY-500

```
ExecutMode : W  BLOCK  1
InputDev : W  BLOCK  1
OutputDev : W  BLOCK  1
UserIndex : W  BLOCK  1
ExecutionInfo : EQU 37B9 + 143B
        ...
        CALLG ExecutionInfo, 4, ExecutMode, InputDev, OutputDev, UserIndex
```

MAC

```
        MON       143       %Monitor call ExecutionInfo.
        STA       EXMOD     %Store execution mode.
        STT       IFILE     %Store file number of command input file.
        COPY      SD  DA
        STA       OFILE     %Store file number of command output file.
        STX       INDEX     %Store directory and user indexes.
        ...
EXMOD,  0
IFILE,  0
OFILE,  0
INDEX,  0
```

ND-100 and ND-500	All users	All users

LEAVE — ExitFromProgram — OB

Terminates the program. Returns to SINTRAN III. Batch jobs continues with the next command.

- Background programs close all files not set permanently open. RT programs do not close any files.
- RT programs release all reserved devices.

See also ExitRTProgram.

PARAMETERS

This monitor call has no parameters.

PASCAL

```
StopProgram;      [Note routine name.]
```

COBOL

```
    MONITOR-CALL "ExitFromProgram".
```

FORTRAN

```
      Monitor_Call('ExitFromProgram')
```

ND-100 and ND-500	All users	All programs

OB	EXITFROMPROGRAM	LEAVE

Continued from previous page...

PLANC

```
Monitor_Call('ExitFromProgram')
```

ASSEMBLY-500

```
ExitFromProgram : EQU 37B9 + OB
        ...
        CALLG ExitFromProgram, 0
        IF K  GO Error              %Possible if wrong number of parameters.
```

MAC

```
        MON       0         %Monitor call ExitFromProgram.
```

ND-100 and ND-500	All users	All users

MEXIT — EXITFROMSEGMENT — 133B

Exchanges one or both current segments. Commonly used to return after the monitor call JumpToSegment.

- The L register must contain the return address. This makes the monitor call complicated too use in high level languages.

See also JumpToSegment and ChangeSegment.

PARAMETERS

→ Segment numbers. The most significant byte of this 16-bit integer contains the first segment number. The next byte contains the second segment number. Use 377B for the segment you do not want to change.

PASCAL

```
SegmentNumber : INTEGER2;
...
ExitFromSegment(SegmentNumber);
```

COBOL

Not available.

FORTRAN

Not avilable. ...'

ND-100 | User RT and user SYSTEM | RT programs

133B EXITFROMSEGMENT MEXIT

Continued from previous page...

PLANC

Not available....

ASSEMBLY-500

Not available.

MAC

```
        ...             %See JumpToSegment.
        ...             %This code is on segment 30B.
SUBR,   STT     SEGNO   %Entry point after JumpToSegment.
        COPY    SL  DT  %Save T and L registers.
        STT     RETUR   %L register contains return address.
        ...             %Any processing on the segment.
        ...
        LDT     RETUR   %Restore return address in L register.
        COPY    ST  DL
        LDT     SEGNO   %Restore calling segment number.
        MON     133     %Monitor call ExitFromSegment.
SEGNO,  ...             %Segment number.
RETUR,  ...             %Return address.
```

ND-100 User RT and user SYSTEM RT programs

RTEXT — EXITRTPROGRAM — 134B

Terminates the calling RT or background program. Releases all reserved resources. The monitor call has the same effect as exit for interactive background programs.

- Batch jobs are aborted.

See also ExitFromProgram and StopRTProgram.

PARAMETERS

This monitor call has no parameters.

PASCAL

```
ExitRTProgram;
```

COBOL

```
    MONITOR-CALL "ExitRTProgram".
```

FORTRAN

```
      Monitor_Call('ExitRTProgram')
```

ND-100 and ND-500 | All users | All programs

134B **EXITRTPROGRAM** **RTEXT**

Continued from previous page...

PLANC

```
Monitor_Call('ExitRTProgram')
```

ASSEMBLY-500

```
ExitRTProgram : EQU 37B9 + 134B
        ...
        CALLG ExitRTProgram, 0
```

MAC

```
        MON       134       %Monitor call ExitRTProgram.
```

ND-100 and ND-500 **All users** **All users**

EXPFI — EXPANDFILE — 231B

Expands the file size. You use this monitor call to increase the size of contiguous and allocated files. The space following the file on the disk must be free.

- Indexed files created with 0 pages may be expanded.
- Public users must have directory access to the file. User RT and SYSTEM can expand any file.

See also CreateFile and @EXPAND-FILE.

PARAMETERS

→ File name. It may be abbreviated, but this slows down execution.
→ Number of additional pages.
← Standard error code. See appendix A.

PASCAL

```
NoOfPages : LONGINT;
FileName : PACKED ARRAY [0..63] OF CHAR;
...
ExpandFile(FileName, NoOfPages);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  NoOfPages  COMP PIC S9(10).
01  FileName   PIC X(64).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "ExpandFile" USING FileName, NoOfPages.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER*4  NoOfPages
      CHARACTER  FileName*64
      ...
      Monitor_Call('ExpandFile', FileName(1:64), NoOfPages)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

231B **EXPANDFILE** **EXPFI**

Continued from previous page...

PLANC

```
INTEGER4 : NoOfPages
BYTES : FileName(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('ExpandFile', FileName, NoOfPages)
```

ASSEMBLY-500

```
NoOfPages : W  BLOCK  1
FileName : STRINGARRAY 64
ErrCode : W  BLOCK  1
ExpandFile : EQU 37B9 + 231B
        ...
        CALLG ExpandFile, 2, FileName, NoOfPages
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (FILE    %Address of file name string.
        LDT     (PAGES   %Address of  double word with number of pages.
        MON     231      %Monitor call ExpandFile.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILE,   'EXAMPLE:SYMB'   %File name.
PAGES,  ...              %A double word.
        ...              %
```

ND-100 and ND-500 **All users** **All users**

PSCNT **FILEASSEGMENT** 412B

Connects a file as a segment to your domain. You can then access the file as a logical segment. This reduces the access time.

- The file must be open. The access must be specified in the OpenFile call.
- The file is disconnected when it is closed.
- A file may be connected to several processes simultaneously. It is your responsibility to synchronize simultaneous accesses.

See also FileNotAsSegment which disconnects the file.

PARAMETERS

→ File number. See OpenFile.
→ Logical segment number in the domain. The segment number must be free. Use 0 to select the first free segment. The last parameter returns the logical segment number.
→ Type. Use 0 if the file contains initial data. 1 means uninitialized, empty file. 2 means primarily sequential access. Use 3 for a combination of 1 and 2. Type 2 reduces the access time if access is sequential.
← Logical segment number selected if you give 0 in the second parameter.

PASCAL

```
FileNo, LogSegmentNo, Type, SegmentNo : LONGINT;
...
FileAsSegment(FileNo, LogSegmentNo, Type, SegmentNo);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNo  COMP.
01  LogSegmentNo  COMP.
01  Type  COMP.
01  SegNo  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "FileAsSegment" USING FileNo, LogSegmentNo, Type, SegNo.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNo, LogSegmentNo, Type, SegmentNo
      ...
      Monitor_Call('FileAsSegment', FileNo, LogSegmentNo, Type, SegmentNo)
      IF (ErrCode .NE. 0) THEN ...
```

ND-500 All users All programs

412B **FILEASSEGMENT** FSCNT

Continued from previous page...

PLANC

```
INTEGER : FileNo, LogSegmentNo, Type, SegmentNo
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('FileAsSegment', FileNo, LogSegmentNo, Type, SegmentNo)
```

ASSEMBLY-500

```
FileNo : W  BLOCK  1
LogSegmentNo : W  BLOCK  1
Type : W  BLOCK  1
SegmentNo : W  BLOCK  1
ErrCode : W  BLOCK  1
FileAsSegment : EQU 37B9 + 412B
        ...
        CALLG FileAsSegment, 4, FileNo, LogSegmentNo, Type, SegmentNo
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

Not available.

ND-500 All users All users

FSDCNT FILENOTASSEGMENT 413B

Disconnects a file as a segment in your domain. FileAsSegment allows files to be accessed as segments. This monitor call disconnects the file.

- The file is not closed.
- The file is automatically disconnected by CloseFile.

See also FileAsSegment.

PARAMETERS

→ File number. See OpenFile.

PASCAL

```
FileNumber : LONGINT;
...
FileNotAsSegment(FileNumber);
```

COBOL

```
01  FileNumber  COMP.
    ...
    MONITOR-CALL "FileNotAsSegment" USING FileNumber.
```

FORTRAN

```
      INTEGER  FileNumber
      ...
      Monitor_Call('FileNotAsSegment', FileNumber)
```

ND-500 All users All programs

413B	FILENOTASSEGMENT	PSDCMT

Continued from previous page...

PLANC

```
INTEGER : FileNumber
...
Monitor_Call('FileNotAsSegment', FileNumber)
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
FileNotAsSegment : EQU 37B9 + 413B
        ...
        CALLG FileNotAsSegment, 1, FileNumber
```

MAC

Not available.

ND-500	All users	All users

PSHTY FILESYSTEMFUNCTION 327B

Makes sure that an uncontrolled system stop does not leave the file system inconsistent. The file index block of an open file is written back to the disk.

- This monitor call is particularly useful for SIBAS and ISAM applications.

PARAMETERS

→ Function code.
1 means write back to disk the index block of an open file.
2 means return the block size of an opened file.
3 means get full file name of an opened file.
→ File number. See OpenFile.
→ Parameter 2. Block size if function code 2.
→ Parameter 3. File name if function code 3.
← Standard error code. See appendix A.

PASCAL

```
FuncCode, FileNo : INTEGER2;
...
FileSystemFunction(FuncCode, FileNo);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FuncCode  COMP.
01  FileNo  COMP.
01  Param2  COMP.
01  Param3  X PIC(64).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "FileSystemFunction" USING FuncCode, FileNo,Param3,Param2.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FuncCode, FileNo, Param2
      CHARACTER  Param3*64
      ...
      Monitor_Call('FileSystemFunction', FuncCode, FileNo, Param3, Param2)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

327B **FILESYSTEMFUNCTION** PSMTY

Continued from previous page...

PLANC

```
INTEGER : FuncCode,FileNo,Param2
BYTES : Param3(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('FileSystemFunction', FuncCode, FileNo, Param3, Param2)
```

ASSEMBLY-500

```
Func1Code : W  DATA  1    %Various function codes.
Func2Code : W  DATA  2
Func3Code : W  DATA  3
FileNo : W  BLOCK  1
BlockSize : W  BLOCK  1
FileName : STRINGARRAY 200B
ErrCode : W  BLOCK  1
FileSystemFunction : 37B9 + 327B
        ...
        CALLG FileSystemFunction, 2, Func1Code, FileNo
        IF K GO Error
        ...
        CALLG FileSystemFunction, 3, Func2Code, FileNo, BlockSize
        IF K GO Error
        ...
        CALLG FileSystemFunction, 3, Func3Code, FileNo, FileName
        IF K GO Error
        ...
        ...
Error,  W1 =: ErrCode
```

MAC

```
        LDT     FUNC      %Load register T with the function code.
        LDA     FILNO     %Load register A with the file number.
        LDX     (BUFF     %Load register A with the file number.
        MON 327           %Monitor call FileSystemFunction.
        JMP     ERROR     %Error return.
        STA     PAR2      %Returned parameter 2 if func = 2.
        ...
ERROR,  ...
        ...
FUNC,   1                 %Only function 1 is available on ND-100.
FILNO,  ...
BUFF,   0
*+26/                     %Buffer of 26 words.
PAR2,   0                 %Block size.
```

ND-100 and ND-500 All users All users

FIXC FixContiguous 160B

Places a segment in physical memory. Its pages will no longer be swapped to the disk. The segment is placed in a contiguous area of physical memory. This function is useful for time critical operations.

- The segment must be created with the RT LOADER as a non-demand segment.
- Use UnFixSegment or @UNFIX to allow the RT LOADER to clear the segment.
- Your program normally terminates if you refer to a nonexisting segment or a demand segment. An error message is output on the error device. See parameters 1 and 3 to get a status value instead.
- Only a limited number of pages can be fixed. This limit is defined when SINTRAN III is generated. You may change it with the command CHANGE-VARIABLE in the SINTRAN-SERVICE-PROGRAM.

See also UnFixSegment, FixScattered, FixInMemory, MemoryAllocation, FixIOArea, and @FIXC.

PARAMETERS

→ Segment number to be fixed. Set bit 15 to 1 if you want a return status.
→ First physical page number to be used.
← Return status if bit 15 of the segment number is set. Dummy on the ND-500. 0 means OK. -1 means space not available. -2 means illegal segment. -4 means attempt to fix demand segment. -5 means attempt to fix too many pages. -6 means segment already fixed at another address.

PASCAL

```
SegmentNo, PageNumber, Stat : INTEGER2;
...
FixContiguous(SegmentNo, PageNumber, Stat);
```

COBOL

```
01  SegmentNo  COMP.
01  PageNumber  COMP.
01  Stat  COMP.
    ...
    MONITOR-CALL "FixContiguous" USING SegmentNo, PageNumber, Stat.
```

FORTRAN

```
      INTEGER  SegmentNo, PageNumber, Stat
      ...
      Monitor_Call('FixContiguous', SegmentNo, PageNumber, Stat)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

160B **FIXCONTIGUOUS** **FIXC**

Continued from previous page...

PLANC

```
INTEGER : SegmentNo, PageNumber, Stat
...
Monitor_Call('FixContiguous', SegmentNo, PageNumber, Stat)
```

ASSEMBLY-500

```
SegmentNo : W  BLOCK  1
PageNumber : W  BLOCK  1
FixContiguous : EQU 37B9 + 160B
        ...
        CALLG FixContiguous, 2, SegmentNo, PageNumber
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     160     %Monitor call FixContiguous.
        STA     STAT    %Returned status.
        ...
STAT,   0
PAR,    SEGNO           %Segment number to be fixed.
        PAGE            %First physical page number.
        ...
SEGNO,  ...
PAGE,   ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

FIXMEM — FIXINMEMORY — 410B

Fixes a logical segment of your domain in physical memory. You can fix the whole or part of the segment. This speeds up access to the segment. It is also useful in segment sharing.

- Only data segments may be fixed.

See also MemoryUnFix.

PARAMETERS

→ Type. 0 means fix the pages scattered. 1 means fix the pages contiguously and return the start address. 2 means fix at the given memory address.
→ Start address in your domain. It should be a 32 bit address including the segment number.
→ Length in bytes. Use -1 to fix the remaining part of the segment.
→ Physical memory address in the ND-100 if type 1 or 2 is used. The address is the start of a physical page.

PASCAL

```
FixType, FirstAddr, Length, ND100Addr : LONGINT;
...
FixInMemory(FixType, FirstAddr, Length, ND100Addr);
```

COBOL

```
01  FixType  COMP.
01  FirstAddr  COMP.
01  Length  COMP.
01  ND100Addr  COMP..
    ...
    MONITOR-CALL "FixInMemory" USING FixType, FirstAddr, Length, ND100Addr.
```

FORTRAN

```
      INTEGER  FixType, FirstAddr, Length, ND100Addr
      ...
      Monitor_Call('FixInMemory', FixType, FirstAddr, Length, ND100Addr)
```

ND-500 | User RT and user SYSTEM | RT programs

410B **FixInMemory** **FIXMEM**

Continued from previous page...

PLANC

```
INTEGER : FixType, FirstAddr, Length, ND100Addr
...
Monitor_Call('FixInMemory', FixType, FirstAddr, Length, ND100Addr)
```

ASSEMBLY-500

```
FixType : W  BLOCK  1
FirstAddr : W  BLOCK  1
Length : W  BLOCK  1
ND100Addr : W  BLOCK  1
FixInMemory : EQU 37B9 + 410B
        ...
        CALLG FixInMemory, 4, FixType, FirstAddr, Length, ND100Addr
        IF K GO Error
        ...
Error,  ...
```

MAC

Not available.

ND-500 **User RT and user SYSTEM** **RT programs**

IOFIX — FixIOArea — 404B

Fixes an address area in a domain in physical memory. The memory area can be used for later input and output monitor calls, eg., ReadFromFile or WriteTofile..

- Use MemoryUnFix to release the pages.
- The pages are released when the domain terminates.

See also FixScattered, FixContiguous, FixInMemory, and MemoryAllocation.

PARAMETERS

→ Start address in the domain.
→ Number of bytes to fix.

PASCAL

```
FirstAddress, SizeOfArea : LONGINT;
...
FixIOArea(FirstAddress, SizeOfArea);
```

COBOL

```
01  FirstAddress  COMP.
01  SizeOfArea  COMP.
    ...
    MONITOR-CALL "FixIOArea" USING FirstAddress, SizeOfArea.
```

FORTRAN

```
      INTEGER  FirstAddress, SizeOfArea
      ...
      Monitor_Call('FixIOArea', FirstAddress, SizeOfArea)
```

ND-500 | **All users** | **All programs**

404B	FixIOArea	IOFIX

Continued from previous page...

PLANC

```
INTEGER : FirstAddress, SizeOfArea
...
Monitor_Call('FixIOArea', FirstAddress, SizeOfArea)
```

ASSEMBLY-500

```
Buffer : W  BLOCK  1000B
SizeOfArea : W  BLOCK  1
FixIOArea : EQU 37B9 + 404B
        ...
        CALLG FixIOArea, 2, Buffer, SizeOfArea
```

MAC

Not available.

ND-500	All users	All users

FIX **FIXSCATTERED** 115B

Place a segment in physical memory. Its pages will no longer be swapped to the disk. The segment must be non demand. Its pages will be scattered in physical memory. You may, for example, use this function for time critical operations or for allocating DMA buffers.

- Use UnFixSegment or @UNFIX to allow the RT LOADER to clear the segment.
- Your program terminates if you refer to a non-existing segment or a demand segment. An error message is output on the error device.
- Only a limited number of pages can be fixed. This limit is defined when SINTRAN III is generated. You may change it with the command CHANGE-VARIABLE in the SINTRAN-SERVICE-PROGRAM.

See also UnFixSegment, FixContiguous, FixInMemory, MemoryAllocation, FixIOArea, and @FIXC.

PARAMETERS

→ Segment number to be fixed in memory.

PASCAL

```
SegmentNumber : INTEGER2;
...
FixScattered(SegmentNumber);
```

COBOL

```
01  SegmentNumber  COMP.
    ...
    MONITOR-CALL "FixScattered" USING SegmentNumber.
```

FORTRAN

```
      INTEGER  SegmentNumber
      ...
      Monitor_Call('FixScattered', SegmentNumber)
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

115B **FIXSCATTERED** **FIX**

Continued from previous page...

PLANC

```
INTEGER : SegmentNumber
...
Monitor_Call('FixScattered', SegmentNumber)
```

ASSEMBLY-500

```
SegmentNumber : W  BLOCK  1
FixScattered : EQU 37B9 + 115B
        ...
        CALLG FixScattered, 1, SegmentNumber
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     115     %Monitor call FixScattered
        ...
PAR,    SEGNO           %Segment number to be fixed in memory.
        ...
SEGNO,  ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

PRLS — FORCERELEASE — 125B

Releases a device reserved by an RT program other than that which is calling. You can then reserve the device for your own RT program. Some devices, such as terminals, have both an input and output part. You can only release one part with each ForceRelease.

- Use ReservationInfo to get the RT description address of the reserving RT program. You may then give the device back with ForceReserve.
- Programs in a waiting queue may reserve the device between your ForceRelease and ReserveResource calls if they have higher priority then your program.
- The SINTRAN III REAL TIME GUIDE (ND-60.133) describes this in more detail.

See also ForceReserve, ReleaseResource, and @PRLS.

PARAMETERS

→ Logical device number. See appendix B.
→ Input or output flag. Use 0 for the input part and 1 for the output part.

PASCAL

```
DeviceNumber, IOflag : INTEGER2;
...
PrivRelease(DeviceNumber, IOflag);   [Note routine name.]
```

COBOL

```
01  DeviceNumber  COMP.
01  IOflag  COMP.
    ...
    MONITOR-CALL "ForceRelease" USING DeviceNumber, IOflag.
```

FORTRAN

```
      INTEGER  DeviceNumber, IOflag
      ...
      Monitor_Call('ForceRelease', DeviceNumber, IOflag)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

125B **FORCERELEASE** **PRLS**

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, IOflag
...
Monitor_Call('ForceRelease', DeviceNumber, IOflag)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
IOflag : W  BLOCK  1
ForceRelease : EQU 37B9 + 125B
        ...
        CALLG ForceRelease, 2, DeviceNumber, IOflag
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     125     %Monitor call ForceRelease.
        ...
PAR,    DEVNO           %Logical device number.
        IOFL            %Input or Output flag.
        ...
DEVNO,  ...
IOFL,   ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

PRSRV — FORCERESERVE — 124B

Reserves a device for an RT program other than that which is calling. Use ForceRelease if the device is already reserved.

- You can only release peripheral devices, such as terminals and printers, and semaphores in this way.
- Programs in a waiting queue may reserve the device between your ForceRelease and ForceReserve calls if they have higher priority then your program.
- The SINTRAN III REAL TIME GUIDE (ND-60.133) describes this in more detail.

See also ForceRelease, ReserveResource, and @PRSRV.

PARAMETERS

→ Logical device number. See appendix B.
→ Input or output flag. Use 0 for the input part and 1 for the output part.
→ RT description address of the RT program to reserve the device. Use 0 for your own program.
← Return status. 0 means OK. A negative value is returned if the device already was reserved. No value is returned from the ND-500.

PASCAL

```
DeviceNo, IOFlag, RTProgram, Stat : INTEGER2;
...
PrivReserve(DeviceNo, IOFlag, RTProgram, Stat);     [Note routine name.]
```

COBOL

```
01  DeviceNo  COMP.
01  IOFlag  COMP.
01  RTProgram  COMP.
01  Stat  COMP.
    ...
    MONITOR-CALL "ForceReserve" USING DeviceNo, IOFlag, RTProgram, Stat.
```

FORTRAN

```
      INTEGER  DeviceNo, IOFlag, RTProgram, Stat
      ...
      Monitor_Call('ForceReserve', DeviceNo, IOFlag, RTProgram, Stat)
```

ND-100 and ND-500 | **User RT and user SYSTEM** | **RT programs**

124B **FORCERESERVE** **FRSRV**

Continued from previous page...

PLANC

```
INTEGER : DeviceNo, IOFlag, RTProgram, Stat
...
Monitor_Call('ForceReserve', DeviceNo, IOFlag, RTProgram, Stat)
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
IOFlag : W  BLOCK  1
RTProgram : W  BLOCK  1
ForceReserve : EQU 37B9 + 124B
        ...
        CALLG ForceReserve, 3, DeviceNo, IOFlag, RTProgram
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     124     %Monitor call ForceReserve.
        STA     STAT    %Store status returned.
        ...
STAT,   0
PAR,    DEVNO           %Logical device number.
        IOF             %Input or Output flag.
        RTPRO           %Address of RT description.
        ...
DEVNO,  ...
IOF,    ...
RTPRO,  ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

PRT	FORCETRAP	435B

Forces a programmed trap to occur in another ND-500 process. The trap handler in this process is started.

- The trapped process gets your process number through GetTrapReason. It is stored in the upper half of the error code. The lower half contains the reason code you specify as a parameter.

See also GetTrapReason.

PARAMETERS

→ Process number to be trapped.
→ Reason code. Bit 31:16 contains the process number and the magic number.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-500	All users	All programs

435B	FORCETRAP	PRT

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
ProcessNumber : W  BLOCK  1
ReasonCode : W  BLOCK  1
ForceTrap :  EQU  37B9 + 435B
        ...
        CALLG  ForceTrap, 2, ProcessNumber, ReasonCode
```

MAC

Not available.

ND-500	All users	All users

DEABF FULLFILENAME 256B

Returns a complete file name from an abbreviated one. The directory, the user, the file name, the file type, and the version are returned.

- You must have read access to the file.
- The abbreviation must be unambiguous.
- SINTRAN III, version K, allows remote file names.

See also GetFileName.

PARAMETERS

→ The abbreviated file name. You may include a file type.
← Full file name.
→ Default file type. Do not include the colon.
← Standard error code. See appendix A.

PASCAL

```
AbbrevFileName, FileName : PACKED ARRAY [0..63] OF CHAR;
FileType : PACKED ARRAY [0..3] OF CHAR;
...
FullFileName(AbbrevFileName, FileName, FileType);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  AbbrevFileName   PIC X(64).
01  FileName   PIC X(64).
01  FileType   PIC X(4).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "FullFileName" USING AbbrevFileName, FileName, FileType.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      CHARACTER  AbbrevFileName*64, FileName*64, FileType*4
      ...
      Monitor_Call('FullFileName', AbbrevFileName(1:64), FileName(1:64),
     C                  FileType(1:4))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

256B **FULLFILENAME** **DEABF**

Continued from previous page...

PLANC

```
BYTES : AbbrevFileName(0:63), FileName(0:63), FileType(0:3)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('FullFileName', AbbrevFileName, FileName, FileType)
```

ASSEMBLY-500

```
AbbrevFileName : STRINGDATA  'EX'''
FileName : STRING 64
FileType : STRINGDATA 'SYMB'''  %Default file type.
ErrCode : W  BLOCK  1
FullFileName : EQU 37B9 + 256B
        ...
        CALLG FullFileName, 3, AbbrevFileName, FileName, FileType
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (ABBR    %Address of abbreviated file name string.
        LDA     (FILE    %Address of string to receive full file name.
        LDT     (TYPE    %Address of default file type string.
        MON     256      %Monitor call FullFileName.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
ABBR,   'EX'             %Find full file name of EX.
FILE,   0                %Empty string. (A and X may be identical.)
*+76/                    %Make space to receive full file name.
TYPE,   'SYMB'           %Default file type SYMB.
```

ND-100 and ND-500 **All users** **All users**

GASGM — GETACTIVESEGMENT — 421B

Gets the name of the segments in your domain. A 2048 byte buffer is returned. It contains 32 pointers to segment names in the buffer. Each pointer consists of 12 bytes. The first four is an address. The second four is the offset from this address to the start of the segment name. The last four are the offset to the end of the segment name.

- Unused segments have the address 0.

PARAMETERS

→ A 2048 byte buffer, ie., 1 page.

PASCAL

```
Buffer : ARRAY [0..31] OF BITMAP;
...
GetActiveSegment(Buffer);
```

COBOL

```
01  Buffer.
    02  array COMP OCCURS 1024 TIMES.
    ...
    MONITOR-CALL "GetActiveSegment" USING Buffer.
```

FORTRAN

```
      INTEGER  Buffer(1024)
      ...
      Monitor_Call('GetActiveSegment', Buffer(1))
```

ND-500 | All users | All programs

421B	GETACTIVESEGMENT	GASGM

Continued from previous page...

PLANC

```
BYTE ARRAY : Buffer(0:2047)
...
Monitor_Call('GetActiveSegment', Buffer(0))
```

ASSEMBLY-500

```
Buffer : W  BLOCK 1024
GetActiveSegment : EQU 37B9 + 421B
        ...
        CALLG GetActiveSegment, 1, Buffer
```

MAC

Not available.

ND-500	All users	All users

GBSIZ GetAddressArea 222B

Gets the size of your address area. Your address area may consist of one or two 128 Kbyte areas. This depends on the size of the background segments:

- The size of the background segment is defined when SINTRAN III is generated for your computer.

See also @CHANGE-BACKGROUND-SEGMENT-SIZE.

PARAMETERS

← The size of the address area. 100B means one 128 Kbyte address area. 200B means two 128 Kbyte address areas, one for instructions and one for data.

PASCAL

```
SegmentSize : INTEGER2;
...
GetAddressArea(SegmentSize);
```

COBOL

```
01  SegmentSize  COMP.
    ...
    MONITOR-CALL "GetAddressArea" USING SegmentSize.
```

FORTRAN

```
      INTEGER  SegmentSize
      ...
      Monitor_Call('GetAddressArea', SegmentSize)
```

ND-100 All users Background programs

222B **GETADDRESSAREA** **GBSIZ**

Continued from previous page...

PLANC

```
INTEGER : SegmentSize
...
Monitor_Call('GetAddressArea', SegmentSize)
```

ASSEMBLY-500

Not available.

MAC

```
        MON     222     %Monitor call GetAddressArea.
        STA     SIZE    %Store returned size of background segment.
        ...
SIZE,   0
```

ND-100 **All users** **Background programs**

GUIOI GetAllFileIndexes 217B

Gets the directory index, the user index, and the object index of a file. These are indexes in the file system. Appendix C describes the file system.

- The file must be open.
- On the ND-100 you may get this information if the file is on a remote computer system. The computers must be connected through a COSMOS network.

See also GetDirUserIndexes, GetDirNameIndex, GetFileIndexes, GetDirEntry, GetUserEntry, GetObjectEntry, and GetDefaultDir.

PARAMETERS

→ File number. See OpenFile.
← The directory index.
← The user index.
← The object index.
← Remote flag. Set to 0 if the file is on the local computer. A file on a a remote computer returns 1. ND-500 does not use the remote parameters.
← Remote system identification if the remote flag is set to 1. It includes information needed to log, eg., SNURRE(P-HANSEN(&&&&&)). The password is coded as a sequence of control characters.
← Standard error code. See appendix A.

PASCAL

```
FileNo, DirIndex, UserIndex, ObjectIndex : INTEGER2;
...
GetAllFileIndexes(FileNo, DirIndex, UserIndex, ObjectIndex);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNo  COMP.
01  DirIndex  COMP.
01  UserIndex  COMP.
01  ObjectIndex  COMP.
01  RemoteFlag  COMP.
01  RemoteSystem  PIC X(64).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetAllFileIndexes" USING FileNo, DirIndex, UserIndex,
                 ObjectIndex, RemoteFlag, RemoteSystem.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNo, DirIndex, UserIndex, ObjectIndex, RemoteFlag
      CHARACTER RemoteSystem*64
      ...
      Monitor_Call('GetAllFileIndexes', FileNo, DirIndex, UserIndex,
     C                 ObjectIndex, RemoteFlag, RemoteSystem(1:64))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

217B **GETALLFILEINDEXES** **GUIOI**

Continued from previous page...

PLANC

```
INTEGER : FileNo, DirIndex, UserIndex, ObjectIndex, RemoteFlag
BYTES : RemoteSystem(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetAllFileIndexes', FileNo, DirIndex, UserIndex, ObjectIndex,&
                RemoteFlag, RemoteSystem)
```

ASSEMBLY-500

```
FileNo : W  BLOCK  1
DirIndex : W  BLOCK  1     %If bit 7 is set, SysId is a 5th parameter.
UserIndex : W  BLOCK  1
ObjectIndex : W  BLOCK  1
SysId : STRINGARRAY 16     %Remote system name as an optional 5th parameter.
ErrCode : W  BLOCK  1
GetAllFileIndexes : EQU 37B9 + 217B
        ...
        CALLG GetAllFileIndexes, 4, FileNo, DirIndex, UserIndex, ObjectIndex
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDA     FILNO    %File number returned from earlier OpenFile.
        MON     217      %Monitor call GetAllFileIndexes.
        JMP     ERROR    %Error return from monitor call.
        STT     INDEX    %Normal return, store directory and user index.
        STX     OBJIX    %Store object index.
        COPY    SD DA    %Remote identification in D register if bit 15
        STA     REMID    %in T register is set.
        ...
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
INDEX,  0                %Dir index: left byte. User index: right byte.
OBJIX,  0
REMID,  0                %Address of remote identification string.
```

ND-100 and ND-500 **All users** **All users**

TIMR GetBasicTime 11B

Gets the current internal time. The internal time is specified in basic time units. There are 50 basic time units in a second.

- The internal time is set to 0 each time SINTRAN III is started.

See also GetCurrentTime, AdjustClock, and SetClock.

PARAMETERS

← Time in basic time units.

PASCAL

```
BasicTime : LONGINT;
...
GetBasicTime(BasicTime);
```

COBOL

```
01  BasicTime  COMP PIC S9(10).
    ...
    MONITOR-CALL "GetBasicTime" USING BasicTime.
```

FORTRAN

```
      INTEGER*4  BasicTime
      ...
      Monitor_Call('GetBasicTime', BasicTime)
```

ND-100 and ND-500 | All users | All programs

11B **GETBASICTIME** **TIME**

Continued from previous page...

PLANC

```
INTEGER4 : BasicTime
...
Monitor_Call('GetBasicTime', BasicTime)
```

ASSEMBLY-500

```
BasicTime : W  BLOCK  1
GetBasicTime : EQU 37B9 + 11B
        ...
        CALLG GetBasicTime, 0
        W1 =: BasicTime              %Result is returned in W1 register.
```

MAC

```
        MON       11        %Monitor call GetTime.
        STD       TIME      %Store time returned in the A and D register.
        ...
TIME,   0                   %Time in basic time units as a double word.
        0                   %
```

ND-100 and ND-500 **All users** **All users**

GBRK GETBREAKPOINTINFO 46B

Gets information about a breakpoint when debugging a program. The program must have stopped at a defined breakpoint. See DefineBreakpoint. The register contents are stored. Execution continues at the address specified by DefineBreakpoint.

- Both the program and the debug program must be loaded to the same address space.
- DefineBreakpoint must be executed before this call.
- This monitor call is mainly used by the MAC assembler.

See also DefineBreakpoint and SetBreakpoint.

PARAMETERS

← Standard error code. See appendix A.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 | All users | All programs

46B	GETBREAKPOINTINFO	GBRK

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Not available.

MAC

```
        MON     46      %Monitor call GetBreakpointInfo.
```

ND-100	All users	All users

RMAX GETBYTESINFILE 62B

Gets the number of bytes in a file. Only the bytes containing data are counted.

- The file must be open.
- The number of bytes are only relevant to sequentially accessed files.

See also SetMaxBytes and @FILE-STATISTICS.

PARAMETERS

→ File number. See OpenFile.
← Number of bytes in the file.
← Standard error code. See appendix A.

PASCAL

```
FileNumber : INTEGER2;
NoOfBytes : LONGINT;
...
GetBytesInFile(FileNumber, NoOfBytes);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNumber  COMP.
01  NoOfBytes  COMP PIC S9(10).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetBytesInFile" USING FileNumber, NoOfBytes.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNumber
      INTEGER*4  NoOfBytes
      ...
      Monitor_Call('GetBytesInFile', FileNumber, NoOfBytes)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

62B GETBYTESINFILE BMAX

Continued from previous page...

PLANC

```
INTEGER : FileNumber
INTEGER4 : NoOfBytes
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetBytesInFile', FileNumber, NoOfBytes)
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
NoOfBytes : W  BLOCK  1
ErrCode : W  BLOCK  1
GetBytesInFile : EQU 37B9 + 62B
        ...
        CALLG GetBytesInFile, 2, FileNumber, NoOfBytes
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     FILNO   %File number returned from earlier open.
        MON     62      %Monitor call GetBytesInFile.
        JMP     ERROR   %Error return from monitor call.
        STD     BYTES   %Normal return, store the number of bytes obtained.
        ...
ERROR,  ...             %Error number in register A.
        ...
FILNO,  ...
BYTES,  0               %A double word
        0               %
```

ND-100 and ND-500 All users All users

CLOCK **GETCURRENTTIME** **113B**

Gets the current time and date.

- The time is returned as basic time units, seconds, minutes, hours, day, month, and year.

See also GetBasicTime, AdjustClock, SetClock, and @DATCL.

PARAMETERS

← A 14 byte buffer on the ND-100 and a 28 byte buffer on the ND-500. The bytes are used as follows:

0:1	basic time units	(0:3 on the ND-500)
2:3	seconds	(4:7 on the ND-500)
4:5	minutes	(8:11 on the ND-500)
6:7	hours	(12:15 on the ND-500)
8:9	day	(16:19 on the ND-500)
10:11	month	(20:23 on the ND-500)
12:13	year	(24:27 on the ND-500)

PASCAL

```
TimeBuffer : BITMAP;
...
GetCurrentTime(TimeBuffer);
```

COBOL

```
01  TimeBuffer.
    02  array COMP OCCURS 7 TIMES.
    ...
    MONITOR-CALL "GetCurrentTime" USING TimeBuffer.
```

FORTRAN

```
      INTEGER  TimeBuffer(7)
      ...
      Monitor_Call('GetCurrentTime', TimeBuffer(1))
```

ND-100 and ND-500 **All users** **All programs**

113B **GETCURRENTTIME** **CLOCK**

Continued from previous page...

PLANC

```
INTEGER ARRAY : TimeBuffer(0:6)
...
Monitor_Call('GetCurrentTime', TimeBuffer(0))
```

ASSEMBLY-500

```
TimeBuffer : W  BLOCK  7
GetCurrentTime : EQU 37B9 + 113B
        ...
        CALLG GetCurrentTime, 1, TimeBuffer
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     113     %Monitor call GetCurrentTime.
        ...
PAR,    CLOK
        ...
CLOK,   0               %Basic time units.
        0               %Seconds.
        0               %Minutes.
        0               %Hours.
        0               %Day.
        0               %Month.
        0               %Year.
```

ND-100 and ND-500 **All users** **All users**

PDPDI GETDEFAULTDIR 250B

Gets the user's default directory. The directory index and the user index are returned.

- Use ExecutionInfo to get the user index and the directory index of the user executing the program.

See also GetDirUserIndexes, GetDirNameIndex, GetFileIndexes, GetDirEntry, GetUserEntry, GetObjectEntry, and GetAllFileIndexes.

PARAMETERS

→ The user name. A user in a remote system may be identified.
← The directory index.
← the user index in the directory.
← Standard error code. See appendix A.

PASCAL

```
DirectoryIndex, UserIndex : INTEGER2;
UserName : PACKED ARRAY [0..15] OF CHAR;
...
GetDefaultDir(UserName, DirectoryIndex, UserIndex);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DirectoryIndex  COMP.
01  UserIndex  COMP.
01  UserName   PIC X(16).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetDefaultDir" USING UserName, DirectoryIndex, UserIndex.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DirIndex, UserIndex
      CHARACTER  UserName*16
      ...
      Monitor_Call('GetDefaultDir', UserName(1:16), DirIndex, UserIndex)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

250B **GETDEFAULTDIR** **FDFDI**

Continued from previous page...

PLANC

```
INTEGER : DirectoryIndex, UserIndex
BYTES : UserName(0:15)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetDefaultDir', UserName, DirectoryIndex, UserIndex)
```

ASSEMBLY-500

```
DirectoryIndex : W  BLOCK  1
UserIndex : W  BLOCK  1
UserName : STRINGDATA  'A-HANSEN'''
ErrCode : W  BLOCK  1
GetDefaultDir : EQU 37B9 + 250B
        ...
        CALLG GetDefaultDir, 3, UserName, DirectoryIndex, UserIndex
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (USER    %Address of string containing user name.
        MON     250      %Monitor call GetDefaultDir.
        JMP     ERROR    %Error return from monitor call.
        STT     DIRIX    %Normal return, store directory index.
        STX     USRIX    %Store user index in default directory.
        ...
ERROR,  ...              %Error number in register A.
        ...
USER,   'A-HANSEN'       %To find default directory of user A-HANSEN.
DIRIX,  0
USRIX,  0
```

ND-100 and ND-500 **All users** **All users**

GDEVT GetDeviceType 263B

Gets the device type, eg., terminal, floppy disk, mass storage file, etc. The monitor call also provides information on how to handle the device.

PARAMETERS

→ Logical device number. See appendix B.
→ Input or output part. Use 0 for input and 1 for output.
← Device type. The numbers below are returned.

0: Unspecified.	1: Terminal.
2: Terminal access device (TAD).	3: Communication channel.
4: Internal block device.	5: Floppy disk drive.
6: Magnetic tape station.	7: Mass storage file.

← Device information. The bits have the following meaning.

Bit 0: InByte or OutByte allowed.	Bit 1: StartOnInterrupt allowed.
Bit 2: DeviceControl allowed.	Bit 3: Block calls allowed.
Bit 4: ClearDevice available.	Bit 5: Reservation not needed.
Bit 6: COSMOS remote open file.	

← Standard error code. See appendix A.

PASCAL

```
DeviceNo, IOFlag, DevType : INTEGER2;
DevAttr : LONGINT;
....
GetDeviceType(DeviceNo, IOFlag, DevType, DevAttr);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNo  COMP.
01  IOFlag  COMP.
01  DevType  COMP.
01  DevAttr  COMP PIC S9(10).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetDeviceType" USING DeviceNo, IOFlag, DevType, DevAttr.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNo, IOFlag, DevType
      INTEGER*4  DevAttr
      ...
      Monitor_Call('GetDeviceType', DeviceNo, IOFlag, DevType, DevAttr)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

263B GETDEVICETYPE GDEVT

Continued from previous page...

PLANC

```
INTEGER : DeviceNo, IOFlag, DevType
INTEGER4 : DevAttr
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetDeviceType', DeviceNo, IOFlag, DevType, DevAttr)
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
IOFlag : W  BLOCK  1
DevType : W  BLOCK  1
DevAttr : W  BLOCK  1
ErrCode : W  BLOCK  1
GetDeviceType : EQU 37B9 + 263B
        ...
        CALLG GetDeviceType, 4, DeviceNo, IOFlag, DevType, DevAttr
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Logical device number.
        LDA     IOF      %Input/output flag.
        MON     263      %Monitor call GetDeviceType.
        JMP     ERROR    %Error return from monitor call.
        STT     TYPE     %Normal return, store device type.
        STD     ATBUT    %Store device attributes.
        ...
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
IOF,    ...
TYPE,   0
ATBUT,  0                %A double number.
        0                %
```

ND-100 and ND-500 | All users | All users

GDIEN GETDIRENTRY 244B

Get information about a directory. The directory entry is returned. Appendix C describes the file system in more detail.

- On ND-100 you may access directories on remote systems. The computers must be connected through a COSMOS network.

See also GetDirUserIndexes, WriteDirEntry, GetUserEntry, and GetObjectEntry.

PARAMETERS

→ The directory index. See GetDirUserIndexes.
← The 48 byte directory entry. See appendix C.
→ Remote flag. Use 0 for the local computer and 1 for a remote computer.
→ Remote system identification if remote flag is 1. Not used on ND-500.
← Standard error code. See appendix A.

PASCAL

```
DirectoryIndex, Flag : INTEGER2;
DirEntry : ARRAY [0..1] OF BITMAP;
...
GetDirEntry(DirectoryIndex, DirEntry, Flag);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DirectoryIndex  COMP.
01  Flag  COMP.
01  DirEntry.
    02  array COMP OCCURS 24 TIMES.
01  RemoteFlag  COMP.
01  RemoteSystem  PIC X(64).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetDirEntry" USING DirectoryIndex, DirEntry, Flag,
                 RemoteFlag, RemoteSystem)
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DirectoryIndex, Flag, RemoteFlag
      CHARACTER RemoteSystem*64
      INTEGER  DirEntry(24)
      ...
      Monitor_Call('GetDirEntry', DirectoryIndex, DirEntry(1), Flag,
     C                  RemoteFlag, RemoteSystem(1:16))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **All programs**

244B **GETDIRENTRY** **GDIEN**

Continued from previous page...

PLANC

```
INTEGER : DirectoryIndex, Flag, RemoteFlag
BYTE ARRAY : DirEntry(0:47)
BYTES : RemoteSystem(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetDirEntry', DirectoryIndex, DirEntry(0), Flag, RemoteFlag,&
                RemoteSystem)
```

ASSEMBLY-500

```
DirectoryIndex : W  BLOCK  1  %Bit 7 set if SysId is supplied.
Flag : W  BLOCK  1
DirEntry : W  ARRAY 25B
SysId : STRINGARRAY 16      %Remote system name as an optional 4th parameter.
ErrCode : W  BLOCK  1
GetDirEntry : EQU 37B9 + 244B
        ...
        CALLG GetDirEntry, 3, DirectoryIndex, DirEntry, Flag
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                              %ErrorCode in W1 register.
```

MAC

```
        LDT     DIRIX     %Directory index.
        LDX     (BUFF     %Buffer to receive directory entry.
        LDA     (REMID    %Remote system identification in D register.
        COPY    SA DD     %Used only if bit 15 in X register is set.
        MON     244       %Monitor call GetDirEntry.
        JMP     ERROR     %Error return from monitor call.
        ...               %Normal return.
ERROR,  ...               %Error number in register A.
        ...
DIRIX,  ...               %Set bit 15 if remote system.
BUFF,   0                 %
*+30/                     %24 word long buffer.
REMID,  0                 %Remote identification string.
*+40/                     %32 words for string.
```

ND-100 and ND-500 **User RT and user SYSTEM** **All users**

FDINA GETDIRNAMEINDEX 243B

Get directory index and name index. The name index identifies the device description of the disk. You have to specify the directory name.

See also GetDirUserIndexes, GetAllFileIndexes, and GetDirEntry.

PARAMETERS

→ Directory name.
← Directory index.
← Name index.
← Standard error code. See appendix A.

PASCAL

```
DirName : PACKED ARRAY [0..15] OF CHAR;
DirIndex, NameIndex : INTEGER2;
...
GetDirNameIndex(DirName, DirIndex, NameIndex);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DirName   PIC X(16).
01  DirIndex  COMP.
01  NameIndex  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetDirNameIndex" USING DirName, DirIndex, NameIndex.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      CHARACTER  DirName*16
      INTEGER  DirIndex, NameIndex
      ...
      Monitor Call('GetDirNameIndex', DirName(1:16), DirIndex, NameIndex)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

243B **GETDIRNAMEINDEX** **FDIRA**

Continued from previous page...

PLANC

```
BYTES : DirName(0:15)
INTEGER : DirIndex, NameIndex
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetDirNameIndex', DirName, DirIndex, NameIndex)
```

ASSEMBLY-500

```
DirName : STRINGDATA  'PACK-TWO'''       %Get index of directory PACK-TWO.
DirIndex : W  BLOCK  1
NameIndex : W  BLOCK  1
ErrCode : W  BLOCK  1
GetDirNameIndex : EQU 37B9 + 243B
        ...
        CALLG GetDirNameIndex, 3, DirName, DirIndex, NameIndex
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (DIRNAM  %Address of directory name string.
        MON     243      %Monitor call GetDirNameIndex.
        JMP     ERROR    %Error return from monitor call.
        STT     DIRIX    %Normal return, store directory index.
        STA     NAMIX    %Store name index.
        ...
ERROR,  ...              %Error number in register A.
        ...
DIRNAM, 'PACK-TWO'       %Obtain directory and name index of PACK-TWO.
DIRIX,  0
NAMIX,  0
```

ND-100 and ND-500 **All users** **All users**

MUIDI GetDirUserIndexes 213B

Get a directory index and a user index. You have to specify a directory name and a user name.

- Use ExecutionInfo to get the user index and the directory index of the user executing a program.

See also GetDirNameIndexes, GetUserIndex, and GetAllFileIndexes.

PARAMETERS

→ Directory name and user name, eg., PACK-ONE:P-HANSEN. If no colon exists the name is taken as a user name. The user's default directory index is returned.
← Directory index.
← User index.
← Standard error code. See appendix A.

PASCAL

```
UserName : PACKED ARRAY [0..15] OF CHAR;
DirIndex, UserIndex : INTEGER2;
...
GetDirUserIndexes(UserName, DirIndex, UserIndex);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  UserName   PIC X(16).
01  DirIndex  COMP.
01  UserIndex  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetDirUserIndexes" USING UserName, DirIndex, UserIndex.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      CHARACTER  UserName*16
      INTEGER  DirIndex, UserIndex
      ...
      Monitor_Call('GetDirUserIndexes',UserName(1:16),DirIndex,UserIndex)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

213B **GETDIRUSERINDEXES** **MUTDI**

Continued from previous page...

PLANC

```
BYTES : UserName(0:15)
INTEGER : DirIndex, UserIndex
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetDirUserIndexes', UserName, DirIndex, UserIndex)
```

ASSEMBLY-500

```
UserName : STRINGDATA  'A-HANSEN'''
DirIndex : W  BLOCK  1
UserIndex : W  BLOCK  1
ErrCode : W  BLOCK  1
GetDirUserIndexes : EQU 37B9 + 213B
        ...
        CALLG GetDirUserIndexes, 3, UserName, DirIndex, UserIndex
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (USER    %Address of string with directory and user name.
        MON     213      %Monitor call GetDirUserIndexes.
        JMP     ERROR    %Error return from monitor call.
        STX     INDEX    %Normal return, store indexes.
        ...
ERROR,  ...              %Error number in register A.
        ...
USER,   'A-HANSEN'       %Obtain indexes for A-HANSEN.
INDEX,  0                %Left byte: dir index. Right byte: user index.
```

ND-100 and ND-500 **All users** **All users**

GETERRORDEVICE

GEEDV 254B

Gets the logical device number of the error device. The error device may be reserved by an RT program. If this is the case, the monitor call returns the address of the RT description. The error device is the terminal which outputs system errors and RT program messages. The error device is normally the console.

See also @GET-ERROR-DEVICE and @SET-ERROR-DEVICE.

PARAMETERS

← The logical device number of the error device.
← RT description address of reserving RT program. 0 means unreserved.

PASCAL

```
ErrorDevice, RTProgram : INTEGER2;
...
FindErrorDevice(ErrorDevice, RTProgram);     [Note routine name.]
```

COBOL

```
01  ErrorDevice  COMP.
01  RTProgram  COMP.
    ...
    MONITOR-CALL "GetErrorDevice" USING ErrorDevice, RTProgram.
```

FORTRAN

```
      INTEGER  ErrorDevice, RTProgram
      ...
      Monitor_Call('GetErrorDevice', ErrorDevice, RTProgram)
```

ND-100 and ND-500 User SYSTEM All programs

254B GETERRORDEVICE GERDV

Continued from previous page...

PLANC

```
INTEGER : ErrorDevice, RTProgram
...
Monitor_Call('GetErrorDevice', ErrorDevice, RTProgram)
```

ASSEMBLY-500

```
ErrorDevice : W  BLOCK  1
RTProgram : W  BLOCK  1
GetErrorDevice : EQU 37B9 + 254B
        ...
        CALLG GetErrorDevice, 2, ErrorDevice, RTProgram
```

MAC

```
        MON     254     %Monitor call GetErrorDevice.
        STA     ERDEV   %Store logical number of error device.
        COPY    SD  DA
        STA     RTPRO   %Store address of RT description.
        ...
ERDEV,  0
RTPRO,  0
```

ND-100 and ND-500 User SYSTEM All users

RERRP — GETERRORINFO — 207B

Gets information about the last real time error. The monitor call returns the error, the RT program responsible for the error, and the program address where it occurred. A flag tells whether the RT program was aborted or not.

PARAMETERS

← A 12 byte error information. It has the meaning below.

Byte 0:1	Error number as two ASCII characters. Parity bits are not set. See appendix A and F for error codes together with an ASCII table.
2:3	Pointer to the user program address where the error occurred.
4:5	Additional error information. See appendix A.
6:7	Additional error information. See appendix A.
8:9	RT description address of the RT program causing the error.
10:11	Flag. 0 if SINTRAN III aborted the RT program.

← Returned status. 0 means OK, 153B means illegal output buffer.

PASCAL

```
Buffer : BITMAP;
ReturnStatus : INTEGER2;
...
ReadErrorParam(Buffer, ReturnStatus);   [Note routine name.]
```

COBOL

```
01  Buffer.
    02  array COMP OCCURS 6 TIMES.
01  ReturnStatus  COMP.
    ...
    MONITOR-CALL "GetErrorInfo" USING Buffer, ReturnStatus.
```

FORTRAN

```
      INTEGER  Buffer(6)
      INTEGER  ReturnStatus
      ...
      Monitor_Call('GetErrorInfo', Buffer(1), ReturnStatus)
```

ND-100 — User RT and user SYSTEM — RT programs

207B GETERRORINFO RERBP

Continued from previous page...

PLANC

```
BYTE ARRAY : Buffer(0:11)
INTEGER : ReturnStatus
...
Monitor_Call('GetErrorInfo', Buffer(0), ReturnStatus)
```

ASSEMBLY-500

Not available.

MAC

```
        LDA     (PAR      %Load register A with address of parameter list.
        MON     207       %Monitor call GetErrorInfo.
        JAF     ERROR     %Do error handling if register A is non-zero.
        ...
ERROR,  ...               %Error number in register A.
        ...
PAR,    BUFF              %Buffer for returned error information.
        ...
BUFF,   0
        0
        0
        0
        0
        0
```

ND-100 User RT and user SYSTEM RT programs

GETXM GetErrorMessage 334B

Gets a file system error message text. Appendix A shows the messages connected to each error code. The error code is input. The program continues.

- This monitor call is convenient for advanced use of the terminal screen. For example, you may output the error message in inverse video at the bottom line.
- Error code 0 is illegal on input.

See also WarningMessage and ErrorMessage. The last monitor call writes out the error message and terminates the program.

PARAMETERS

→ Error code of the message to be printed. Use octal numbers.
← Error message text.
← Standard error code. See appendix A.

PASCAL

```
ErrCode : INTEGER2;
Buffer : PACKED ARRAY [0..127] OF CHAR;
...
GetErrorMessage(ErrCode, Buffer);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  ErrCode  COMP.
01  Buffer   PIC X(128).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetErrorMessage" USING ErrCode, Buffer.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  ErrCode
      CHARACTER  Buffer*128
      ...
      Monitor_Call('GetErrorMessage', ErrCode, Buffer(1:128))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

334B **GETERRORMESSAGE** **GETERR**

Continued from previous page...

PLANC

```
INTEGER : ErrCode
BYTES : Buffer(0:127)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetErrorMessage', ErrCode, Buffer)
```

ASSEMBLY-500

```
ErrCode : W  BLOCK  1
Buffer : W  BLOCK  100
ErrCode : W  BLOCK  1
GetErrorMessage : EQU 37B9 + 334B
        ...
        CALLG GetErrorMessage, 2, ErrCode, Buffer
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDA     ERRNO    %Error number of error message to be printed.
        LDX     (BUF     %Address of buffer to receive the error message.
        MON     334      %Monitor call GetErrorMessage.
        STA     STAT     %Returns here if error occur.
        ...              %Normal return here.
ERRNO,  ...
BUF,    ...              %Buffer space for 128 bytes.
*+100/
STAT,   ...
```

ND-100 and ND-500 **All users** **All users**

MGDAB GetEscLocalChars 2308

You can terminate most programs with the ESCAPE key. A LOCAL key has a similar function. It terminates a connection to a remote computer in a COSMOS network. The system supervisor may select other keys for these functions. This monitor call tells you which keys to use.

- The local function is only used by COSMOS.

See also SetEscapeHandling, SetEscLocalChars, OnEscLocalChars, and OffEscLocalChars.

PARAMETERS

→ Logical device number. Only used by RT programs. Your own terminal is always used in background program. See appendix B.
← The local character.
← The escape character.

PASCAL

```
DeviceNo, DisconnectChar, EscapeChar : INTEGER2;
...
GetEscLocalChar(DeviceNo, DisconnectChar, EscapeChar);
```

COBOL

```
01  DeviceNo  COMP.
01  DisconChar  COMP.
01  EscapeChar  COMP.
    ...
    MONITOR-CALL "GetEscLocalChar" USING DeviceNo, DisconChar, EscapeChar.
```

FORTRAN

```
      INTEGER  DeviceNo, DisconChar, EscapeChar
      ...
      Monitor_Call('GetEscLocalChar', DeviceNo, DisconChar, EscapeChar)
```

ND-100 and ND-500 | All users | All programs

230B **GETESCLOCALCHARS** **MODAE**

Continued from previous page...

PLANC

```
INTEGER : DeviceNo, DisconnectChar, EscapeChar
...
Monitor_Call('GetEscLocalChar', DeviceNo, DisconnectChar, EscapeChar)
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
DisconnectChar : W  BLOCK  1
EscapeChar : W  BLOCK  1
GetEscLocalChar : EQU 37B9 + 230B
        ...
        CALLG GetEscLocalChar, 3, DeviceNo, DisconnectChar, EscapeChar
```

MAC

```
        LDT     DEVNO    %Logical device number.
        MON     230      %Monitor call GetEscLocalChar.
        STA     CHAR     %Store returned characters.
        ...
DEVNO,  ...
CHAR,   0                %Left byte: discon char. Right byte: escape char.
```

ND-100 and ND-500 **All users** **All users**

POBJN GetFileIndexes 274B

Gets the directory index, the user index, and the object index of a file. These are indexes in the file system. See the SINTRAN III SYSTEM SUPERVISOR (ND-30.003) for more details.

- The file need not to be open.

See also GetAllFileIndexes, GetDirUserIndexes, GetDirNameIndex, GetDirEntry, GetUserEntry, GetObjectEntry, and GetDefaultDir.

PARAMETERS

→ File name. Abbreviated file names are less efficient.
→ File type. Not used on the ND-100. Do not include the colon.
← Directory index.
← User index.
← Object index.
← Object index of the next file version. Equal to object index if no more versions exist.
← Standard error code. See appendix A.

PASCAL

```
FileName : PACKED ARRAY [0..63] OF CHAR;
FileType : PACKED ARRAY [0..3] OF CHAR;
DirIndex, UserIndex, ObjectIndex, NextObjectIndex : INTEGER2;
...
FindFileIndexes(FileName, FileType, DirIndex, UserIndex,
                ObjectIndex, NextObjectIndex);  [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileName    PIC X(64).
01  FileType    PIC X(4).
01  DirIndex  COMP.
01  UserIndex  COMP.
01  ObjectIndex  COMP.
01  NextObjectIndex  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetFileIndexes" USING FileName, FileType, DirIndex,
                         UserIndex, ObjectIndex, NextObjectIndex.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      CHARACTER  FileName*64, FileType*4
      INTEGER  DirIndex, UserIndex, ObjectIndex, NextObjectIndex
      ...
      Monitor_Call('GetFileIndexes', FileName(1:64), FileType(1:4),
     C                 DirIndex, UserIndex, ObjectIndex, NextObjectIndex)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

274B GetFileIndexes POBJW

Continued from previous page...

PLANC

```
BYTES : FileName(0:63), FileType(0:3)
INTEGER : DirIndex, UserIndex, ObjectIndex, NextObjectIndex
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetFileIndexes', FileName, FileType, DirIndex, &
                               UserIndex, ObjectIndex, NextObjectIndex)
```

ASSEMBLY-500

```
FileName : STRINGARRAY 64
FileType : STRINGARRAY 4
DirIndex : W  BLOCK  1
UserIndex : W  BLOCK  1
ObjectIndex : W  BLOCK  1
NextObjectIndex : W  BLOCK  1
ErrCode : W  BLOCK  1
GetFileIndexes : EQU 37B9 + 274B
        ...
        CALLG GetFileIndexes, 6, FileName, FileType, DirIndex, &
                               UserIndex, ObjectIndex, NextObjectIndex
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (FILE    %Address of file name string.
        MON     274      %Monitor call GetFileIndexes.
        JMP     ERROR    %Error return from monitor call.
        STT     INDEX    %Normal return, store obtained indexes.
        STA     OBJIX    %Store object index.
        COPY    SD  DA
        STA     NEXTO    %Store object index of next version.
        ...
ERROR,  ...              %Error number in register A.
        ...
FILE,   'EXAMPLE:SYMB'   %Obtain object index of file EXAMPLE:SYMB.
INDEX,  0                %Directory index in left byte. User index right.
OBJIX,  0
NEXTO,  0
```

ND-100 and ND-500 | All users | All users

MSFIL GetFileName 2738

Gets the name of a file. You specify the directory index, the user index, and the object index. The file name, the file type, and the version are returned.

- The file need not to be open.
- On the ND-100 you may specify a file on a remote computer system. The computers must be connected through a COSMOS network.

See also FullFileName.

PARAMETERS

→ Directory index.
→ User index.
→ Object index.
← File name.
→ Remote flag. Use 0 for a file on the local computer. Use 1 for a file on a remote computer. Specify the remote computer in the next parameter.
→ Remote system identification if remote flag is 1. Not returned by ND-500.
← Standard error code. See appendix A.

PASCAL

```
DirIndex, UserIndex, ObjectIndex : INTEGER2;
FileName : PACKED ARRAY [0..63] OF CHAR;
...
GetFileName(DirIndex, UserIndex, ObjectIndex, FileName);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DirIndex  COMP.
01  UserIndex  COMP.
01  ObjectIndex  COMP.
01  FileName   PIC X(64).
01  RemoteFlag COMP.
01  RemoteSystem   PIC X(64).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetFileName" USING DirIndex, UserIndex, ObjectIndex,
                 FileName, RemoteFlag, RemoteSystem.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DirIndex, UserIndex, ObjIndex, RemoteFlag
      CHARACTER  FileName*64, RemoteSystem*64
      ...
      Monitor_Call('GetFileName', DirIndex, UserIndex, ObjIndex,
     C                 FileName(1:64), RemoteFlag, RemoteSystem(1:64))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

273B **GETFILENAME** **MSFIL**

Continued from previous page...

PLANC

```
INTEGER : DirIndex, UserIndex, ObjectIndex, RemoteFlag
BYTES : FileName(0:63), RemoteSystem(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetFileName', DirIndex, UserIndex, ObjectIndex, FileName,&
                RemoteFlag, RemoteSystem)
```

ASSEMBLY-500

```
DirIndex : W  BLOCK  1
UserIndex : W  BLOCK  1
ObjectIndex : W  BLOCK  1
FileName : STRING 64
SysId : STRINGARRAY 20   %Optional parameter if bit 7 in DirIndex is set.
ErrCode : W  BLOCK  1
GetFileName : EQU 37B9 + 273B
        ...
        CALLG GetFileName, 4, DirIndex, UserIndex, ObjectIndex, FileName
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     INDEX    %Left byte: dir index. Right byte: user index.
        LDA     (REMID   %Remote system identification in register D.
        COPY    SA DD    %Used only if bit 15 in register A is set.
        LDA     OBJIX    %Object index.
        LDX     (BUFF    %Address of buffer to receive file name.
        MON     273      %Monitor call GetFileName.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
INDEX,  ...              %Set bit 15 if remote file.
OBJIX,  ...
BUFF,   0                %
*+32/                    %Make a buffer big enough to receive file name.
REMID,  0                %Remote identification string.
*+40/                    %32 words for string.
```

ND-100 and ND-500 **All users** **All users**

RFLAG — GetInputFlags — 402B

ND-100 and ND-500 programs may communicate through two 32-bit flag arrays. You can use the flags as you wish. GetInputFlags reads the input flags. The ND-100 sets these flags with the monitor call ND500Function. See the ND-500 LOADER/MONITOR (ND-60.136).

- You get the last values written to the flags. There is no queue.

See also SetOutputFlags and ND500Function.

PARAMETERS

→ Flag values as a 32-bit integer.

PASCAL

```
Value : LONGINT;
...
GetInputFlags(Value);
```

COBOL

```
01  Value  COMP.
    ...
    MONITOR-CALL "GetInputFlags" USING Value.
```

FORTRAN

```
      INTEGER  Value
      ...
      Monitor_Call('GetInputFlags', Value)
```

ND-500 | All users | All programs

402B	GETINPUTFLAGS	RFLAG

Continued from previous page...

PLANC

```
INTEGER : Value
...
Monitor_Call('GetInputFlags', Value)
```

ASSEMBLY-500

```
Value : W  BLOCK  1
GetInputFlags : EQU 37B9 + 402B
        ...
        CALLG GetInputFlags, 1, Value
```

MAC

Not available.

ND-500	All users	All users

DIW GETINREGISTERS 165B

Reads the device interface registers. Intended for internal usage.

See also SetOutRegisters.

PARAMETERS

→ Number of registers.
→ Buffer with logical unit.
→ Data buffer.
← Error indicator.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 and ND-500 User RT and user SYSTEM RT programs

165B **GETINREGISTERS** **DIW**

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
NoOfReg : W  BLOCK  1
Buffer : W  BLOCK  1024
DataBuffer : W  BLOCK  1024
ErrorIndicator : W  BLOCK  1
        ...
GetInRegister : EQU  37B9 + 165B
        CALLG GetInRegister, 4, NoOfReg, Buffer, DataBuffer, ErrorIndicator
```

MAC

Internal usage.

Trailer

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

LASTC GETLASTBYTE 26B

Gets the last character typed on a terminal. The monitor call can be used to terminate long output sequences, reading from files, etc. The program never enter the I/O wait state because of this monitor call.

- You do not need to reserve the terminal for your program.

See also InByte.

PARAMETERS

→ Logical device number of a terminal.
← The last character typed on the terminal.
Errors in the parameters returns -1.

PASCAL

```
DeviceNumber, LastCharTyped : INTEGER2;
...
GetLastByte(DeviceNumber, LastCharTyped);
```

COBOL

```
01  DeviceNumber  COMP.
01  LastCharTyped  COMP.
    ...
    MONITOR-CALL "GetLastByte" USING DeviceNumber, LastCharTyped.
```

FORTRAN

```
      INTEGER  DeviceNumber, LastCharTyped
      ...
      Monitor_Call('GetLastByte', DeviceNumber, LastCharTyped)
```

ND-100 and ND-500 User SYSTEM All programs

26B **GETLASTBYTE** **LASTC**

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, LastCharTyped
...
Monitor_Call('GetLastByte', DeviceNumber, LastCharTyped)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
LastCharTyped : W  BLOCK  1
GetLastByte : EQU 37B9 + 26B
        ...
        CALLG GetLastByte, 1, DeviceNumber
        W1 =: LastCharTyped             %Result is returned in W1 register.
```

MAC

```
        LDA     (PAR      %Load register A with address of parameter list.
        MON     26        %Monitor call GetLastByte.
        JAN     ERROR     %Error in parameters return -1.
        STA     CHAR      %No errors, store byte returned.
        ...
ERROR,  ...               %Handle the error.
        ...
CHAR,   0                 %Last typed character right adjusted.
PAR,    DEVNO             %Logical device number of a terminal.
        ...
DEVNO,  ...
```

ND-100 and ND-500 **User SYSTEM** **All users**

GETNAMEENTRY 245B

Gets information about devices, eg., disks and floppy disks. The monitor call returns the name entry of a device. You specify the name index.

- GetDirNameIndex provides the name index.

See also GetDirEntry.

PARAMETERS

→ The name index of the device.
← The 28 byte name entry. It contains the following:

0:13	Device name.
16:19	Storage capacity in pages.
20:21	Sector size of the disk.
22:23	Various flags.
	Bit 15 Cartridge disk.
	Bit 14 Drum.
	Bit 13 Single user device, eg., floppy disk or magnetic tape.
	Bit 12 10 Mbyte cartridge disk.
	Bit 11 Magnetic tape station.
	Bit 10 EEC disks.
	Bit 9 Big disk.
	Bit 8 Floppy disk.
	Bit 7 Phoenix disk.
	Bit 2:0 Number of directories for the disk.
24:25	Address of data transfer routine in SINTRAN III.
26:27	Logical device number of the disk's semaphore.

← Standard error code. See appendix A.

PASCAL

```
NameIndex : INTEGER2;
NameTableEntry : BITMAP;
...
GetNameEntry(NameIndex, NameTableEntry);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  NameIndex  COMP.
01  NameTableEntry.
    02  array COMP OCCURS 14 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetNameEntry" USING NameIndex, NameTableEntry.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  NameIndex
      INTEGER  NameTableEntry(14)
      ...
      Monitor_Call('GetNameEntry', NameIndex, NameTableEntry(1))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

245B GETNAMEENTRY GNAEEN

Continued from previous page...

PLANC

```
INTEGER : NameIndex
BYTE ARRAY : NameTableEntry(0:27)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetNameEntry', NameIndex, NameTableEntry(0))
```

ASSEMBLY-500

```
NameIndex : W  BLOCK  1
NameTableEntry : BY  BLOCK  34B
ErrCode : W  BLOCK  1
GetNameEntry : EQU 37B9 + 245B
        ...
        CALLG GetNameEntry, 2, NameIndex, NameTableEntry
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     NAMIX    %Name index.
        LDX     (BUFF    %Address of buffer to receive name table entry.
        MON     245      %Monitor call GetNameEntry.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
NAMIX,  ...
BUFF,   0
*+16/                    %Make a buffer of 14 words.
```

ND-100 and ND-500 All users All users

SPAGET GetND500Param 437B

Gets information about why the last ND-500 program terminated. There are 5 parameters for each background user. These can be set by SINTRAN III or your background program.

- Use SetND500Param to set the parameter values.
- SINTRAN III sets some of the parameter values if you give the command @ENABLE-TERMINATION-HANDLING first.

See also TerminationHandling, SetND500Param, GetUserParam, and SetUserParam.

PARAMETERS

← The five user parameters as an array. SINTRAN III's termination handling returns the following:

Parameter 1: Bit 24:16 contain the user index. Bit 7:0 contains the directory index.
2: Logical device number of the terminal.
3: Fatal error or the monitor call ErrorMessage returns the error number. If escape was pressed, -1 is returned.
4: User defined.
5: User defined.

The parameters are returned if the user presses the ESCAPE key, if the monitor calls ExitFromProgram or ErrorMessage are executed, or if a fatal error occurs. All parameters can be user defined if no termination handling is enabled.

PASCAL

```
Buffer : BITMAP;
...
GetND500Param(Buffer);
```

COBOL

```
01  Buffer.
    02  array COMP OCCURS 5 TIMES.
    ...
    MONITOR-CALL "GetND500Param" USING Buffer.
```

FORTRAN

```
      INTEGER  Buffer(5)
      ...
      Monitor_Call('GetND500Param', Buffer(1))
```

ND-500 All users Background programs

437B	GetND500Param	SPAGET

Continued from previous page...

PLANC

```
BYTE ARRAY : Buffer(0:9)
...
Monitor_Call('GetND500Param', Buffer(0))
```

ASSEMBLY-500

```
Buffer : W  BLOCK  5
GetND500Param : EQU 37B9 + 437B
        ...
        CALLG GetND500Param, 1, Buffer
```

MAC

Not available.

ND-500	All users	Background programs

DBOBJ GETOBJECTENTRY 215B

Gets information about a file. An object entry describes each file. It contains the file name, the access rights, the date last opened for read and write, the size, and more. See the file system description in appendix C. You specify the directory index, the user index, and the object index.

- There is one object entry for each version of a file.
- Only user SYSTEM can get the object entry of a file without read or write access to the file.
- On ND-100 you may access files on remote computer systems. The computers must be connected through a COSMOS network.

See also ReadObjectEntry, SetObjectEntry, and GetAllFileIndexes.

PARAMETERS

← The 64 byte object entry.
→ The directory index. See GetAllFileIndexes.
→ The user index. See GetAllFileIndexes.
→ The object index. See GetAllFileIndexes.
→ Remote flag. Use 0 for the local computer and 1 for a remote computer.
→ Remote system identification if remote flag is 1. Not used on ND-500.
← Standard error code. See appendix A.

PASCAL

```
DirIndex, UserIndex, ObjectIndex : INTEGER2;
Buffer : ARRAY [0..1] OF BITMAP;
...
GetObjectEntry(Buffer, DirIndex, UserIndex, ObjectIndex);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DirIndex  COMP.
01  UserIndex  COMP.
01  ObjectIndex  COMP.
01  RemoteFlag  COMP.
01  RemoteSystem  PIC X(64).
01  Buffer.
    02  array COMP OCCURS 32 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetObjectEntry" USING Buffer, DirIndex, UserIndex,
                 ObjectIndex, RemoteFlag, RemoteSystem)
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DirIndex, UserIndex, ObjectIndex, RemoteFlag
      CHARACTER RemoteSystem*64
      INTEGER  Buffer(32)
       ...
      Monitor_Call('GetObjectEntry', Buffer(1), DirIndex, UserIndex,
                   ObjectIndex, RemoteFlag, RemoteSystem(1:64))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

215B **GETOBJECTENTRY** **DROBJ**

Continued from previous page...

PLANC

```
INTEGER : DirIndex, UserIndex, ObjIndex, RemoteFlag
BYTES : RemoteSystem(0:63)
BYTE ARRAY : Buffer(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetObjectEntry', Buffer(0), DirIndex, UserIndex, ObjIndex,&
                RemoteFlag, RemoteSystem)
```

ASSEMBLY-500

```
DirIndex : W  BLOCK  1
UserIndex : W  BLOCK  1
ObjectIndex : W  BLOCK  1
Buffer : STRINGARRAY 20B
SysId : STRINGARRAY 16     %Optional parameter if bit 7 in DirIndex set.
ErrCode : W  BLOCK  1
GetObjectEntry : EQU 37B9 + 215B
        ...
        CALLG GetObjectEntry, 4, Buffer, DirIndex, UserIndex ObjectIndex
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDA     (REMID   %Remote identification in register D.
        COPY    SA  DD   %Set bit 15 in register T if remote file.
        LDA     (BUFF    %Address of buffer for returned object entry.
        LDT     INDEX    %Left byte: Dir index. Right byte: User index.
        LDX     OBJIX    %Object index.
        MON     215      %Monitor call GetObjectEntry.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
BUFF,   0
*+40/                    %Make a buffer of 32 words.
INDEX,  ...              %Set bit 15 if remote object entry.
OBJIX,  ...
REMID,  0                %Remote identification string..
*+40/                    %Make a buffer of 32 words.
```

ND-100 and ND-500 **All users** **All users**

GPRNAME GetOwnProcessInfo 427B

Gets the name and number of your own process in the ND-500. You get a process each time you enter the ND-500-MONITOR.

- The default process names are TERMINAL-xx where xx is a number.

See also GetProcessNo and SetProcessName.

PARAMETERS

← Process name, maximum 35 bytes.
← Process number in bit 31:16 and magic number in bit 15:0.
← Standard error code. See appendix A.

PASCAL

```
ProcessName : PACKED ARRAY [0..33] OF CHAR;
ProcessNumber : LONGINT;
...
GetOwnProcessInfo(ProcessName, ProcessNumber);
```

COBOL

```
01  ProcessName    PIC X(34).
01  ProcessNumber  COMP.
    ...
    MONITOR-CALL "GetOwnProcessInfo" USING ProcessName, ProcessNumber.
```

FORTRAN

```
      CHARACTER  ProcessName*34
      INTEGER  ProcessNumber
      ...
      Monitor_Call('GetOwnProcessInfo', ProcessName(1:34), ProcessNumber)
```

ND-500 All users All programs

427B **GETOWNPROCESSINFO** **GPHNAME**

Continued from previous page...

PLANC

```
BYTES : ProcessName(0:33)
INTEGER : ProcessNumber
...
Monitor_Call('GetOwnProcessInfo', ProcessName, ProcessNumber)
```

ASSEMBLY-500

```
ProcessName : STRINGARRAY 44B
ProcessNumber : W  BLOCK  1
GetOwnProcessInfo : EQU 37B9 + 427B
        ...
        CALLG GetOwnProcessInfo, 2, ProcessName, ProcessNumber
```

MAC

Not available.

ND-500 **All users** **All users**

GETRT — GetOwnRTAddress — 30B

Gets the address of the calling program's RT description. Background programs get the RT description address of the RT program which controls the terminal.

See also GetRTAddress, GetRTDescr, GetRTName, and @LIST-RT-PROGRAMS.

PARAMETERS

← The RT description address.

PASCAL

```
RTDescrAddress : INTEGER2;
...
GetOwnRTAddress(RTDescrAddress);
```

COBOL

```
01  RTDescrAddress  COMP.
    ...
    MONITOR-CALL "GetOwnRTAddress" USING RTDescrAddress.
```

FORTRAN

```
      INTEGER  RTDescrAddress
      ...
      Monitor_Call('GetOwnRTAddress', RTDescrAddress)
```

ND-100 and ND-500 | All users | All programs

30B GETOWNRTADDRESS GETRT

Continued from previous page...

PLANC

```
INTEGER : RTDescrAddress
...
Monitor_Call('GetOwnRTAddress', RTDescrAddress)
```

ASSEMBLY-500

```
RTDescrAddress : W  BLOCK  1
GetOwnRTAddress : EQU 37B9 + 30B
        ...
        CALLG GetOwnRTAddress, 0
        W1 =: RTDescrAddress            %Result is returned in W1 register.
```

MAC

```
        MON       30        %Monitor call GetOwnRTAddress.
        STA       RTPRO     %Store address of RT description.
        ...
RTPRO,  0
```

ND-100 and ND-500 All users All users

GPRNUM	GETPROCESSNO	426B

Gets the number of a process in the ND-500. You specify the process name. The process number is assigned when you start the ND-500-monitor.

- The process name may not be abbreviated.

See also GetOwnProcessInfo and SetProcessName.

PARAMETERS

→ Process name. You may include a user name, eg., (P-HANSEN)WP-PROCESS.
← Process number in bit 31:16 and magic number in bit 15:0.

PASCAL

```
ProcessName : PACKED ARRAY [0..33] OF CHAR;
ProcessNumber : LONGINT;
...
GetProcessNo(ProcessName, ProcessNumber);
```

COBOL

```
01  ProcessName    PIC X(34).
01  ProcessNumber  COMP.
    ...
    MONITOR-CALL "GetProcessNo" USING ProcessName, ProcessNumber.
```

FORTRAN

```
      CHARACTER  ProcessName*34
      INTEGER  ProcessNumber
      ...
      Monitor_Call('GetProcessNo', ProcessName(1:34), ProcessNumber)
```

ND-500	All users	All programs

426B **GETPROCESSNO** **GPRNUM**

Continued from previous page...

PLANC

```
BYTES : ProcessName(0:33)
INTEGER : ProcessNumber
...
Monitor_Call('GetProcessNo', ProcessName, ProcessNumber)
```

ASSEMBLY-500

```
ProcessName : STRINGARRAY  44B
ProcessNumber : W  BLOCK  1
GetProcessNo : EQU 37B9 + 426B
        ...
        CALLG GetProcessNo, 2, ProcessName, ProcessNumber
```

MAC

Not available.

ND-500 **All users** **All users**

GRTDA GetRTAddress 151B

Gets the address of an RT description. You specify the name of the RT program. See the SINTRAN III REAL TIME GUIDE (ND-60.133) for further information.

- In SINTRAN III, version J and older, this monitor call is only available to RT programs.

See also GetRTDescr, GetRTName, and @LIST-RT-PROGRAM.

PARAMETERS

→ RT program name.
← RT program address. Non-existing RT program names returns 0.

PASCAL

```
RTProgramName : PACKED ARRAY [0..5] OF CHAR;
RTProgram : INTEGER2;
...
GetRTAddress(RTProgramName, RTProgram);
```

COBOL

```
01  RTProgramName   PIC X(6).
01  RTProgram  COMP.
    ...
    MONITOR-CALL "GetRTAddress" USING RTProgramName, RTProgram.
```

FORTRAN

```
      CHARACTER  RTProgramName*6
      INTEGER  RTProgram
      ...
      Monitor_Call('GetRTAddress', RTProgramName(1:6), RTProgram)
```

ND-100 and ND-500 | All users | All programs

151B GETRTADDRESS GRTDA

Continued from previous page...

PLANC

```
BYTES : RTProgramName(0:5)
INTEGER : RTProgram
...
Monitor_Call('GetRTAddress', RTProgramName, RTProgram)
```

ASSEMBLY-500

```
RTProgramName : STRINGDATA  'SIB2A'''       %Get address of process SIB2A.
RTProgram : W  BLOCK  1
GetRTAddress : EQU 37B9 + 151B
        ...
        CALLG GetRTAddress, 1, RTProgramName
        IF K GO Error
        W1 =: RTProgram
        ...
Error,  ...
```

MAC

```
        LDA     (PAR        %Load register A with address of parameter list.
        MON     151         %Monitor call GetRTAddress.
        STA     RTPRO       %Store address of RT description.
        ...
RTPRO,  0
PAR,    RTNAM               %String containing name of RT program.
        ...
RTNAM,  'SIB2A'             %Get address of RT description of SIB2A.
```

ND-100 and ND-500 All users All users

RTDSC GetRTDescr 27B

Reads an RT description. The RT description contains various information about an RT program. You specify the RT program address. See the SINTRAN III REAL TIME GUIDE (ND-60.133) for further details.

- Use GetRTAddress if you only know the name of the RT program.
- This monitor call is only available to background programs in SINTRAN III VSX, version K.

See also GetOwnRTAddress and @LIST-RT-DESCRIPTION.

PARAMETERS

→ RT description address. Use 0 for the calling RT program.
← A 52 byte RT description.
← Number of devices connected to the RT program through StartOnInterrupt. Wrong RT description addresses return -1.

PASCAL

```
RTProgram, NoOfConnDev : INTEGER2;
RTDescriptor : ARRAY [0..1] OF BITMAP;
...
GetRTDescr(RTProgram, RTDescriptor, NoOfConnDev);
```

COBOL

```
01  RTProgram  COMP.
01  NoOfConnDev  COMP.
01  RTDescriptor.
    02  array COMP OCCURS 26 TIMES.
    ...
    MONITOR-CALL "GetRTDescr" USING RTProgram, RTDescriptor, NoOfConnDev.
```

FORTRAN

```
      INTEGER  RTProgram, NoOfConnDev
      INTEGER  RTDescriptor(26)
      ...
      Monitor_Call('GetRTDescr', RTProgram, RTDescriptor(1), NoOfConnDev)
```

ND-100 and ND-500 User RT and user SYSTEM RT programs

27B **GETRTDESCR** **RTDSC**

Continued from previous page...

PLANC

```
INTEGER : RTProgram, NoOfConnDev
BYTE ARRAY : RTDescriptor(0:51)
...
Monitor_Call('GetRTDescr', RTProgram, RTDescriptor(0), NoOfConnDev)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
NoOfConnDev : W  BLOCK  1
RTDescriptor : H  BLOCK  32B
GetRTDescr : EQU 37B9 + 27B
        ...
        CALLG GetRTDescr, 2, RTProgram, RTDescriptor
        IF K GO Error
        W1 =: NoOfConnDev               %Result is returned in W1 register.
        ...
Error,  ...
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     27      %Monitor call GetRTDescr.
        JAN     ERROR   %Handle error if register A is negative.
        STA     NODEV   %Store number of connected devices.
        ...
ERROR,  ...             %Handle this error, register A = -1.
        ...
NODEV,  0
PAR,    RTPRO           %Address of RT description.
        BUFF            %Buffer receiving RT descrition.
        ...
RTPRO,  ...
BUFF,   0
*+32B/                  %Make a 52 byte buffer.
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

GETNA **GETRTNAME** 152B

Gets the name of an RT program. You specify the RT description address.

- This monitor call is only available to background programs in SINTRAN III VSX, version K.

See also GetOwnRTAddress, GetRTAddress, GetRTDescr, @GET-RT-NAME, and @LIST-RT-Programs.

PARAMETERS

→ The address of the RT description. Use 0 for the calling program.
← Name of the RT program.

PASCAL

```
RTProgram : INTEGER2;
RTProgramName : PACKED ARRAY [0..5] OF CHAR;
...
GetRTName(RTProgram, RTProgramName);
```

COBOL

```
01  RTProgram  COMP.
01  RTProgramName   PIC X(6).
    ...
    MONITOR-CALL "GetRTName" USING RTProgram, RTProgramName.
```

FORTRAN

```
      INTEGER  RTProgram
      CHARACTER  RTProgramName*6
      ...
      Monitor_Call('GetRTName', RTProgram, RTProgramName(1:6))
```

ND-100 and ND-500 All programs

152B **GETRTNAME** **GETRTA**

Continued from previous page...

PLANC

```
INTEGER : RTProgram
BYTES : RTProgramName(0:5)
...
Monitor_Call('GetRTName', RTProgram, RTProgramName)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
RTProgramName : BY BLOCK 6
GetRTName : EQU 37B9 + 152B
        ...
        CALLG GetRTName, 2, RTProgram, RTProgramName
```

MAC

```
        LDA     (PAR      %Load register A with address of parameter list.
        MON     152       %Monitor call GetRTName.
        STT     RTNAM     %Store first 2 characters of RT name.
        STD     RTNAM+1   %Store last 4 characters.
        ...
PAR,    RTPRO             %Address of RT description.
        ...
RTPRO,  ...
RTNAM,  0
*+3/                      %Make a string of 6 bytes.
```

ND-100 and ND-500 **All users**

GSWSP GETSCRATCHSEGMENT 422B

Connects an empty data segment to your domain. The monitor call reserves space on the swap file for it.

- The segment is given the default name SCRATCH-SEGMENT:DSEG.

See also ClearCapability.

PARAMETERS

→ Segment size in bytes.
→ Logical segment number. The segment must be free. Use 0 to get the first free segment.
← Selected logical segment number if you specified 0 as logical segment number.

PASCAL

```
SizeInBytes, LogSegmentNo, RetLogSegmentNo : LONGINT;
...
GetScratchSegment(SizeInBytes, LogSegmentNo, RetLogSegmentNo);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  SizeInBytes  COMP.
01  LogSegmentNo  COMP.
01  RetLogSegmentNo  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetScratchSegment" USING SizeInBytes, LogSegmentNo,
                                           RetLogSegmentNo.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  SizeInBytes, LogSegmentNo, RetLogSegmentNo
      ...
      Monitor_Call('GetScratchSegment', SizeInBytes, LogSegmentNo,
     C                                  RetLogSegmentNo)
      IF (ErrCode .NE. 0) THEN ...
```

ND-500 All users All programs

422B **GETSCRATCHSEGMENT** GSVSP

Continued from previous page...

PLANC

```
INTEGER : SizeInBytes, LogSegmentNo, RetLogSegNo
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetScratchSegment', SizeInBytes, LogSegmentNo, RetLogSegNo)
```

ASSEMBLY-500

```
SizeInBytes : W  BLOCK  1
LogSegmentNo : W  BLOCK  1
RetLogSegmentNo : W  BLOCK  1
ErrCode : W  BLOCK  1
GetScratchSegment : EQU 37B9 + 422B
        ...
        CALLG GetScratchSegment, 3, SizeInBytes, LogSegmentNo,
RetLogSegmentNo
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

Not available.

ND-500 All users All users

RSEGM GetSegmentEntry 53B

Gets information about a segment in the ND-100. The monitor call returns the segment entry. You specify the segment number. See the SINTRAN III REAL TIME GUIDE (ND-60.133) for further information.

- Use GetSegmentNo to get a segment number from a segment name.

See also @LIST-SEGMENT and the SINTRAN-SERVICE-PROGRAM command DUMP-SEGMENT-TABLE-ENTRY.

PARAMETERS

→ Segment number. Use 0 for RT common. Only the returned segment link, the first physical page, and the flag are relevant to RT common.
← The 10 byte segment entry.
← Standard error code. See appendix A.

PASCAL

```
SegmentNumber : INTEGER2;
Buffer : BITMAP;
...
ReadSegmentEntry(SegmentNumber, Buffer);  [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  SegmentNumber  COMP.
01  Buffer.
    02  array COMP OCCURS 5 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetSegmentEntry" USING SegmentNumber, Buffer.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  SegmentNumber
      INTEGER  Buffer(5)
      ...
      Monitor_Call('GetSegmentEntry', SegmentNumber, Buffer(1))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | User RT and user SYSTEM | All programs

53B **GETSEGMENTENTRY** **BSBGM**

Continued from previous page...

PLANC

```
INTEGER : SegmentNumber
BYTE ARRAY : Buffer(0:9)
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
...
Monitor_Call('GetSegmentEntry', SegmentNumber, Buffer(0))
```

ASSEMBLY-500

```
SegmentNumber : W  BLOCK  1
Buffer : W  BLOCK 5
GetSegmentEntry : EQU 37B9 + 53B
        ...
        CALLG GetSegmentEntry, 2, SegmentNumber, Buffer
```

MAC

```
        LDA     (PAR      %Load register A with address of parameter list.
        MON     53        %Monitor call GetSegmentEntry.
        JMP     ERROR     %Handle error.
        ...               %Normal return.
ERROR,  ...
        ...
PAR,    SEGNO             %Segment number.
        BUFF              %Buffer for receiving a segment entry.
        ...
SEGNO,  ...
BUFF,   0
*+6/                      %Make a 6 word buffer.
        ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **All users**

GSGNO GetSegmentNo 322B

Gets the number of a segment in the ND-100. You specify the segment name. Segment names are created with the RT LOADER or when a program is dumped reentrant. See @DUMP-PROGRAM-REENTRANT.

See also GetSegmentEntry.

PARAMETERS

→ Segment name.
← Segment number. Nonexistent segment names return a negative value.

PASCAL

```
SegmentNumber : INTEGER2;
SegmentName : PACKED ARRAY [0..5] OF CHAR;
...
GetSegmentNo(SegmentName, SegmentNumber);
```

COBOL

```
01  SegmentNumber  COMP.
01  SegmentName   PIC X(6).
    ...
    MONITOR-CALL "GetSegmentNo" USING SegmentName, SegmentNumber.
```

FORTRAN

```
      INTEGER  SegmentNumber
      CHARACTER  SegmentName*6
      ...
      Monitor_Call('GetSegmentNo', SegmentName(1:6), SegmentNumber)
```

ND-100 and ND-500 | All users | All programs

322B **GETSEGMENTNO** **GSGNO**

Continued from previous page...

PLANC

```
INTEGER : SegmentNumber
BYTES : SegmentName(0:5)
...
Monitor_Call('GetSegmentNo', SegmentName, SegmentNumber)
```

ASSEMBLY-500

```
SegmentNumber : W  BLOCK  1
SegmentName : STRINGDATA  'EXSEG'''          %Get number of segment EXSEG.
ErrCode : W  BLOCK  1
GetSegmentNo : EQU 37B9 + 322B
        ...
        CALLG GetSegmentNo, 1, SegmentName
        IF K GO ERROR
        W1 =: SegmentNumber
        ...
ERROR : W1 =: ErrCode                        %ErrorCode in W1 register.
```

MAC

```
        LDA     (PAR     %Load register A with address of parameter list.
        MON     322      %Monitor call GetSegmentNo.
        JAN     ERROR    %Handle error if register A is negative.
        STA     SEGNO    %Store segment number.
        ...
ERROR,  ...              %No segment has EXSEG as name.
        ...
SEGNO,  0
PAR,    SEGNAM           %String containing segment name.
        ...
SEGNAM, 'EXSEG'          %Obtain segment number of EXSEG.
```

ND-100 and ND-500	All users	All users

MSIBB GetSIBASMessage 30SB

This is a special monitor call for SIBAS processes. It communicates with an application program. A message is received and processed. The application is restarted.

- The SIBAS process must have access to RT common.

See also SendSIBASMessage, SIBASFunction, and AnswerSIBAS.

PARAMETERS

→ The SIBAS number where the database runs.
→ Function. Use 0 to send an answer, restart the application program, and wait for the next message. Other values send an answer, restart the application and continue.
← Standard error code. See appendix A.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 | User RT and user SYSTEM | RT programs

305B GETSIBASMESSAGE MSIBD

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Not available.

MAC

```
        LDT     SIBNO   %SIBAS number.
        LDA     CTRL    %Control flag.
        MON     305     %Monitor call GetSIBASMessage.
        JMP     ERROR   %Error return from monitor call.
        ...             %Normal return.
ERROR,  ...             %Error number in register A.
        ...
SIBNO,  ...
CTRL,   ...
```

ND-100 User RT and user.SYSTEM RT programs

RSPQB GetSpoolingEntry 55B

Gets the next spooling queue entry, that is, the next file to be printed. The entry is removed from the spooling queue.

See also AppendSpooling, CloseSpoolingFile, and @LIST-SPOOLING-QUEUE.

PARAMETERS

→ Logical device number of the printer. See appendix B.
← The 272-byte spooling entry.
Byte 0:1 Number of printed copies.
2:3 If ASCII apostrophe, the file is printed independently of spooling conditions.
4:97 File name of spooling file.
98:255 Message to be output on the error device before printing.

← Standard error code. See appendix A.

PASCAL

```
SpoolDevNumber : INTEGER2;
Buffer : ARRAY [0..8] OF BITMAP;
...
GetSpoolingEntry(SpoolDevNumber, Buffer);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  SpoolDevNumber  COMP.
01  Buffer.
    02  array COMP QCCURS 136 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetSpoolingEntry" USING SpoolDevNumber, Buffer.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  SpoolDevNumber
      INTEGER  Buffer(136)
      ...
      Monitor_Call('GetSpoolingEntry', SpoolDevNumber, Buffer(1))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

55B **GETSPOOLINGENTRY** **RSPOR**

Continued from previous page...

PLANC

```
INTEGER : SpoolDevNumber
BYTE ARRAY : Buffer(0:271)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetSpoolingEntry', SpoolDevNumber, Buffer(0))
```

ASSEMBLY-500

```
SpoolDevNumber : W  BLOCK  1
Buffer : BY  BLOCK  1000B
ErrCode : W  BLOCK  1
GetSpoolingEntry : EQU 37B9 + 55B
        ...
        CALLG GetSpoolingEntry, 2, SpoolDevNumber, Buffer
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (BUFF    %Address of buffer for receiving queue entry.
        LDT     SPOOL    %Logical number of spooling device.
        MON     55       %Monitor call GetSpoolingEntry.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
SPOOL,  5                %Obtain spooling entry of line printer no. 1.
BUFF,   0
*+128/                   %Make a buffer of 256 bytes.
```

ND-100 and ND-500 **All users** **All users**

RBABT GETSTARTBYTE 75B

Gets the number of the next byte to access in a file. The bytes in a file are numbered from 0.

- The file must be opened for sequential access.
- You cannot use this monitor call for pheripheral files.

See also SetStartByte and SetStartBlock.

PARAMETERS

→ File number. See OpenFile.
← The number of the next byte to access.
← Standard error code. See appendix A.

PASCAL

```
FileNumber : INTEGER2;
BytePointer : LONGINT;
...
GetStartByte(FileNumber, BytePointer);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNumber  COMP.
01  BytePointer  COMP PIC S9(10).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetStartByte" USING FileNumber, BytePointer.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNumber
      INTEGER*4  BytePointer
      ...
      Monitor_Call('GetStartByte', FileNumber, BytePointer)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

75B GETSTARTBYTE REABT

Continued from previous page...

PLANC

```
INTEGER : FileNumber
INTEGER4 : BytePointer
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetStartByte', FileNumber, BytePointer)
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
BytePointer : W  BLOCK  1
ErrCode : W  BLOCK  1
GetStartByte : EQU 37B9 + 75B
        ...
        CALLG GetStartByte, 2, FileNumber, BytePointer
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     FILNO    %File number returned from earlier open.
        MON     75       %Monitor call GetStartByte.
        JMP     ERROR    %Error return from monitor call.
        STD     POINT    %Normal return, store byte pointer.
        ...
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
POINT,  0                %A double word.
        0                %
```

ND-100 and ND-500 All users All users

TPSTRA	GETSTOPINFO	407B

Stops the process and returns to the shadow process with 7 parameters. See the ND-500 LOADER/MONITOR (ND-60.136) for details.

PARAMETERS

← 7 parameters.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-500	All users	All programs

407B **GETSTOPINFO** **TPSTBA**

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
p1 : W  BLOCK  1
p2 : W  BLOCK  1
p3 : W  BLOCK  1
p4 : W  BLOCK  1
p5 : W  BLOCK  1
p6 : W  BLOCK  1
p7 : W  BLOCK  1
GetStopInfo :  EQU  37B9 + 407B
        ...
        CALLG GetStopInfo, 7, p1, p2, p3, p4, p5, p6, p7
```

MAC

Not available.

ND-500 **All users** **All users**

CPUST **GETSYSTEMINFO** 262B

Gets various system information. The system number, the CPU type, the SINTRAN III version, the instruction set, the patch indicator, and the system generation time are returned.

PARAMETERS

→ A number. It should always be 0.
← The 24 byte system information. The following bytes are used:

```
   0:1  System number, ie., normally the CPU number.
     2  Cpu type, 2 means ND-100 with 48 bits floating point instructions.
                  3 means ND-100 with 32 bits floating point instructions.
                  4 means ND-110 with 48 bits floating point instructions.
                  5 means ND-110 with 32 bits floating point instructions.
     3  Instruction set, 0 means ND-100 standard.
                         1 means ND-100 CE.
                         2 means ND-100 CX.
                         3 means ND-100 CX or ND-100 CX with 16 page tables.
     8  Operating system, 1 means SINTRAN III VSE.
                          2 means SINTRAN III VSE-500.
                          3 means SINTRAN III RTP.
                          4 means SINTRAN III VSX
                          5 means SINTRAN III VSX-500
     9  Operating system version, ASCII character A-Z without parity.
  12:13 Patch level indicator. System dependent coding.
  14:15 System generation time, minutes.
  16:17 System generation time, hours.
  18:19 System generation time, day.
  20:21 System generation time, month.
  22:23 System generation time, year.
```

← Standard error code. See appendix A.

PASCAL

```
Number : INTEGER2;
Buff : BITMAP;
...
GetSystemInfo(Number, Buff);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  Number  COMP.
01  Buff.
    02  array COMP OCCURS 12 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetSystemInfo" USING Number, Buff.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  Number
      INTEGER  Buff(12)
      ...
      Monitor_Call('GetSystemInfo', Number, Buff(1))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | ATT users | ATT programs

262B **GETSYSTEMINFO** **CPUST**

Continued from previous page...

PLANC

```
INTEGER : Number
BYTE ARRAY : Buff(0:23)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetSystemInfo', Number, Buff(0))
```

ASSEMBLY-500

```
Number : W  BLOCK  1
Buff : H  BLOCK 14B
ErrCode : W  BLOCK  1
GetSystemInfo : EQU 37B9 + 262B
        ...
        CALLG GetSystemInfo, 2, Number, Buff
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        SAA     0       %Load register A with zero.
        LDX     (BUFF   %Address of a 12 word long buffer.
        MON     262     %Monitor call GetSystemInfo.
        JMP     ERROR   %Error return from monitor call. (If A not 0.)
        ...             %Normal return.
ERROR,  ...             %Error number in register A.
        ...
BUFF,   0
*+14/                   %A 12 word long buffer.
```

ND-100 and ND-500 **All users** **All users**

GTMOD GetTerminalMode 306B

Gets the terminal mode. The terminal mode tells how the terminal function, ie., if all letters are converted to uppercase, if output stops when a full page is displayed, etc.

See also TerminalMode and @TERMINAL-MODE.

PARAMETERS

→ The logical device number of the terminal. See appendix B.
← The terminal mode. The numbers below are used.

Terminal mode	Capital letters?	Delay after return?	Stop on full page?	Logout on missing carrier?
0	No	No	No	No
1	Yes	No	No	No
2	No	Yes	No	No
3	Yes	Yes	No	No
4	No	No	Yes	No
5	Yes	No	Yes	No
6	No	Yes	Yes	No
7	Yes	Yes	Yes	No
8	No	No	No	Yes
9	Yes	No	No	Yes
10	No	Yes	No	Yes
11	Yes	Yes	No	Yes
12	No	No	Yes	Yes
13	Yes	No	Yes	Yes
14	No	Yes	Yes	Yes
15	Yes	Yes	Yes	Yes

← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, TerminalMode : INTEGER2;
...
GetTermMode(DeviceNumber, TerminalMode);   [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  TerminalMode  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetTerminalMode" USING DeviceNumber, TerminalMode.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, TerminalMode
      ...
      Monitor_Call('GetTerminalMode', DeviceNumber, TerminalMode)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 All users All programs

306B GETTERMINALMODE GTMOD

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, TerminalMode
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetTerminalMode', DeviceNumber, TerminalMode)
```

ASSEMBLY-500

Not available.

MAC

```
        LDT     DEVNO    %Logical device number.
        MON     306      %Monitor call GetTerminalMode.
        JMP     ERROR    %Error return from monitor call.
        STA     TMODE    %Normal return, store terminal mode number.
        ...
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
TMODE,  0
```

ND-100 and ND-500 | All users | All users

MGTTY	GETTERMINALTYPE	16B

Gets the terminal type. The terminal type tells SINTRAN III how to handle a particular terminal. A wrong terminal type normally distorts the screen. The function-keys cannot be used.

- Appendix H lists the terminal types.

See also SetTerminalType, and @GET-TERMINAL-TYPE.

PARAMETERS

→ The logical device number of the terminal. You may use 1 for your own terminal in background programs. You may specify TADs.
← The terminal type.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, TerminalType : INTEGER2;
...
GetTerminalType(DeviceNumber, TerminalType);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  TerminalType  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetTerminalType" USING DeviceNumber, TerminalType.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, TerminalType
      ...
      Monitor_Call('GetTerminalType', DeviceNumber, TerminalType)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500	All users	All programs

16B **GETTERMINALTYPE** **MONITY**

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, TerminalType
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetTerminalType', DeviceNumber, TerminalType)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
TerminalType : W  BLOCK  1
ErrCode : W  BLOCK  1
GetTerminalType : EQU 37B9 + 16B
        ...
        CALLG GetTerminalType, 2, DeviceNumber, TerminalType
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Logical device number, must be a terminal.
        MON     16       %Monitor call GetTerminalType.
        JMP     ERROR    %Error return from monitor call.
        STA     TYPE     %Normal return, store terminal type number.
        ...
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
TYPE,   0
```

ND-100 and ND-500 **All users** **All users**

TUSED GetTimeUsed 114B

Gets the time you have used the CPU since you logged in. In batch jobs, you get the time since you entered the job.

- The CPU time used is given in basic time units. A basic time unit is 1/50th of a second.

See also @TIME-USED.

PARAMETERS

← The CPU time used.

PASCAL

```
TimeUsed : LONGINT;
...
GetTimeUsed(TimeUsed);
```

COBOL

```
01  TimeUsed  COMP PIC S9(10).
    ...
    MONITOR-CALL "GetTimeUsed" USING TimeUsed.
```

FORTRAN

```
      INTEGER*4  TimeUsed
      ...
      Monitor_Call('GetTimeUsed', TimeUsed)
```

ND-100 and ND-500 | All users | Background programs

114B GETTIMEUSED TUSED

Continued from previous page...

PLANC

```
INTEGER4 : TimeUsed
...
Monitor_Call('GetTimeUsed', TimeUsed)
```

ASSEMBLY-500

```
TimeUsed : W  BLOCK  1
GetTimeUsed : EQU 37B9 + 114B
        ...
        CALLG GetTimeUsed, 0
        W1 =: TimeUsed              %Result is returned in W1 register.
```

MAC

```
        MON       114       %Monitor call GetTimeUsed.
        STD       TIME      %Store CPU time.
        ...
TIME,   0                   %A double word.
        0                   %
```

ND-100 and ND-500 | All users | Background programs

GETERCOD **GETTRAPREASON** 505B

Gets the error code from the swapper process. This is only relevant to programmed trap handlers. The swapper process starts the trap handler when it detects a fatal error, eg., address outside segment. Use this monitor call to get the error code.

- The error code may be a file system error or a swapper error.
- You clear the error code when you read it.

PARAMETERS

← Error code.

PASCAL

```
ErrorCode : LONGINT;
...
GetTrapReason(ErrorCode);
```

COBOL

```
01  ErrorCode  COMP.
    ...
    MONITOR-CALL "GetTrapReason" USING ErrorCode.
```

FORTRAN

```
      INTEGER  ErrorCode
      ...
      Monitor_Call('GetTrapReason', ErrorCode)
```

ND-500 All users All programs

505B **GETTRAPREASON** GETTRAPCOD

Continued from previous page...

PLANC

```
INTEGER : ErrorCode
...
Monitor_Call('GetTrapReason', ErrorCode)
```

ASSEMBLY-500

```
ErrorCode : W  BLOCK  1
GetTrapReason : EQU 37B9 + 505B
        ...
        CALLG GetTrapReason, 1, ErrorCode
```

MAC

Not available.

ND-500 All users All users

RUSER GetUserEntry 44B

Gets information about a user. The user entry in the directory is returned. It contains the user name, default file accesses, the pages in use, the password, the table of friends, and more.

- Only user RT and user SYSTEM may read the user entries of other users.

See also GetUserName and @DUMP-USER-ENTRY.

PARAMETERS

→ User name. It may include a directory name, eg., PACK-ONE:P-HANSEN.
← The 64-byte user entry. See appendix C.
← Standard error code. See appendix A.

PASCAL

```
UserName : PACKED ARRAY [0..63] OF CHAR;
Buff : ARRAY [0..1] OF BITMAP;
...
ReadUserEntry(UserName, Buff);     [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  UserName   PIC X(64).
01  Buff.
    02  array COMP OCCURS 32 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetUserEntry" USING UserName, Buff.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      CHARACTER  UserName*64
      INTEGER  Buff(32)
      ...
      Monitor_Call('GetUserEntry', UserName(1:64), Buff(1))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

44B **GETUSERENTRY** **BUSER**

Continued from previous page...

PLANC

```
BYTES : UserName(0:63)
BYTE ARRAY : Buff(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetUserEntry', UserName, Buff(0))
```

ASSEMBLY-500

```
UserName : STRINGDATA  'A-HANSEN'''          %Get entry of user A-HANSEN.
Buff : H  BLOCK  40B
ErrCode : W  BLOCK  1
GetUserEntry : EQU 37B9 + 44B
        ...
        CALLG GetUserEntry, 2, UserName, Buff
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                        %ErrorCode in W1 register.
```

MAC

```
        LDA     (BUFF    %Address for receiving user entry.
        LDX     (USER    %Address of string containing user name.
        MON     44       %Monitor call GetUserEntry.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
USER,   'A-HANSEN'       %Obtain user entry for A-HANSEN.
BUFF,   0
*+40/                    %Make a buffer of 32 words.
```

ND-100 and ND-500 **All users** **All users**

GUSMA **GETUSERNAME** 214B

Gets the name of the user executing the program. The program may be executed by a user on a remote computer if the COSMOS network is installed. The remote system name is then returned.

- RT programs return the name of user RT.

See also GetUserEntry and ExecutionInfo.

PARAMETERS

← User name.
→ Directory index.
→ User index.
← Remote flag. Set to 0 for a user on the local computer and 1 for a user on a remote computer. Not used by ND-500.
← Remote system identification if remote flag is 1. Not used by ND-500.
← Standard error code. See appendix A.

PASCAL

```
DirectoryIndex, UserIndex : INTEGER2;
UserName : PACKED ARRAY [0..15] OF CHAR;
...
FindUserName(UserName, DirectoryIndex, UserIndex);    [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DirectoryIndex  COMP.
01  UserIndex  COMP.
01  UserName   PIC X(16).
01  RemoteFlag COMP.
01  RemoteSystem   PIC X(64).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "GetUserName" USING UserName, DirectoryIndex, UserIndex,
                 RemoteFlag, RemoteName.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DirIndex, UserIndex, RemoteFlag
      CHARACTER  UserName*16, SystemName*64
      ...
      Monitor_Call('GetUserName', UserName(1:16), DirIndex, UserIndex,
     C                  RemoteFlag, RemoteSystem(1:64))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 **All users** **All programs**

214B **GETUSERNAME** **GUSNA**

Continued from previous page...

PLANC

```
INTEGER : DirectoryIndex, UserIndex, RemoteFlag
BYTES : UserName(0:15), SystemName(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('GetUserName', UserName, DirectoryIndex, UserIndex,&
                RemoteFlag, RemoteSystem)
```

ASSEMBLY-500

```
DirectoryIndex : W  BLOCK  1
UserIndex : W  BLOCK  1
UserName : STRINGARRAY 16
SysId : STRINGARRAY 16      %Optional parameter for remote system.
ErrCode : W  BLOCK  1
GetUserName : EQU 37B9 + 214B
        ...
        CALLG GetUserName, 3, UserName, DirectoryIndex, UserIndex
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDA     (USER    %Address of string to receive user name.
        LDX     INDEX    %Left byte: Dir index. Right byte: User index.
        LDT     (REMID   %Remote identification if bit 15 in register X.
        MON     214      %Monitor call GetUserName.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
USER,   0                %A string of 16 characters.
*+10/
INDEX,  ...              %Set bit 15 if remote user.
REMID,  0                %Remote system identification string.
*+32/
```

ND-100 and ND-500 **All users** **All users**

PACKET **GETUSERPARAM** 57B

Gets information about why the last program terminated. There are 5 parameters for each background user. These can be set by SINTRAN III or your background program.

- Use SetUserParam to set the parameter values.
- SINTRAN III sets some of the parameter values if you give the command @ENABLE-TERMINATION-HANDLING first.

See also TerminationHandling, GetND500Param, @DEFINE-TERMINATION-HANDLING, @DISABLE-TERMINATION-HANDLING, and @SET-USER-PARAMETERS.

PARAMETERS

← The five user parameters. This is an array of 16-bit integers on the ND-100. ND-500 returns an array of 32-bit integers. SINTRAN III's termination handling returns the following:

Parameter 1: The last byte contains the user index. The byte in front of it contains the directory index.
2: Logical device number of the terminal.
3: Fatal error or the monitor call ErrorMessage returns the error number. If escape was pressed, -1 is returned.
4: User defined.
5: User defined.

The parameters are returned if the user presses the ESCAPE key, if the monitor calls ExitFromProgram or ErrorMessage are executed, or if a fatal error occurs. All parameters can be user defined if no termination handling is enabled.

PASCAL

```
Buff : BITMAP;
...
GetUserParam(Buff);
```

COBOL

```
01  Buff.
    02  array COMP OCCURS 5 TIMES.
    ...
    MONITOR-CALL "GetUserParam" USING Buff.
```

FORTRAN

```
      INTEGER  Buff(5)
      ...
      Monitor_Call('GetUserParam', Buff(1))
```

ND-100 and ND-500 | All users | Background programs

57B **GETUSERPARAM** PAGE1

Continued from previous page...

PLANC

```
BYTE ARRAY : Buff(0:9)
...
Monitor_Call('GetUserParam', Buff(0))
```

ASSEMBLY-500

```
Buff : H  BLOCK 5
GetUserParam : EQU 37B9 + 57B
        ...
        CALLG GetUserParam, 1, Buff
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     57      %Monitor call GetUserParam.
        ...
PAR,    BUFF            %Buffer of 5 words to receive user parameters.
        ...
BUFF,   0               %Left byte: dir index. Right byte: user index.
        0               %Logical device number, terminal number.
        0               % -1 if escape, otherwise error number.
        0               %User defined.
        0               %User defined.
```

ND-100 and ND-500 | All users | Background programs

GRBLK GetUserRegisters 420B

SwitchUserBreak allows you to save the registers when you terminate an ND-500 program with the ESCAPE key. You can get the contents of the registers with GetUserRegisters.

- 39 registers are saved.

See also SwitchUserBreak.

PARAMETERS

← 154 bytes containing the registers in their number sequence.

PASCAL

```
Buffer : ARRAY [0..4] OF BITMAP;
...
GetUserRegister(Buffer);
```

COBOL

```
01  Buffer.
    02  array COMP OCCURS 77 TIMES.
    ...
    MONITOR-CALL "GetUserRegister" USING Buffer.
```

FORTRAN

```
      INTEGER  Buffer(77)
      ...
      Monitor_Call('GetUserRegister', Buffer(1))
```

ND-500 All users All programs

420B	GETUSERREGISTERS	GBBLK

Continued from previous page...

PLANC

```
BYTE ARRAY : Buffer(0:153)
...
Monitor_Call('GetUserRegister', Buffer(0))
```

ASSEMBLY-500

```
Buffer : W  BLOCK 50B
GetUserRegister : EQU 37B9 + 420B
        ...
        CALLG GetUserRegister, 1, Buffer
```

MAC

Not available.

ND-500	All users	All users

GRAPHIC — GRAPHICFUNCTION — 155B

Executes various functions on a graphic peripheral, such as a NORDCOM terminal, a pen plotter, or a Textronix display.

- Some functions require a delay from the calling program. See the specifications for the peripheral in use. SuspendProgram delays a program.
- The old monitor call PLOT is obsolete.

PARAMETERS

→ The Y-coordinate of new line relative to current reference point.
→ The X-coordinate of new line relative to current reference point.
→ Integer code.
→ Logical device number.
→ Function code. 0 means PLOT. 1 means PLOTS, ie., establish reference point and/or clear a NORDCOM screen. 2 means NEWP, ie.,select pen or screen.
← Return value. Output parameter for the PLOT function. Not used on the ND-500.

PASCAL

```
Xcoor, Ycoor, Code, DeviceNo, Func, ReturnValue : INTEGER2;
...
GraphicFunction(Xcoor, Ycoor, Code, DeviceNo, Func, ReturnValue);
```

COBOL

```
01  Xcoor  COMP.
01  Ycoor  COMP.
01  Code  COMP.
01  DeviceNo  COMP.
01  Func  COMP.
01  ReturnValue  COMP.
    ...
    MONITOR-CALL "GraphicFunction" USING Xcoor, Ycoor, Code, DeviceNo,
                                         Func, ReturnValue.
```

FORTRAN

```
      INTEGER  Xcoor, Ycoor, Code, DeviceNo, Func, ReturnValue
      ...
      Monitor_Call('GraphicFunction', Xcoor, Ycoor, Code, DeviceNo,
     C                                Func, ReturnValue)
```

ND-100 and ND-500 | All users | All programs

155B GRAPHICFUNCTION GRAPHIC

Continued from previous page...

PLANC

```
INTEGER : Xcoor, Ycoor
INTEGER : Code, DeviceNo, Func, ReturnValue
...
Monitor_Call('GraphicFunction', Xcoor, Ycoor, Code, DeviceNo, &
                                Func, ReturnValue)
```

ASSEMBLY-500

```
Xcoor : W  BLOCK  1
Ycoor : W  BLOCK  1
Code : W  BLOCK  1
DeviceNo : W  BLOCK  1
Func : W  BLOCK  1
GraphicFunction : EQU 37B9 + 155B
        ...
        CALLG GraphicFunction, 5, Xcoor, Ycoor, Code, DeviceNo, Func
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     155     %Monitor call Graphic.
        STA     STAT    %Store return status.
        ...
STAT,   0
PAR,    XCOOR           % X coordinate of endpoint.
        YCOOR           % Y coordinate of endpoint.
        CODE            %Integer code.
        DEVNO           %Logical device number.
        FUNC            %Function.
        ...
XCOOR,  ...
YCOOR,  ...
CODE,   ...
DEVNO,  ...
FUNC,   ...
```

ND-100 and ND-500 All users All users

MHDLC HDLCFUNCTION 2018

Performs various HDLC functions. A HDLC is a link to another computer. You may send data, receive data, and control HDLC. Data is sent as driver control blocks. See the SINTRAN III COMMUNICATION GUIDE (ND-60.134) for details.

PARAMETERS

→ Function code. Use 0 to send and 1 to receive.
→ Logical device number. There are different logical device numbers for the input and output part. See appendix B.
→ Address of driver control block.
↔ Size of the used part of the driver control block in bytes.
↔ Depends on the function code. Maximum size of the driver control block in bytes for the send function. Wait flag for the receive function. The program waits for a driver control block if 1. Use 0 to make the program continue.
← HDLC error code.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 and ND-500 | All users | All programs

201B HDLCFUNCTION HDLC

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
Func :  W  BLOCK  1
DevNo : W  BLOCK  1
Buffer : W  BLOCK  1024   %Even byte address.
USize : W  BLOCK  1
MSize : W  BLOCK  1
Status : W  BLOCK  1
HDLCFunction : EQU 37B9 + 201B
        ...
        CALLG HDLCFunction, 5, Func, DevNo, Buffer, USize, MSize,
        IF K GO Error
        W1 =: Status
        ...
Error,
```

MAC

```
        MON      201      %Monitor call HDLCFunction.
```

ND-100 and ND-500 | All users | All users

B4IRW In4x2Bytes 63B

Reads 8 bytes from a word-oriented or character-oriented device, eg., internal devices.

- Do not use this monitor call for terminals.
- This monitor call was mainly used for SIBAS communication via ND-NET. It is now seldom used.

See also In8Bytes, InUpTo8Bytes, In8AndFlag, InString, InputString, InByte, and Out8Bytes.

PARAMETERS

→ Logical device number. See appendix B.
← Number of bytes read.
← The string of bytes read.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, NoOfBytes : INTEGER2;
DataRead : PACKED ARRAY [0..7] OF CHAR;
...
In4x2Bytes(DeviceNumber, NoOfBytes, DataRead);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  NoOfBytes  COMP.
01  DataRead   PIC X(8).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "In4x2Bytes" USING DeviceNumber, NoOfBytes, DataRead.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, NoOfBytes
      CHARACTER  DataRead*8
      ...
      Monitor_Call('In4x2Bytes', DeviceNumber, NoOfBytes, DataRead(1:8))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

63B **In4x2Bytes** B41BW

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, NoOfBytes
BYTES : DataRead(0:7)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('In4x2Bytes', DeviceNumber, NoOfBytes, DataRead)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
NoOfBytes : W  BLOCK  1
DataRead : STRINGARRAY 8
ErrCode : W  BLOCK  1
In4x2Bytes : EQU 37B9 + 63B
        ...
        CALLG In4x2Bytes, 2, DeviceNumber, DataRead
        IF K GO ERROR
        W1 =: NoOfBytes
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Logical device number.
        MON     63       %Monitor call In4x2Bytes.
        JMP     ERROR    %Error return from monitor call.
        STD     BYTES    %Normal return, store first 4 bytes read.
        COPY    SL  DA
        STA     BYTES+2  %Store next 2 bytes read.
        STX     BYTES+3  %Store last 2 bytes read.
        STT     COUNT    %Store number of bytes read.
        ...
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
COUNT,  0
BYTES,  0
        0
        0
        0
```

ND-100 and ND-500 All users All users

T8IMB	IN8ANDFLAG	3108

Reads 8 bytes from a device, eg., a terminal. The monitor call applies to the defined echo and break setting. See SetEcho and SetBreak.

- Input of a break character stops the reading. The number of characters read are output. It is ouput as a negative number if a break character has been read. That is, bit 15 is set to 1.
- This monitor call can be used together with SetBreak and SetEcho.
- Appendix F contains an ASCII table.

See also In8Bytes, InUpTo8Bytes, InByte, InString, InputString, In4x2Bytes, and Out8Bytes.

PARAMETERS

→ Logical device number. See appendix B.
← Number of bytes read. A negative number means that a break character is read.
← The string of bytes read.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, NoOfBytes : INTEGER2;
Buffer : PACKED ARRAY [0..7] OF CHAR;
...
In8AndFlag(DeviceNumber, NoOfBytes, Buffer);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  NoOfBytes  COMP.
01  Buffer   PIC X(8).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "In8AndFlag" USING DeviceNumber, NoOfBytes, Buffer.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, NoOfBytes
      CHARACTER  Buffer*8
      ...
      Monitor_Call('In8AndFlag', DeviceNumber, NoOfBytes, Buffer(1:8))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500	All users	All programs

310B **IN8ANDFLAG** **T81NB**

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, NoOfBytes
BYTES : Buffer(0:7)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('In8AndFlag', DeviceNumber, NoOfBytes, Buffer)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
NoOfBytes : W  BLOCK  1
Buffer : STRINGARRAY 8
In8AndFlag : EQU 37B9 + 310B
        ...
        CALLG In8AndFlag, 3, DeviceNumber, NoOfBytes, Buffer
        IF K GO Error,
        ...
Error,  ...
```

MAC

```
        LDT     DEVNO     %Logical device number.
        MON     310       %Monitor call In8AndFlag.
        JMP     ERROR     %Error return.
        STD     BYTES     %Store first 4 bytes read.
        COPY    SL  DA
        STA     BYTES+2   %Store next 2 bytes read.
        STX     BYTES+3   %Store last 2 bytes read.
        STT     COUNT     %Store number of bytes read.
        ...
ERROR,  ...               %Error handling.
        ...
DEVNO,  ...
COUNT,  0
BYTES,  0
        0
        0
        0
```

ND-100 and ND-500 **All users** **All users**

B81MB — IN8BYTES — 23B

Reads 8 bytes from a device. The input is fast, but the monitor call does not apply the defined echo and break setting.

- Do not use this monitor call for terminals and TAD's when echo and break should be applied.
- Appendix F contains an ASCII table.

See also In8AndFlag, InUpTo8Bytes, InByte, InString, InputString, In4x2Bytes, and Out8Bytes.

PARAMETERS

→ Logical device number. See appendix B.
← Number of bytes read. If the monitor call is used on a word oriented device, the number of words is returned.
← The string of bytes read.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, NoOfBytes : INTEGER2;
DataRead : PACKED ARRAY [0..7] OF CHAR;
...
In8Bytes(DeviceNumber, NoOfBytes, DataRead);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  NoOfBytes  COMP.
01  DataRead   PIC X(8).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "In8Bytes" USING DeviceNumber, NoOfBytes, DataRead.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, NoOfBytes
      CHARACTER  DataRead*8
      ...
      Monitor_Call('In8Bytes', DeviceNumber, NoOfBytes, DataRead(1:8))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

23B In8Bytes B8INB

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, NoOfBytes
BYTES : DataRead(0:7)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('In8Bytes', DeviceNumber, NoOfBytes, DataRead)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
NoOfBytes : W  BLOCK  1
DataRead : STRINGARRAY 8
ErrCode : W  BLOCK  1
In8Bytes : EQU 37B9 + 23B
        ...
        CALLG In8Bytes, 3, DeviceNumber, NoOfBytes, DataRead
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Logical device number.
        MON     23       %Monitor call In8Bytes.
        JMP     ERROR    %Error return from monitor call.
        STD     BYTES    %Normal return, store first 4 bytes read.
        COPY    SL  DA
        STA     BYTES+2  %Store next 2 bytes read.
        STX     BYTES+3  %Store last 2 bytes read.
        STT     COUNT    %Store number of bytes (or words)  read.
        ...
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
COUNT,  0
BYTES,  0
        0
        0
        0
```

ND-100 and ND-500 | All users | All users

ISIZE — InBufferSpace — 6GB

Gets the current number of bytes in the input buffer. Terminals and other character devices place input in a buffer. All input monitor calls reads from this buffer.

- Use ExecutionInfo to get the logical device number for terminals. You can specify 1 for your own terminal.
- The buffer size depends on the device.

See also InBufferState, ClearInBuffer and OutBufferSpace.

PARAMETERS

→ Logical device number. See appendix B.
← Number of bytes in the buffer.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, NoOfBytes : INTEGER2;
...
BytesInBuffer(DeviceNumber, NoOfBytes);     [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  NoOfBytes  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "InBufferSpace" USING DeviceNumber, NoOfBytes.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, NoOfBytes
      ...
      Monitor_Call('InBufferSpace', DeviceNumber, NoOfBytes)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

66B InBufferSpace ISIZE

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, NoOfBytes
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('InBufferSpace', DeviceNumber, NoOfBytes)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
NoOfBytes : W  BLOCK  1
ErrCode : W  BLOCK  1
InBufferSpace : EQU 37B9 + 66B
        ...
        CALLG InBufferSpace, 1, DeviceNumber
        IF K GO ERROR
        W1 =: NoOfBytes                 %Result is returned in W1 register.
        ...
ERROR : W1 =: ErrCode                   %ErrorCode in W1 register.
```

MAC

```
        LDA     DEVNO    %Logical device number.
        MON     66       %Monitor call InBufferSpace.
        JMP     ERROR    %Error return from monitor call.
        STA     COUNT    %Normal return, store number of bytes in inbuffer.
        ...
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
COUNT,  0
```

ND-100 and ND-500 | All users | All users

IBBSIZ INBUFFERSTATE 313B

Gets information about an input buffer. The current number of bytes in it and the number of bytes until a break character are returned.

- Use ExecutionInfo to get the logical device number for terminals. You can specify 1 for your own terminal.

See also InBufferSpace, ClearInBuffer and OutBufferSpace.

PARAMETERS

→ Logical device number. See appendix B.
← Number of bytes in the buffer.
← Number of bytes before break.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, NoInBuffer, NoUntilBreak : INTEGER2;
...
InBufferState(DeviceNumber, NoInBuffer, NoUntilBreak);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNo  COMP.
01  NoInBuffer  COMP.
01  NoUntilBreak  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "InBufferState" USING DeviceNo, NoInBuffer, NoUntilBreak.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, NoInBuffer, NoUntilBreak
      ...
      Monitor_Call('InBufferState', DeviceNumber, NoInBuffer, NoUntilBreak)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

313B **InBufferState** IBMSIZ

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, NoInBuffer, NoUntilBreak
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('InBufferState', DeviceNumber, NoInBuffer, NoUntilBreak)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
NoInBuffer : W  BLOCK  1
NoUntilBreak : W  BLOCK  1
ErrCode : W  BLOCK  1
InBufferState : EQU 37B9 + 313B
        ...
        CALLG InBufferState, 2, DeviceNumber, NoUntilBreak
        IF K GO ERROR
        W1 =: NoOfBytes
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Logical device number.
        MON     313      %Monitor call InBufferState.
        JMP     ERROR    %Error return from monitor call.
        STA     COUNT    %Normal return, store number of bytes in inbuffer.
        STX     NOBRK    %Store number of bytes in inbuffer until break.
        ...
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
COUNT,  0
NOBRK,  0
```

ND-100 and ND-500 All users All users

INBT InByte 1B

Reads one byte from a character device, eg., a terminal or an opened file. If the device is a word-oriented device, one word is read. This monitor call can be used on most input devices.

- Bit 7 is a parity bit if terminal or file input. IOMultiFunction may change this.
- The program waits if there is no bytes in the input buffer of the device. You can change this with NoWaitSwitch or TerminalNoWait.
- The pointer to the next byte is incremented when you read from a mass storage file.
- Input from card readers are converted to ASCII characters. Use DeviceControl to read the 12-bit card columns.
- Background programs may read from logical device number 0. This is the SINTRAN III command buffer. You may read parameters following the program name this way. Break and echo are both set to 1. Normal SINTRAN III command editing is available. All letters are converted to uppercase. You may control this with IOMultiFunction.
- Appendix F contains an ASCII table.

See also In8AndFlag, InUpTo8Bytes, In8Bytes, InString, InputString, In4x2Bytes, and OutByte.

PARAMETERS

→ Logical device number. See appendix B. Use 1 for your own terminal.
← The read byte.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, ReturnValue : INTEGER2;
...
InByte(DeviceNumber, ReturnValue);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  ReturnValue  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "InByte" USING DeviceNumber, ReturnValue.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, ReturnValue
      ...
      Monitor_Call('InByte', DeviceNumber, ReturnValue)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 All users All programs

1B **INBYTE** INBT

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, ReturnValue
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('InByte', DeviceNumber, ReturnValue)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
ReturnValue : W  BLOCK  1
ErrCode : W  BLOCK  1
InByte : EQU 37B9 + 1B
        ...
        CALLG InByte, 2, DeviceNumber, ReturnValue
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Logical device number.
        MON     1        %Monitor call InByte.
        JMP     ERROR    %Error return from monitor call.
        STA     BYTE     %Normal return, store byte read.
        ...
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
BYTE,   0
```

ND-100 and ND-500 All users All users

DVTEST **INPUTSTRING** 503B

Reads a string from a device, eg., a terminal or an opened file. This monitor call provide a fast input to ND-500 programs.

- Only the first four parameters are used for file input.
- Appendix F contains an ASCII table.

See also InByte, InString, and OutputString.

PARAMETERS

→ Logical device number. See appendix B. Use 1 for your own terminal. The open file number is obtained by opening a file and specifying connection number 0. See the ND-500 LOADER/MONITOR (ND-60.136).
→ Maximum number of bytes to read.
← Number of bytes read.
→ Buffer to receive input.
→ Break setting. See SetBreak. Use 8 not to change the break setting.
→ Echo setting. See SetEcho. Use 8 to change the echo setting.
→ Break table, bit 0:31. Bits set to 1 cause break on the character.
→ Break table, bit 32:63.
→ Break table, bit 64:95.
→ Break table, bit 96:127.
→ Echo table, bit 0:31. Bits set to 0 cause echo on the character.
→ Echo table, bit 32:63.
→ Echo table, bit 64:95.
→ Echo table, bit 96:127.

PASCAL

```
DevNo, MaxNo, NoOfBytesRet, BreakStrat, EchoStrat, BreakT1 : LONGINT;
BreakT2, BreakT3, BreakT4, EchoT1, EchoT2, EchoT3, EchoT4 : LONGINT;
Buff : ARRAY [0..8] OF BITMAP;
...
InputString(DevNo, MaxNo, NoOfBytesRet, Buff, BreakStrat, EchoStrat,
        BreakT1, BreakT2, BreakT3, BreakT4, EchoT1, EchoT2, EchoT3, EchoT4);
```

COBOL

```
01  DevNo  COMP.  01  MaxNo  COMP.  01  NoOfBytesRet  COMP.
01  BreakStrat  COMP.  01  EchoStrat  COMP.
01  BreakT1  COMP. 01  BreakT2  COMP. 01  BreakT3  COMP. 01  BreakT4  COMP.
01  EchoT1  COMP. 01  EchoT2  COMP. 01  EchoT3  COMP. 01  EchoT4  COMP.
01  Buff.
    02  array COMP OCCURS 100 TIMES.
    ...
    MONITOR-CALL "InputString" USING DevNo, MaxNo, NoOfBytesRet, Buff,
                 BreakStrat, EchoStrat, BreakT1, BreakT2, BreakT3,
                 BreakT4, EchoT1, EchoT2, EchoT3, EchoT4.
```

FORTRAN

```
      INTEGER  DevNo, MaxNo, NoOfBytesRet, BreakStrat, EchoStrat, BreakT1
      INTEGER  BreakT2, BreakT3, BreakT4, EchoT1, EchoT2, EchoT3, EchoT4
      INTEGER  Buff(100)
      ...
      Monitor_Call('InputString', DevNo, MaxNo, NoOfBytesRet, Buff(1),
     C                BreakStrat, EchoStrat, BreakT1, BreakT2, BreakT3,
     C                BreakT4, EchoT1, EchoT2, EchoT3, EchoT4)
```

ND-500 All users All programs

503B **INPUTSTRING** DVINST

Continued from previous page...

PLANC

```
INTEGER : DevNo, MaxNo, NoOfBytesRet, BreakStrat, EchoStrat, BreakT1
INTEGER : BreakT2, BreakT3, BreakT4, EchoT1, EchoT2, EchoT3, EchoT4
BYTE ARRAY : Buff(0:199)
...
Monitor_Call('InputString', DevNo, MaxNo, NoOfBytesRet, Buff(0), &
                BreakStrat, EchoStrat, BreakT1, BreakT2, BreakT3, &
                BreakT4, EchoT1, EchoT2, EchoT3, EchoT4)
```

ASSEMBLY-500

```
DevNo : W  BLOCK  1
MaxNo : W  BLOCK  1
NoOfBytesRet : W  BLOCK  1
BreakStrat : W  BLOCK  1
EchoStrat : W  BLOCK  1
BreakT1 : W  BLOCK  1
BreakT2 : W  BLOCK  1
BreakT3 : W  BLOCK  1
BreakT4 : W  BLOCK  1
EchoT1 : W  BLOCK  1
EchoT2 : W  BLOCK  1
EchoT3 : W  BLOCK  1
EchoT4 : W  BLOCK  1
Buff : BY  BLOCK 400B
InputString : EQU 37B9 + 503B
        ...
        CALLG InputString, 14, DevNo, MaxNo, NoOfBytesRet, Buff, &
                               BreakStrat, EchoStrat, BreakT1, BreakT2, &
                               BreakT3, BreakT4, EchoT1, EchoT2, EchoT3, EchoT4
```

MAC

Not available.

ND-500 All users All users

IMSTR — InString — 161B

Reads a string of characters from a peripheral device, eg., a terminal.

- Background programs wait if there is no characters in the device buffer. RT programs continue.
- All parameters are fetched and returned via the alternative page table.
- You are adviced to use the faster InputString on the ND-500.
- Appendix F contains an ASCII table.

See also InputString, In8AndFlag, InUpTo8Bytes, In8Bytes, In4x2Bytes, and OutString.

PARAMETERS

→ Logical device number of a peripheral device. See appendix B.
← String of characters.
→ Maximum number of characters to be read.
→ Terminating character. Input stops when this character is read.
← A 16-bit integer status. Errors in parameters return -1. If bit 15:14 is 0, the maximum number of characters is read. If 1, a termination character is read. 2 means not terminated. This applies only to RT program. 3 means device error. Bit 7:0 contains the error number. If bit 15:14 are 0, 1 or 2, the number of characters read is returned in bit 13:0.

PASCAL

```
DeviceNo, NoOfBytes, Terminator, ReturnStatus : INTEGER2;
TextRead : PACKED ARRAY [0..255] OF CHAR;
...
InString(DeviceNo, TextRead, NoOfBytes, Terminator, ReturnStatus);
```

COBOL

```
01  DeviceNo  COMP.
01  TextRead   PIC X(256).
01  NoOfBytes  COMP.
01  Terminator  COMP.
01  ReturnStatus  COMP.
    ...
    MONITOR-CALL "InString" USING DeviceNo, TextRead, NoOfBytes,
                                  Terminator, ReturnStatus.
```

FORTRAN

```
      INTEGER  DeviceNo, NoOfBytes, Terminator, ReturnStatus
      CHARACTER  TextRead*256
      ...
      Monitor_Call('InString', DeviceNo, TextRead(1:256), NoOfBytes,
     C                         Terminator, ReturnStatus)
```

ND-100 and ND-500 | All users | All programs

161B **INSTRING** **INSTR**

Continued from previous page...

PLANC

```
INTEGER : DeviceNo, NoOfBytes, Terminator, ReturnStatus
BYTES : TextRead(0:255)
...
Monitor_Call('InString', DeviceNo, TextRead, NoOfBytes, &
                         Terminator, ReturnStatus)
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
TextRead : STRINGARRAY 4000B
NoOfBytes : W  BLOCK  1
Terminator : W  BLOCK  1
ReturnStatus : W  BLOCK  1
InString : EQU 37B9 + 161B
        ...
        CALLG InString, 4, DeviceNo, TextRead, NoOfBytes, Terminator
        IF K GO Error
        W1 =: ReturnStatus            %Status is returned in W1 register.
        ...
Error,  ...
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     161     %Monitor call InString.
        STA     STAT    %Store status returned.
        ...
STAT,   0
PAR,    DEVNO           %Logical device number.
        TEXT            %String read.
        COUNT           %Maximum number of characters to be read.
        TERM            %Terminator character.
        ...
DEVNO,  ...
TEXT,   0
*+50/                   %Make a string of 80 characters.
COUNT,  120             %To read maximum 80 characters.
TERM,   @'              %Terminate reading when @ is read.
```

ND-100 and ND-500 **All users** **All users**

RBIRB InUpTo8Bytes 21B

Reads up to 8 bytes from a device, eg., a terminal. The monitor call applies the defined echo and break setting.

- This monitor call can only be used for terminals, internal devices, TADs, HDLCs, and synchronous modems.
- You are adviced to use In8AndFlag instead of this monitor call.

See also In8Bytes, InByte, InString, In4x2Bytes, and Out8Bytes.

PARAMETERS

→ Logical device number. See appendix B.
← Number of bytes read. If you use the monitor call on a word-oriented device, it returns the number of words read.
← The 8 bytes.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, NoOfBytes : INTEGER2;
InData : BITMAP;
...
InUpTo8Bytes(DeviceNumber, NoOfBytes, InData);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  NoOfBytes  COMP.
01  InData   PIC X(8).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "InUpTo8Bytes" USING DeviceNumber, NoOfBytes, InData.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, NoOfBytes
      CHARACTER  InData*8
      ...
      Monitor_Call('InUpTo8Bytes', DeviceNumber, NoOfBytes, InData(1:8))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

21B **InUpTo8Bytes** MSINB

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, NoOfBytes
BYTES : InData(0:7)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('InUpTo8Bytes', DeviceNumber, NoOfBytes, InData)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
NoOfBytes : W  BLOCK  1
InData : STRINGARRAY 8
ErrCode : W  BLOCK  1
InUpTo8Bytes : EQU 37B9 + 21B
        ...
        CALLG InUpTo8Bytes, 3, DeviceNumber, NoOfBytes, InData
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Logical device number.
        MON     21       %Monitor call InUpTo8Bytes.
        JMP     ERROR    %Error return from monitor call.
        STD     BYTES    %Normal return, store first 4 bytes read.
        COPY    SL  DA
        STA     BYTES+2  %Store next 2 bytes read.
        STX     BYTES+3  %Store last 2 bytes read.
        STT     COUNT    %Store number of bytes (or words) read.
        ...
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
COUNT,  0
BYTES,  0
        0
        0
        0
```

ND-100 and ND-500 All users All users

IOIX IOINSTRUCTION 31B

Executes an IOX machine instruction. The IOX instruction handles the device registers. The IOX instruction must be inserted in the IOX table by the SINTRAN-SERVICE-PROGRAM command INSERT-IN-IOX-TABLE first.

- SINTRAN III must know the device register addresses.
- This monitor call may be used in debugging of device interfaces.

See also PrivInstruction, CamacIOInstruction and @EXECUTE-IOX.

PARAMETERS

→ Register contents before execution.
→ Device register address.
← Register contents after execution.

PASCAL

```
RegContents, DevRegAddr, ContentsAfter : INTEGER2;
...
IOInstruction(RegContents, DevRegAddr, ContentsAfter);
```

COBOL

```
01  RegContent  COMP.
01  DevRegAddr  COMP.
01  ContentAfter  COMP.
    ...
    MONITOR-CALL "IOInstruction" USING RegContent, DevRegAddr, ContentAfter.
```

FORTRAN

```
      INTEGER  RegContents, DevRegAddr, ContentAfter
      ...
      Monitor_Call('IOInstruction', RegContents, DevRegAddr, ContentAfter)
```

ND-100 and ND-500 | User RT and user SYSTEM | ATT programs

31B IOINSTRUCTION EXIOX

Continued from previous page...

PLANC

```
INTEGER : RegContents, DevRegAddr, ContentsAfter
...
Monitor_Call('IOInstruction', RegContents, DevRegAddr, ContentsAfter)
```

ASSEMBLY-500

```
RegContents : W  BLOCK  1
DevRegAddr : W  BLOCK  1
ContentsAfter : W  BLOCK  1
IOInstruction : EQU 37B9 + 31B
        ...
        CALLG IOInstruction, 2, RegContents, DevRegAddr
        IF K GO error
        W1 =: ContentsAfter          %Result is returned in W1 register.
        ...
Error,  ...
```

MAC

```
        LDA     (PAR     %Load register A with address of parameter list.
        MON     31       %Monitor call IOInstruction.
        STA     STAT     %Store status returned.
        ...
STAT,   0
PAR,    REG              %Register contents.
        DEV              %Device register address.
        ...
REG,    ...
DEV,    ...
```

ND-100 and ND-500 | User RT and user SYSTEM | All users

NCALL JUMPTOSEGMENT 132B

Calls a routine on another segment in the ND-100. You can divide an ND-100 RT program on various segments. This monitor call switches one or both of the current segments.

- Use ExitFromSegment to return from the routine.
- JumpToSegment may be nested.
- This monitor call do not change reentrant segments.

See also ExitFromSegment and ChangeSegment.

PARAMETERS

→ Address of routine.
→ New segments. The most significant byte identifies the first segment. The least significant byte identifies the second segment. Use 377B as segment number if you do not want to change it. The RT description identifies the segments which are current.

PASCAL

```
SubroutineAddr, NewSegment : INTEGER2;
...
JumpToSegment(SubroutineAddr, NewSegment);
```

COBOL

Not available.

FORTRAN

Not available.

ND-100 User RT and user SYSTEM RT programs

132B JUMPTOSEGMENT MCALL

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Not available.

MAC

```
        ...             %This code is on segment 20B.
        LDT     (PAR    %Load register T with address of parameter list.
        MON     132     %Monitor call JumpToSegment.
        ...             %Return after ExitFromSegment.
PAR,    SUBR            %Address of subroutine.
        SEG             %New segment to be loaded.
        ...
SUBR,   ...
SEG,  10030             %New segments are 20B and 30B in this example.
                        %ExitFromProgram shows the code on segment 30.
```

ND-100 User RT and user SYSTEM RT programs

MLAMU — LAMUFUNCTION — 3158

Performs various functions on the LAMU system. A LAMU is a logically addressed memory unit. The LAMU system is an extension to the ND-100 segment structure. Programs may address more space than provided by the 3 available segments.

- Several CPUs may share the same LAMU address space.

PARAMETERS

→ Function code. 0 means create a LAMU.
1 means delete a LAMU.
2 means connect a LAMU to your program.
3 means disconnect a LAMU.
4 means LAMU pages to nothing.
5 means LAMU pages to swapping.
9 means create system LAMU.
→ LAMU identifier.
→ The RT program the LAMU is connected to. 0 means background programs.
→ The number of pages in the LAMU.
→ The first logical address in the LAMU. Legal values are 100B to 277B.
→ The LAMU's start address in physical memory. You can specify all existing physical pages defined as LAMU areas.
← Standard error code. See appendix A.

PASCAL

```
Func, LAMUident, Prog, NoPages, LogAddr, PhysAddr : INTEGER2;
...
LAMUFunction(Func, LAMUident, Prog, NoPages, LogAddr, PhysAddr);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  Func  COMP.
01  LAMUident  COMP.
01  Prog  COMP.
01  NoPages  COMP.
01  LogAddr  COMP.
01  PhysAddr  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "LAMUFunction" USING Func, LAMUident, Prog,
                                      NoPages, LogAddr, PhysAddr.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  Func, LAMUident, Prog, NoPages, LogAddr, PhysAddr
      ...
      Monitor_Call('LAMUFunction', Func, LAMUident, Prog,
     C                             NoPages, LogAddr, PhysAddr)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 — All users — All programs

315B **LAMUFUNCTION** XLAMU

Continued from previous page...

PLANC

```
INTEGER : Func, LAMUident, Prog, NoPages, LogAddr, PhysAddr
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('LAMUFunction', Func, LAMUident, Prog, NoPages, &
                                LogAddr, PhysAddr)
```

ASSEMBLY-500

Not available.

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     315     %Monitor call LAMUFunction.
        JMP     ERROR   %Error return from monitor call.
        ...             %Normal return.
ERROR,  ...             %Error number in register A.
        ...
PAR,    FUNC            %Function.
        LAMUID          %LAMU-identifier.
        PROG            %Program that the LAMU is (dis)connected from.
        PAGES           %Number of pages in LAMU.
        LOGAD           %Logical address.
        PHYS            %Physical address.
        ...
FUNC,   ...
LAMUID, ...
PROG,   ...
PAGES,  ...
LOGAD,  ...
PHYS,   ...
```

ND-100 All users All users

BLOGI LogInStart 326B

Logs in a user on a terminal and starts a subsystem.

- The terminal must be free.
- The subsystem must be reentrant. See @DUMP-PROGRAM-REENTRANT.

PARAMETERS

→ Logical device number of a terminal.
→ User name.
→ Password.
→ Project password.
→ Subsystem to start.
→ The 5 user parameters. See SetUserParam.
← Status. Unsuccessful log in returns -1.

PASCAL

```
TermNo : INTEGER2;
UserName : PACKED ARRAY [0..63] OF CHAR;
Password, ProjPassword : PACKED ARRAY [0..15] OF CHAR;
Subsystem : PACKED ARRAY [0..31] OF CHAR;
UserParam : BITMAP;
...
LogInStart(TermNo, UserName, Password, ProjPassword, Subsystem, UserParam);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  TermNo  COMP.
01  UserName   PIC X(64).
01  Password   PIC X(16).
01  ProjPassword   PIC X(16).
01  Subsystem   PIC X(32).
01  UserParam.
    02  array COMP OCCURS 5 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "LogInStart" USING TermNo, UserName, Password,
                            ProjPassword, Subsystem, UserParam.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  TermNo
      CHARACTER  UserName*64, Password*16, ProjPassword*16, Subsystem*32
      INTEGER  UserParam(5)
      ...
      Monitor_Call('LogInStart', TermNo, UserName(1:64), Password(1:16),
     C               ProjPassword(1:16), Subsystem(1:32), UserParam(1))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 All users All programs

326B **LogInStart** **NLOGI**

Continued from previous page...

PLANC

```
INTEGER : TermNo
BYTES : UserName(0:63), Password(0:15), ProjPassword(0:15), Subsystem(0:31)
INTEGER ARRAY : UserParam(0:4)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('LogInStart', TermNo, UserName, Password, ProjPassword, &
                           Subsystem, UserParam(0))
```

ASSEMBLY-500

Not available.

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     326     %Monitor call LogInStart.
        ...
        JMP     ERROR   %Handle error if STAT is negative.
        ...
ERROR,  ...             %Handle this error.
        ...
PAR,    TERNO           %Terminal number.
        USER            %User name.
        PASSW           %Password.
        PROJP           %Project password.
        SUBSYS          %Subsystem.
        USRPAR          %User parameter.
        STAT            %Status returned.
        ...
TERNO,  ...
USER,   'A-HANSEN'      %Login user A-HANSEN.
PASSW,  'MAY'           %Use MAY as password.
PROJP,  'CHEESE'        %Use CHEESE as a project password.
SUBSYS, 'NOTIS-WP'      %Use NOTIS-WP as a subsystem.
USRPAR, ...             %
        ...             %
        ...             %
        ...             %
        ...             %
STAT,   0
```

ND-100 **All users** **All users**

MXPISG — MaxPagesInMemory — 417B

Sets the maximum number of pages a segment may have in physical memory at a time.

- This monitor call applies to ND-500 logical segments only.
- The segment must be in use.

See also FixInMemory.

PARAMETERS

→ Logical segment number in your domain. If you specify 0, a segment number is found from the parameter address.
→ Segment type. Use 0 for data segments and 1 for program segments.
→ Number of pages.
← Standard error code. See appendix A.

PASCAL

```
SegmentNo, SegType, NoOfPages : LONGINT;
...
MaxPagesInMemory(SegmentNo, SegType, NoOfPages);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  SegmentNo  COMP.
01  SegType  COMP.
01  NoOfPages  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "MaxPagesInMemory" USING SegmentNo, SegType, NoOfPages.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  SegmentNo, SegType, NoOfPages
      ...
      Monitor_Call('MaxPagesInMemory', SegmentNo, SegType, NoOfPages)
      IF (ErrCode .NE. 0) THEN ...
```

ND-500 | All users | All programs

417B **MaxPagesInMemory** MXPISG

Continued from previous page...

PLANC

```
INTEGER : SegmentNo, SegType, NoOfPages
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('MaxPagesInMemory', SegmentNo, SegType, NoOfPages)
```

ASSEMBLY-500

```
SegmentNo : W  BLOCK  1
SegType : W  BLOCK  1
NoOfPages : W  BLOCK  1
MaxPagesInMemory : EQU 37B9 + 417B
        ...
        CALLG MaxPagesInMemory, 3, SegmentNo, SegType, NoOfPages
```

MAC

Not available.

ND-500 All users All users

FEXC5 MEMORYALLOCATION 61B

Fixes or unfixes ND-100 segments to be used by the ND-500-MONITOR. You may also reserve a contiguous area in physical memory. Various other memory functions are reserved for the ND-500-MONITOR.

- This monitor call is not normally used by ordinary programs.
- You may use this monitor call to reserve space for DMA buffers.

See also FixScattered, FixInMemory, FixContiguous, and UnFixSegment.

PARAMETERS

→ Function code. Use 4, 5 or 6 as shown on the next page.
→ Parameter 2. Depends on the function code.
→ Parameter 3. Depends on the function code.
→ Parameter 4. Depends on the function code.
→ Parameter 5. Depends on the function code.
→ Parameter 6. Depends on the function code.
← Standard error code. See appendix A.

PASCAL

```
FuncCode, Param2, Param3, Param4, Param5, Param6 : INTEGER2;
...
MemoryAllocation(FuncCode, Param2, Param3, Param4, Param5, Param6);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FuncCode  COMP.
01  Param2  COMP.
01  Param3  COMP.
01  Param4  COMP.
01  Param5  COMP.
01  Param6  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "MemoryAllocation" USING FuncCode, Param2, Param3,
                 Param4, Param5, Param6.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FuncCode, Param2, Param3, Param4, Param5, Param6
      ...
      Monitor_Call('MemoryAllocation', FuncCode, Param2, Param3,
     C                                 Param4, Param5, Param6)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 All users All programs

61B MEMORYALLOCATION FEXCS

Continued from previous page...

PLANC

```
INTEGER : FuncCode, Param2, Param3, Param4, Param5, Param6
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('MemoryAllocation', FuncCode, Param2, Param3, &
                                 Param4, Param5, Param6)
```

ASSEMBLY-500

Not available.

MAC

```
        LDA     (PAR      %Load register A with address of parameter list.
        MON     61        %Monitor call MemoryAllocation.
        JMP     ERROR     %Error return from monitor call.
        ...               %Normal return.
ERROR,  ...               %Error number in register A.
        ...
PAR,    FUNC
        PARA2
        PARA3
        PARA4
        PARA5
        PARA6
        ...
FUNC,   ...
PARA2,  ...
PARA3,  ...
PARA4,  ...
PARA5,  ...
PARA6,  ...
```

Function code 4: Fix a segment contiguously at any address within a specified physical memory area.

Parameter 2: Segment number.
Parameter 3: First legal physical memory page.
Parameter 4: Last legal physical memory page.
Parameter 5 and 6: Not used.

ErrCode 0: Physical memory area occupied.
ErrCode 1: Parameter 3 greater last page in physical memory.
ErrCode 2: Segment error, eg., Return: A = 2 (demand, already fixed etc.) Skip return: OK. Segment is fixed contiguously. Skip return: A = first physical page in segment.

Function code 5: Reserve a contiguous area in physical memory.

Parameter 2: Number of pages to reserve.
Parameter 3: First legal physical memory page.
Parameter 4: Last legal physical memory page.
Parameter 5 and 6: Not used. Return: A = 0 area occupied Return: A = 1 Parameter 3 > last physical page in memory Return: A = 2 no free table element in table for allocated memory areas Return: A = 2 (system generation) Skip return: OK. Area allocated. Skip return: A = first physical page in area Skip return: T = area index (to be used when deallocating area) Skip return: T = see function code 6

Function code 6: Release an area in physical memory which has been reserved by function code 5.

Parameter 2 = table index (T-reg on return from function 5)
Parameter 3 = first physical page in area (A-reg on return from func 5) Parameter 4 = not used Parameter 5 = not used Parameter 6 = not used Return: error Skip return: OK

ND-100	All users	All users

UNFIXM **MEMORYUNFIX** 411B

Releases a fixed segment in your domain from physical memory. A fixed segment has all its pages fixed in physical memory. After MemoryUnFix the pages may be swapped between the disk and physical memory.

See also FixInMemory.

PARAMETERS

→ Address. Only the segment number in the address is significant.

PASCAL

```
Address : LONGINT;
...
MemoryUnFix(Address);
```

COBOL

```
01  Address  COMP.
    ...
    MONITOR-CALL "MemoryUnFix" USING Address.
```

FORTRAN

```
      INTEGER  Address
      ...
      Monitor_Call('MemoryUnFix', Address)
```

ND-500 All users All programs

411B **MEMORYUNFIX** UNFIXM

Continued from previous page...

PLANC

```
INTEGER : Address
...
Monitor_Call('MemoryUnFix', Address)
```

ASSEMBLY-500

```
Address : W  BLOCK  1
MemoryUnFix : EQU 37B9 + 411B
        ...
        CALLG MemoryUnFix, 1, Address
        IF K GO error,
        ...
Error,  ...
```

MAC

Not available.

ND-500 All users All users

ND500 — ND500FUNCTION — 60B

Controls the ND-500 from the ND-100. Various functions are available.

- This monitor call is intended to be used by the operating system itself only.

PARAMETERS

→ Function. See the next page for a complete list. The A register pointing to the list of parameters. The other parameters depend on the function. The function is a 16-bit word. The parameters are either arrays of 16-bit words or single 32-bit words.

← Successful execution of a function skips one instruction when returning.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 | All users | All programs

60B ND500Function N500M

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Not available.

MAC

```
        LDA     (PAR      %Load register A with address of parameter list.
        MON     60        %Monitor call ND500Function.
        JMP     ERROR     %Error return from monitor call.
        ...               %Normal return.
ERROR   ...               %Handle this error.
        ...
PAR,    FUNC
        REGNO
        VALUE
FUNC,   0                 %
REGNO,  5                 %Function 0B: Read register number 5.
VALUE,  0                 %
        0
```

The following functions are available:

0B	Read register	<register no.> <value>
1B	Write register	<register no.> <value>
2B	Read program memory	<no. of bytes> <ND-500 address> <data area> <no. of bytes returned>
3B	Read data memory	<no. of bytes> <ND-500 address> <data area> <no. of bytes returned>
4B	Write program memory	<no. of bytes> <ND-500 address> <data area>
5B	Write data memory	<no. of bytes> <ND-500 address> <data area>
6B	Place segment	<file name> <segment base> <size in bytes> <segment type>
7B	Load swapper	<swapper segment name>
10B	Read ND-500 Registers	<register block>
11B	Write ND-500 Registers	<register block>
12B	Start program	<stop reason> <returned trap info> <clear time used>
13B	Connect file	<file name> <access code> <default type> <connect no.> <returned connect no.>
14B	Close file	<file no.>
15B	Reserve ND-500-process	<start address after escape> <version string of PTO>
16B	Release ND-500-process	
17B	List open files	
20B	Time used	
21B	Who is on	
22B	Set error flag	<value>
23B	Read Control store	<Control store address> <no of 16-bit words> <data area>
24B	Write Control store	<Control store address> <no of 16-bit words> <data area>

25B	Start micro program	<micro program start address>
26B	Data memory examine	<address> <value>
27B	Data memory deposit	<address> <value>
30B	Prog. memory examine	<address> <value>
31B	Prog. memory deposit	<address> <value>
32B	Absolute memory read	<no. of bytes> <ND-500 address> <data area> <no. of bytes returned>
33B	Absolute memory write	<no.of bytes> <ND-500 address> <data area>
34B	Stop micro program	
35B	Master clear	
37B	Load control store	<Control store address> <no of words> <file name>
40B	Define memory config.	<start page> <no. of memory parts> <part array>
41B	Read comm. status	<status (bits 16:31 ND-500, bits 0:15 ND-100)> <memory address register>
43B	Reserve for spec. use	
44B	Release after spec use	
46B	Define swap file	<file name>
47B	Delete swap file	<file name>
50B	Test function	<I1> <I2> <I3> <I4>
51B	Read interface regist.	<register value/I4>
52B	Give ND-500 pages	<number of pages>
53B	Take ND-500 pages	<number of pages>
54B	Start swapper	
55B	Start place	
56B	End place	
57B	Microprogram version	<version number/I4>
60B	List memory config.	<array (register block start/I2, ND-100 procadr/I4, ND-500 null/I2, memory parts/I2(0:17B), access table/BY(0:17B)>
61B	Reserve N500 and N500 memory	<no. of pages> <first page no.>
62B	Define histogram	<start address> <interval size> <no. of intervals>
63B	Start histogram	
64B	Stop histogram	
65B	Read histogram	<array>
66B	Release histogram	
67B	Search for proc.entry	<process name> <record>
70B	Get process entry	<process> <record>
71B	Seach for phys.segm.	<name of physical segment> <array>
72B	Get physical segment	<physical segment no.> <array>
73B	Read physical segment	<physical segment no.> <address> <no. of bytes> <array>
74B	Set process name	<process name>

75B	User SYSTEM test (skip if SYSTEM)	
76B	Send msg. to swapper	<record>
77B	Read last message	<process no.> <record>
100B	Read process flag	<process no.> <flag>
101B	Set process flag	<process no.> <flag>
102B	Release ND-500 system	
103B	Read system param.	<parameter array>
104B	Write system param.	<parameter array>
105B	Set priority	<ND-100 monitor call priority> <max percent of ND-100 CPU time>
106B	Link to process	<process no.>
110B	Write physical segment	<segm no.> <ND-500 address> <no. of bytes> <data area>
111B	Start process log one	<process no.>
112B	Stop logging	
113B	Read log info	<data area>
114B	Release log facility	
115B	Start log all active processes	
117B	Abort process	<process no.>
120B	Set output device	<unit>
121B	Read from swapper process' data memory	<no. of bytes> <ND-500 address> <data area> <no. of bytes read>
122B	Logout process	<process no>
123B	Release all memory reserved by this process through function ReserveResource.	
124B	Start moncall log	
125B	Print moncall log	<array of 1024 16-bit words>
126B	Stop/release moncall log	
127B	Define standard domain	<array>
130B	Place standard domain	<name>
131B	Delete standard domain	<name>
132B	List standard domain	
133B	List execution queue	
134B	Place Debugger	
135B	Log out, process and abort corresponding RT programs	
136B	Activate a stopped process	
137B	Not used	
140B	Start residual place	
141B	Set block size of a file	
142B	Redefine default infant file	

ND-100	All users	All users

5TMOUT	ND500TIMEOUT	514B

Suspend the execution of an ND-500 program for a given time. The execution then continues after the monitor call. The program is placed in a time queue in the ND-500, not the ND-100.

- No reserved files or devices are released.

See also TimeOut, SuspendProgram and WaitForRestart.

PARAMETERS

→ Number of time units to suspend the program. Use 0 to restart programs immediately. This clears the restart flag.
→ The type of time units. 1 = basic time units, ie., 1/50th of a second, 2 = seconds, 3 = minutes, 4 = hours.
← Restart cause. 0 means that the defined time has elapsed. 1 means that an interrupt occurred. -1 means that the RT program was scheduled for repeated execution.

PASCAL

```
NoOfTimeUnits, TimeUnit, ReturnStatus : LONGINT;
...
ND500TimeOut(NoOfTimeUnits, TimeUnit, ReturnStatus);
```

COBOL

```
01  NoOfTimeUnits  COMP.
01  TimeUnit  COMP.
01  ReturnStatus  COMP.
    ...
    MONITOR-CALL "ND500TimeOut" USING NoOfTimeUnits, TimeUnit, ReturnStatus.
```

FORTRAN

```
      INTEGER  NoOfTimeUnits, TimeUnit, ReturnStatus
      ...
      Monitor_Call('ND500TimeOut', NoOfTimeUnits, TimeUnit, ReturnStatus)
```

ND-500	All users	All programs

514B	ND500TIMEOUT	5TMOUT

Continued from previous page...

PLANC

```
INTEGER : NoOfTimeUnits, TimeUnit, ReturnStatus
...
Monitor_Call('ND500TimeOut', NoOfTimeUnits, TimeUnit, ReturnStatus)
```

ASSEMBLY-500

```
NoOfTimeUnits : W  BLOCK  1
TimeUnit : W  BLOCK  1
ReturnStatus : W  BLOCK  1
ND500TimeOut : EQU 37B9 + 514B
        ...
        CALLG ND500TimeOut, 3, NoOfTimeUnits, TimeUnit, ReturnStatus
```

MAC

Not available.

ND-500	All users	All users

CRALM NEWFILEVERSION 253B

Creates new versions of a file. You may create new versions for both indexed, contiguous and allocated files.

- You must have directory access to the user area where you create the file. User SYSTEM and RT get the owners access rights.
- The number following the semicolon in a file name is the version number. For example, TEST:SYMB;4 version 4 of the file.
- The file must exist in advance.
- Use DeleteFile to delete file versions.

See also CreateFile, @CREATE-NEW-VERSIONS, and @ALLOCATE-NEW-VERSIONS.

PARAMETERS

→ The file name including the version number. The version number defines the total number of versions. It includes the existing versions.
→ Start address of the first new version. Use 0 for contiguous and indexed files.
→ File size in pages. Use 0 for indexed files.
← Standard error code. See appendix A.

PASCAL

```
FileName : PACKED ARRAY [0..63] OF CHAR;
FirstPage, NoOfPages : LONGINT;
...
NewFileVersion(FileName, FirstPage, NoOfPages);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileName    PIC X(64).
01  FirstPage   COMP PIC S9(10).
01  NoOfPages   COMP PIC S9(10).
01  ErrCode     COMP.
    ...
    MONITOR-CALL "NewFileVersion" USING FileName, FirstPage, NoOfPages.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      CHARACTER  FileName*64
      INTEGER*4  FirstPage, NoOfPages
      ...
      Monitor_Call('NewFileVersion', FileName(1:64), FirstPage, NoOfPages)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

253B **NEWFILEVERSION** CRALM

Continued from previous page...

PLANC

```
INTEGER4 : FirstPage, NoOfPages
BYTES : FileName(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('NewFileVersion', FileName, FirstPage, NoOfPages)
```

ASSEMBLY-500

```
FileName : STRINGDATA 'EXAMPLE:SYMB;2'''      %Make a 2nd version of
FirstPage : W  BLOCK  1
NoOfPages : W  BLOCK  1
ErrCode : W  BLOCK  1                         % EXAMPLE:SYMB.
NewFileVersion : EQU 37B9 + 253B
        ...
        CALLG NewFileVersion, 3, FileName, FirstPage, NoOfPages
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                         %ErrorCode in W1 register.
```

MAC

```
        LDX     (FILE    %Address of string containing file name.
        LDD     PAGNO    %Page number of first page.
        LDT     (PAGES   %Address of number of pages.
        MON     253      %Monitor call NewFileVersion.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILE,   'EXAMPLE:SYMB;2' %Create 2nd version of file EXAMPLE:SYMB
PAGNO,  ...              %
        ...              %A double word.
PAGES,  ...              %
        ...              %A double word.
```

ND-100 and ND-500 | All users | All users

SUSCM NEWUSER 241B

Switches the user name you are logged in under. The command is similar to logging out and then log in as another user. Your program continues under this user name.

- Restore the old user name with OldUser.
- You may execute NewUser more than once without OldUser in between. OldUser allways reset the first user name.
- If originally logged in as user RT, it is not possible to log in as user SYSTEM.

See also OldUser and @ENTER.

PARAMETERS

→ New user name.
→ Password. Use the contents of the password location in the user entry. See appendix C.
→ Project password.
← Status. Public users return 0. User SYSTEM returns 1. User RT returns 2.
← Standard error code. See appendix A.

PASCAL

```
UserName, ProjPasswd : PACKED ARRAY [0..15] OF CHAR;
UserPasswd : INTEGER2;
ReturnStatus : INTEGER2;
...
NewUser(UserName, UserPasswd, ProjPasswd, ReturnStatus);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  UserName   PIC X(16).
01  UserPasswd  COMP.
01  ProjPasswd   PIC X(16).
01  RetStat  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "NewUser" USING UserName, UserPasswd, ProjPasswd, RetStat.

    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  ReturnStatus, UserPasswd
      CHARACTER  UserName*16, ProjPasswd*16
      ...
      Monitor_Call('NewUser', UserName(1:16), UserPasswd,
     C                  ProjPasswd(1:16), ReturnStatus)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 User RT and user SYSTEM Background programs

241B **NEWUSER** **SUSCM**

Continued from previous page...

PLANC

```
INTEGER : ReturnStatus, UserPasswd
BYTES : UserName(0:15), ProjPasswd(0:15)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('NewUser', UserName, UserPasswd, ProjPasswd, ReturnStatus)
```

ASSEMBLY-500

```
UserType : W BLOCK 1
UserName : STRINGDATA  'A-HANSEN'''
UserPasswd : W  BLOCK  1
ProjPasswd : STRINGARRAY  40B
ErrCode : W  BLOCK  1
NewUser : EQU 37B9 + 241B
        ...
        CALLG NewUser, 4, UserName, UserPasswd, ProjPasswd, UserType
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     (PROJP   %Address of project password.
        LDX     (USER    %Address of string containing user name.
        LDA     (PASSW   %User password coded as integer.
        MON     241      %Monitor call NewUser.
        JMP     ERROR    %Error return from monitor call.
        STA     STAT     %Normal return, store status returned.
        ...
ERROR,  ...              %Error number in register A.
        ...
USER,   'A-HANSEN'       %Use A-HANSEN as user name.
PASSW,  ...              %Password as represented in the user entry.
PROJP,  ...
STAT,   0
```

ND-100 and ND-500 **User RT and user SYSTEM** **Background programs**

DSCNT — NoInterruptStart — 107B

StartOnInterrupt connects an RT program to interrupts from a device. You remove this connection with NoInterruptStart.

- The program may be in the time queue. It is then removed. Periodic execution is prevented.
- Reserved resources are not released.
- The program is not removed from the execution queue.

See also StartOnInterrupt and @DSCNT.

PARAMETERS

→ Address of RT description.

PASCAL

```
RTProgram : INTEGER2;
...
NoInterruptStart(RTProgram);
```

COBOL

```
01  RTProgram  COMP.
    ...
    MONITOR-CALL "NoInterruptStart" USING RTProgram.
```

FORTRAN

```
      INTEGER  RTProgram
      ...
      Monitor_Call('NoInterruptStart', RTProgram)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

107B **NoInterruptStart** **DSCRT**

Continued from previous page...

PLANC

```
INTEGER : RTProgram
...
Monitor_Call('NoInterruptStart', RTProgram)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
NoInterruptStart : EQU 37B9 + 107B
        ...
        CALLG NoInterruptStart, 1, RTProgram
```

MAC

```
        LDA     (PAR      %Load register A with address of parameter list.
        MON     107       %Monitor call NoInterruptStart.
        ...
PAR,    RTPRO             %Address of RT description.
        ...
RTPRO,  ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

ALTOF NormalPageTable 34B

Sets the alternative page table equal to the normal page table. All memory addresses are mapped through the normal page table after this monitor call.

- Use AltPageTable to set an alternative page table.

See also AltPageTable and @ALTOF.

PARAMETERS

This monitor call has no parameters.

PASCAL

```
NormalPageTable;
```

COBOL

```
    MONITOR-CALL "NormalPageTable".
```

FORTRAN

```
      Monitor_Call('NormalPageTable')
```

ND-100 All users All programs

34B **NormalPageTable** ALTOF

Continued from previous page...

PLANC

```
Monitor_Call('NormalPageTable')
```

ASSEMBLY-500

Not available.

MAC

```
MON     34      %Monitor call NormalPageTable.
```

ND-100 All users All users

NoWaitSwitch

Switches No Wait on and off. No Wait is useful for input from and output to several devices simultaneously. In No Wait, the program does not wait for input or output to complete. Monitor calls like InByte returns the error code 3 instead.

- SuspendProgram or WaitForRestart may passivate the program afterwards. The program then restarts when input or output to the device is completed.

See also InByte, OutByte, and TerminalNoWait.

PARAMETERS

→ Logical device number of a character device. See appendix B.
→ Input or output flag. Use 0 for input and 1 for output.
→ No Wait flag. Use 0 to switch No Wait on, and 1 to switch it off.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, IOFlag, WaitFlag : INTEGER2;
...
NoWaitSwitch(DeviceNumber, IOFlag, WaitFlag);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  IOFlag  COMP.
01  WaitFlag  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "NoWaitSwitch" USING DeviceNumber, IOFlag, WaitFlag.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, IOFlag, WaitFlag
      ...
      Monitor_Call('NoWaitSwitch', DeviceNumber, IOFlag, WaitFlag)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

36B **NoWaitSwitch** **NOWT**

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, IOFlag, WaitFlag
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('NoWaitSwitch', DeviceNumber, IOFlag, WaitFlag)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
IOFlag : W  BLOCK  1
WaitFlag : W  BLOCK  1
ErrCode : W  BLOCK  1
NoWaitSwitch : EQU 37B9 + 36B
        ...
        CALLG NoWaitSwitch, 3, DeviceNumber, IOFlag, WaitFlag
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDA     (PAR     %Load register A with address of parameter list.
        MON     36       %Monitor call NoWaitSwitch.
        JAF     ERROR    %Handle error if register A is non-zero.
        ...
ERROR,  ...              %Handle this error.
        ...
PAR,    DEVNO            %Logical device number.
        IOF              %Input/output flag.
        WAITFL           %No Wait flag.
        ...
DEVNO,  ...
IOF,    ...
WAITFL, ...
```

ND-100 and ND-500 **All users** **All users**

OCTO OctobusFunction 324B

Performs various functions on an Octobus. Mainly intended for internal usage.

PARAMETERS

→ Function.

0 means kick.
1 means wait for kick.
5 means read Octobus status.
6 means "Who am I".

→ Logical device number.

← Function dependent. Function 0 returns the destination station. Function 5 and 6 return a status value. The last transmit status is returned in bit 31:16. The hardware status is in bit 15:0. Function 1 does not use this parameter.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 and ND-500 | All users | All programs

324B OctobusFunction OCTO

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
Func : W  BLOCK  1
DevNo : W  BLOCK  1
Par3 : W  BLOCK  1     %May be destination station or return value.
Status : W  BLOCK  1
OctobusFunction : EQU 37B9 + 324B
        CALLG OctobusFunction, 3, Func, DevNo, Par3
        IF K GO Error
        W1 =: Status
        ...
Error,  ...
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     324     %Monitor call OctobusFunction.
        ...
PAR,    FUNC
        LDN
        STAT
FUNC,   0               %Function.
LDN,    ...             %Logical device number.
STAT,   ...             %Status.
```

ND-100 and ND-500 | All users | All users

ELOFF OffEscLocalFunction 303B

Delays the escape and local functions for your terminal. Then the ESCAPE key or LOCAL key does not terminate a program or remote connection immediately. Their functions are delayed until OnEscLocalFunction is executed.

- Enable the escape and local functions again by OnEscLocalFunction.
- If both the ESCAPE and the LOCAL keys are pressed, local is executed on OnEscLocalFunction.
- This monitor call is used to protect critical instruction sequences.

See also SetEscapeHandling, OnEscLocalFunction, DisableEscape, and DisableLocal.

PARAMETERS

← Standard error code. See appendix A.

PASCAL

```
OffEscLocalFunction;
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  ErrCode  COMP.
    ...
    MONITOR-CALL "OffEscLocalFunction".
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      Monitor_Call('OffEscLocalFunction')
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 | All users | Background programs

303B **OffEscLocalFunction** **ELOFF**

Continued from previous page...

PLANC

```
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('OffEscLocalFunction')
```

ASSEMBLY-500

Not available.

MAC

```
        MON     303     %Monitor call OffEscLocalFunction.
        JMP     ERROR   %Error return from monitor call.
        ...             %Normal return.
ERROR,  ...             %Error number in register A.
        ...
```

ND-100 **All users** **Background programs**

BUSCM OLDUSER 242B

Switches back to the user name you were logged in under before NewUser. The command is similar to logging out and then log in as another user. Your program continues under the old user.

- The monitor call has no function if NewUser has not been executed.
- You may execute NewUser more than once without OldUser in between. OldUser allways reset the first user name.

See also NewUser and @ENTER.

PARAMETERS

This monitor call has no parameters.

PASCAL

```
OldUser;
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  ErrCode  COMP.
    ...
    MONITOR-CALL "OldUser"
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      Monitor_Call('OldUser')
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | Background programs

242B **OLDUSER** **BUSCM**

Continued from previous page...

PLANC

```
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('OldUser')
```

ASSEMBLY-500

```
ErrCode : W  BLOCK  1
OldUser : EQU 37B9 + 242B
        ...
        CALLG OldUser, 0
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        MON       242       %Monitor call OldUser.
        JMP       ERROR     %Error return from monitor call.
        ...                 %Normal return.
ERROR,  ...                 %Error number in register A.
        ...
```

ND-100 and ND-500 **All users** **Background programs**

KLON — OnEscLocalFunction — 302B

Enables delayed escape and local functions for you terminal. The ESCAPE key then terminate a program unless it is disabled. The key with the local function will terminate connections to remote computers.

- Delay the escape and local functions by OnEscLocalFunction.
- A timeout from a remote computer enables the local function.

See also SetEscapeHandling, OffEscLocalFunction, EnableEscape, and EnableLocal.

PARAMETERS

← Standard error code. See appendix A.

PASCAL

```
OnEscLocalFunction;
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  ErrCode  COMP.
    ...
    MONITOR-CALL "OnEscLocalFunction".
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      Monitor_Call('OnEscLocalFunction')
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 | All users | Background programs

302B	ONESCLOCALFUNCTION	ELON

Continued from previous page...

PLANC

```
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('OnEscLocalFunction')
```

ASSEMBLY-500

Not available.

MAC

```
        MON     302     %Monitor call OnEscLocalFunction.
        JMP     ERROR   %Error return from monitor call.
        ...             %Normal return.
ERROR,  ...             %Error number in register A.
        ...
```

ND-100	All users	Background programs

OPEN **OpenFile** 50B

Opens a file. You cannot access a file before you open it. Specify what kind of access you want, eg., sequential write or random read.

- CloseFile closes the file.
- Opened files are closed when a program terminates.
- You may have maximum 18 files opened at a time.
- Files are protected from access by unauthorized users. Use @FILE-STATISTICS to list the file access.
- ND-500 has its own file numbers which may differ from the SINTRAN III file numbers.

See also ScratchOpen, SetPermanentOpen, DirectOpen, and @OPEN-FILE.

PARAMETERS

← File number. Monitor calls for input and output use this number.
→ Access code. The legal values are shown below.
0: Sequential write.
1: Sequential read.
2: Random read or write.
3: Random read only.
4: Sequential read or write.
5: Sequential write append.
6: Random read or write common on contiguous files.
7: Random read common on contiguous files.
8: Random read or write on contiguous files. Direct transfer for ReadFromFile, WriteToFile and DeviceFunction in RT programs.
9: Random read, write append for WriteToFile.
→ File name. Unabbreviated file names are most efficient.
→ Default file type. Do not include the colon.
← Standard error code. See appendix A.

PASCAL

```
FileNo, AccessCode : INTEGER2;
FileName : PACKED ARRAY [0..63] OF CHAR;
FileType : PACKED ARRAY [0..3] OF CHAR;
...
OpenFile(FileNo, AccessCode, FileName, FileType);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNo  COMP. 01  AccessCode  COMP.
01  FileName   PIC X(64). 01  FileType   PIC X(4).
01  ErrCode  COMP.
    MONITOR-CALL "OpenFile" USING FileNo, AccessCode, FileName, FileType.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNo, AccessCode
      CHARACTER  FileName*64, FileType*4
      Monitor_Call('OpenFile', FileNo, AccessCode, FileName(1:64),
     C                 FileType(1:4))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 All users All programs

50B	OPENFILE	OPEN

Continued from previous page...

PLANC

```
INTEGER : FileNo, AccessCode
BYTES : FileName(0:63), FileType(0:3)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('OpenFile', FileNo, AccessCode, FileName, FileType)
```

ASSEMBLY-500

```
FileNo : W  BLOCK  1  %Returned ND-500 open file number if 0 on input.
S3No : W  BLOCK  1    %SINTRAN III open file number as optional parameter.
AccessCode : W  BLOCK  1
FileName : STRINGDATA  'EXAMPLE'''
FileType : STRINGDATA  'SYMB'''
ErrCode : W  BLOCK  1
OpenFile : EQU 37B9 + 50B
        ...
        CALLG OpenFile, 4, FileNo, AccessCode, FileName, FileType
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (FILE    %If 0, the name is read from the terminal.
        LDA     (TYPE    %Address of default file type string.
        LDT     ACCES    %Access code.
        MON     50       %Monitor call OpenFile.
        JMP     ERROR    %Error return from monitor call.
        STA     FILNO    %Normal return, store the file number returned.
        ...
ERROR,  ...              %Error number in register A.
        ...
FILNO,  0
ACCES,  ...
FILE,   'EXAMPLE'        %Open EXAMPLE:SYMB
TYPE,   'SYMB'           %
```

ND-100 and ND-500	All users	All users

POPFN — OpenFileInfo — 257B

Gets information about an open file. You specify the file name. The monitor call returns the file number and the access type. The logical device number of peripheral equipment is returned for peripheral files.

See also OpenFile.

PARAMETERS

→ File name.
→ File type.
← File number.
← Access code. 0 means read. 1 means write. 2 means read and write.
← Logical device number of peripheral device. This is only relevant for peripheral files.
← Standard error code. See appendix A.

PASCAL

```
FileName : PACKED ARRAY [0..63] OF CHAR;
FileType : PACKED ARRAY [0..3] OF CHAR;
FileNo, AccessCode, DevNo : INTEGER2;
...
GetOpenFileInfo(FileName, FileType, FileNo, AccessCode, DevNo);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileName    PIC X(64).
01  FileType    PIC X(4).
01  FileNo  COMP.
01  AccessCode  COMP.
01  DevNo  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "OpenFileInfo" USING FileName, FileType, FileNo,
                                      AccessCode, DevNo.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      CHARACTER  FileName*64, FileType*4
      INTEGER  FileNo, AccessCode, DevNo
      ...
      Monitor_Call('OpenFileInfo', FileName(1:64), FileType(1:4), FileNo,
     C                             AccessCode, DevNo)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

257B OpenFileInfo POPFM

Continued from previous page...

PLANC

```
BYTES : FileName(0:63), FileType(0:3)
INTEGER : FileNo, AccessCode, DevNo
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('OpenFileInfo', FileName, FileType, FileNo,&
                             AccessCode, DevNo)
```

ASSEMBLY-500

```
FileName : STRINGDATA  'EXAMPLE'''          %Get open-file info
FileType : STRINGDATA  'SYMB'''             %  of EXAMPLE:SYMB.
FileNo : W  BLOCK  1                %ND-500 open file number.
DevNo : W  BLOCK  1                 %Peripheral device number.
OpenCode : W  BLOCK  1              %Access code.
S3No : W  BLOCK  1      %Optional SINTRAN III open file number as 6th param.
ErrCode : W  BLOCK  1
OpenFileInfo : EQU 37B9 + 257B
        ...
        CALLG OpenFileInfo, 5, FileName, FileType, FileNo, OpenCode, DevNo
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                           %ErrorCode in W1 register.
```

MAC

```
        LDX     (FILE   %Address of string containing file name.
        LDA     (TYPE   %Address of string containing default file type.
        MON     257     %Monitor call OpenFileInfo.
        JMP     ERROR   %Error return from monitor call.
        STT     FILNO   %Normal return, store file number returned.
        STA     ACODE   %Store access code.
        COPY    SD  DA
        STA     DEVNO   %Store peripheral dev. number, if peripheral file.
        ...
ERROR,  ...             %Handle error, register A: error number,
        ...             %              register D: peripheral file number.
FILE,   'EXAMPLE'       %Obtain file number for EXAMPLE:SYMB.
TYPE,   'SYMB'
FILNO,  0
ACODE,  ...
DEVNO,  ...
```

ND-100 and ND-500 | All users | All users

B8OUT — OUT8BYTES — 24B

Writes 8 bytes to a character device, eg., a terminal. All 8 bytes are output. OutUpTo8Bytes stops if a byte is 0.

- On the ND-500, you are adviced to use the faster OutputString.
- Appendix F contains an ASCII table.

See also OutUpTo8Bytes, OutByte, OutString, OutputString, OutMessage, OutNumber, and In8Bytes.

PARAMETERS

→ Logical device number. See appendix B.
→ The string of bytes to output.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber : INTEGER2;
OutData : BITMAP;
...
Out8Bytes(DeviceNumber, OutData);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  OutData   PIC X(8).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "Out8Bytes" USING DeviceNumber, OutData.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber
      CHARACTER  OutData*8
      ...
      Monitor_Call('Out8Bytes', DeviceNumber, OutData(1:8))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | **All users** | **All programs**

24B **OUT8BYTES** **B8OUT**

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber
BYTES : OutData(0:7)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('Out8Bytes', DeviceNumber, OutData)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
OutData : STRINGDATA  'HELLO!!!'''
ErrCode : W  BLOCK  1
Out8Bytes : EQU 37B9 + 24B
        ...
        CALLG Out8Bytes, 2, DeviceNumber, OutData
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Logical device address.
        LDD     BYTES    %First 4 bytes to be written.
        LDX     BYTES+2  %Next 2 bytes to be written.
        COPY    SX  DL
        LDX     BYTES+3  %Last 2 bytes to be written.
        MON     24       %Monitor call Out8Bytes.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
BYTES,  'HELLO!!!'
```

ND-100 and ND-500 **All users** **All users**

OSIZE — OUTBUFFERSPACE — 67B

Gets the current number of bytes in the output buffer. Terminals and other character devices place output in a buffer. Monitor calls like OutByte writes to this buffer.

- Use ExecutionInfo to get the logical device number for terminals. You can specify 1 for your own terminal.
- This monitor call is not available for internal devices. Use InBufferSize and subtract this size from the buffer size.

See also ClearOutBuffer and InBufferSpace.

PARAMETERS

→ Logical device number. See appendix B.
← Number of bytes which can be written before the program must wait.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, NoOfBytes : INTEGER2;
...
OutBufferSpace(DeviceNumber, NoOfBytes);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  NoOfBytes  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "OutBufferSpace" USING DeviceNumber, NoOfBytes.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, NoOfBytes
      ...
      Monitor_Call('OutBufferSpace', DeviceNumber, NoOfBytes)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | **All users** | **All programs**

67B	OUTBUFFERSPACE	OSIZE

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, NoOfBytes
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('OutBufferSpace', DeviceNumber, NoOfBytes)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
NoOfBytes : W  BLOCK  1
ErrCode : W  BLOCK  1
OutBufferSpace : EQU 37B9 + 67B
        ...
        CALLG OutBufferSpace, 1, DeviceNumber
        IF K GO ERROR
        W1 =: NoOfBytes                 %Result is returned in W1 register.
        ...
ERROR : W1 =: ErrCode                   %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Logical device number.
        MON     67       %Monitor call OutBufferSpace.
        JMP     ERROR    %Error return from monitor call.
        STA     COUNT    %Normal return, store number of bytes.
        ...
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
COUNT,  0                %Number of bytes free space in outbuffer.
```

ND-100 and ND-500	All users	All users

OUTBT OutByte 2B

Writes one byte to a character device, eg., a terminal or an opened file. If the device is a word-oriented device, one word is written.

- The program waits if the output buffer of the device is full. You can change this with NoWaitSwitch or TerminalNoWait.
- The pointer to the next byte is incremented when you write to a mass storage file.
- Output from card readers are converted to ASCII characters. Use DeviceControl to write the 12-bit card columns.
- You are adviced to use the faster OutputString on the ND-500.
- Appendix F contains an ASCII table.

See also OutUpTo8Bytes, Out8Bytes, OutString, OutputString, OutMessage, OutNumber, and InByte.

PARAMETERS

→ Logical device number. See appendix B. Use 1 for your own terminal.
→ The byte to write.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, OutputValue : INTEGER2;
...
OutByte(DeviceNumber, OutputValue);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  OutputValue  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "OutByte" USING DeviceNumber, OutputValue.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, OutputValue
      ...
      Monitor_Call('OutByte', DeviceNumber, OutputValue)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

28	OUTBYTE	OUTBT

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, OutputValue
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('OutByte', DeviceNumber, OutputValue)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
OutputValue : W  BLOCK  1
ErrCode : W  BLOCK  1
OutByte : EQU 37B9 + 2B
        ...
        CALLG OutByte, 2, DeviceNumber, OutputValue
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Logical device number.
        LDA     BYTE     %Byte to be written.
        MON     2        %Monitor call OutByte.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,                   %Error number in A register.
        ...
DEVNO,  ...
BYTE,   ' A'
```

ND-100 and ND-500	All users	All users

MSG **OUTMESSAGE** 32B

Writes a message to the user's terminal. This is convenient for error messages in background programs.

- The maximum string length is 512 characters.
- Appendix F contains an ASCII table.

See also OutUpTo8Bytes, Out8Bytes, OutString, OutputString, OutNumber, and OutByte.

PARAMETERS

→ The string to be output.

PASCAL

```
Message : PACKED ARRAY [0..79] OF CHAR;
...
OutMessage(Message);
```

COBOL

```
01  Message    PIC X(100).
    ...
    MONITOR-CALL "OutMessage" USING Message.
```

FORTRAN

```
      CHARACTER  Message*80
      ...
      Monitor_Call('OutMessage', Message(1:80))
```

ND-100 and ND-500 | All users | Background programs

32B	OUTMESSAGE	MSG

Continued from previous page...

PLANC

```
BYTES : Message(0:79)
...
Monitor_Call('OutMessage', Message)
```

ASSEMBLY-500

```
Message : STRINGDATA  'This is a test.'''          %Message to be sent.
OutMessage : EQU 37B9 + 32B
        ...
        CALLG OutMessage, 1, Message
```

MAC

```
        LDX     (TEXT    %Address string to be sent to user's terminal.
        MON     32       %Monitor call OutMessage.
        ...
TEXT,   'THIS IS A TEXT' %String to be written.
```

ND-100 and ND-500	All users	Background programs

IOUT — OUTNUMBER — 35B

Writes a number to the user's terminal. The number can be output as an octal or a decimal value.

- The number may be in the range -32768 to 32767.

See also OutMessage, OutUpTo8Bytes, Out8Bytes, OutString, OutputString, and OutByte.

PARAMETERS

→ Octal or decimal output. Use 8 for octal and 10 for decimal.
→ The number to be output.

PASCAL

```
Format, Number : INTEGER2;
...
OutNumber(Format, Number);
```

COBOL

```
01  Format  COMP.
01  Number  COMP.
    ...
    MONITOR-CALL "OutNumber" USING Format, Number.
```

FORTRAN

```
      INTEGER  Format, Number
      ...
      Monitor_Call('OutNumber', Format, Number)
```

ND-100 and ND-500 | All users | Background programs

35B	OUTNUMBER	IOUT

Continued from previous page...

PLANC

```
INTEGER : Format, Number
...
Monitor_Call('OutNumber', Format, Number)
```

ASSEMBLY-500

```
Format : W  BLOCK  1 %12B=decimal, 10b=octal, 20b=hexadecimal, 2=bitpattern.
Number : W  BLOCK  1
OutNumber : EQU 37B9 + 35B
        ...
        CALLG OutNumber, 2, Format, Number
```

MAC

```
        LDT     FORM      %Format of number to be printed.
        LDA     NUM       %Number to be printed.
        MON     35        %Monitor call OutNumber.
        ...
FORM,   12                %Interpret NUM as a decimal digit.
NUM,    ...
```

ND-100 and ND-500	All users	Background programs

DVOUTS — OUTPUTSTRING — 504B

Writes a string to a device, eg., a terminal or an opened file.

- This is the most efficient way to output strings on the ND-500.
- The maximum string length is 2048B bytes.
- Appendix F contains an ASCII table.

See also OutMessage, OutUpTo8Bytes, Out8Bytes, OutString, OutNumber, OutByte, and InputString.

PARAMETERS

→ Logical device number, eg., a file number. See appendix B. You may use 1 for your own terminal. Use the SINTRAN III open file number if output to a file. The ND-500 open file number will not function.
→ Number of bytes to write.
→ String to be output.

PASCAL

```
DeviceNo, NoOfBytes : LONGINT;
Buff : ARRAY [0..8] OF BITMAP;
...
OutputString(DeviceNo, NoOfBytes, Buff);
```

COBOL

```
01  DeviceNo  COMP.
01  NoOfBytes  COMP.
01  Buff.
    02  array COMP OCCURS 100 TIMES.
    ...
    MONITOR-CALL "OutputString" USING DeviceNo, NoOfBytes, Buff.
```

FORTRAN

```
      INTEGER  DeviceNo, NoOfBytes
      INTEGER  Buff(100)
      ...
      Monitor_Call('OutputString', DeviceNo, NoOfBytes, Buff(1))
```

ND-500 | All users | All programs

504B	OUTPUTSTRING	INOUTS

Continued from previous page...

PLANC

```
INTEGER : DeviceNo, NoOfBytes
BYTE ARRAY : Buff(0:199)
...
Monitor_Call('OutputString', DeviceNo, NoOfBytes, Buff(0))
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
NoOfBytes : W  BLOCK  1
Buff : W  ARRAY 100
Status : BY BLOCK  4000B
OutputString : EQU 37B9 + 504B
        ...
        CALLG OutputString, 3, DeviceNo, NoOfBytes, Buff
        IF K GO Error
        W1 =: Status
        ...
Error,  ...
```

MAC

Not available.

ND-500	All users	All users

OUTST OUTSTRING 162B

Writes a string of characters to a peripheral file, eg., a terminal or a printer.

- You cannot use this monitor call for mass storage files.
- The output buffer of the device may be to small. Then the program waits until the required buffer space becomes available.
- Parameters are fetched and returned through the alternative page table.
- Appendix F contains an ASCII table.

See also OutMessage, OutUpTo8Bytes, Out8Bytes, OutputString, OutNumber, OutByte, and InString.

PARAMETERS

→ Logical device number. See appendix B. You cannot use 1 for your own terminal. Use ExecutionInfo to get its logical device number instead. File numbers are illegal.
→ Character string to be output.
→ Number of characters to be output.
← Status. It has the following meaning:
-1 means error in parameters.
0 in bit 15:14 means OK.
2 in bit 15:14 means that the number of characters is greater than the output buffer of the device. The program has to wait. This does not apply to terminals and communication channels.
3 in bit 15:14 means device error. Bit 7:0 contains the error number. See appendix A.

PASCAL

```
DeviceNo, NoOfBytes, ReturnStatus : INTEGER2;
TextWrite : PACKED ARRAY [0..79] OF CHAR;
...
OutString(DeviceNo, TextWrite, NoOfBytes, ReturnStatus);
```

COBOL

```
01  DevNo  COMP.
01  NoOfBytes  COMP.
01  RetStatus  COMP.
01  TextWrite   PIC X(100).
    ...
    MONITOR-CALL "OutString" USING DevNo, TextWrite, NoOfBytes, RetStatus.
```

FORTRAN

```
      INTEGER  DeviceNo, NoOfBytes, RetStatus
      CHARACTER  TextWrite*80
      ...
      Monitor_Call('OutString', DeviceNo, TextWrite(1:80), NoOfBytes,
     C                  RetStatus)
```

ND-100 and ND-500 All users All programs

162B **OUTSTRING** **OUTST**

Continued from previous page...

PLANC

```
INTEGER : DeviceNo, NoOfBytes, ReturnStatus
BYTES : TextWrite(0:79)
...
Monitor_Call('OutString', DeviceNo, TextWrite, NoOfBytes, ReturnStatus)
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
TextWrite : STRINGDATA  'This is a text.'''
ReturnStatus : W  BLOCK  1
OutString : EQU 37B9 + 162B
        ...
        CALLG OutString, 2, DeviceNo, TextWrite
        IF K GO Error
        W1 =: ReturnStatus              %Status is returned in W1
        ...
Error,  ...
register.
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     162     %Monitor call OutString.
        STA     STAT    %Store status returned.
        ...
STAT,   0
PAR,    DEVNO           %Logical device number.
        TEXT            %String to be written.
        COUNT           %Number of characters to be written.
        ...
DEVNO,  ...
TEXT,   'THIS IS A TEST'
COUNT,  16              %Write 14 characters.
```

ND-100 and ND-500 **All users** **All users**

BBOUT OutUpTo8Bytes 22B

Writes up to 8 characters to a device, eg., a terminal or an internal device.

- You can only use this monitor call for terminals, TADs, internal devices, and synchronous modems.
- The writing terminates when a character with value 0 is found. The 0 byte is not output.
- Appendix F contains an ASCII table.

See also OutMessage, Out8Bytes, OutputString, OutString, OutNumber, OutByte, and InUpTo8Bytes.

PARAMETERS

→ Logical device number. See appendix B. You can use 1 for your own terminal. File numbers are illegal.
→ The 8 characters to be written.
← Standard error code. See appendix A.

PASCAL

```
DeviceNo : INTEGER2;
OutData : BITMAP;
...
OutUpTo8Bytes(DeviceNo, OutData);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNo  COMP.
01  OutData   PIC X(8).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "OutUpTo8Bytes" USING DeviceNo, OutData.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNo
      CHARACTER  OutData*8
      ...
      Monitor_Call('OutUpTo8Bytes', DeviceNo, OutData(1:8))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

22B **OutUpTo8Bytes** RBOUT

Continued from previous page...

PLANC

```
INTEGER : DeviceNo
BYTES : OutData(0:7)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('OutUpTo8Bytes', DeviceNo, OutData)
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
OutData : STRINGDATA 8
ErrCode : W  BLOCK  1
OutUpTo8Bytes : EQU 37B9 + 22B
        ...
        CALLG OutUpTo8Bytes, 2, DeviceNo, OutData
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Logical device address.
        LDD     BYTES    %First 4 bytes to be written.
        LDX     BYTES+2  %Next 2 bytes to be written.
        COPY    SX  DL
        LDX     BYTES+3  %Last 2 bytes to be written.
        MON     22       %Monitor call OutUpTo8Bytes.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
BYTES,  'HELLO!!!'
```

ND-100 and ND-500 | All users | All users

PIOCM — PIOCFUNCTION — 255B

PIOC is a programmable input and output processor. It is mainly used in data communication to handle networks like X.25 and Ethernet. You use this monitor call to control the PIOC from SINTRAN III. This is explained in the PIOC SOFTWARE GUIDE (ND-60.161).

- PIOC is based on an MC 68000 processor. A PLANC compiler is available for it.

See also XMSGFunction and HDLCFunction.

PARAMETERS

→

←

← Standard error code. See appendix A.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 | All users | All programs

255B	PIOCFUNCTION	PIOCM

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Not available.

MAC

See the PIOC SOFTWARE GUIDE (ND-60.161).

ND-100	All users	All users

IPRIV PRIVINSTRUCTION 146B

Executes a privileged machine instruction on the ND-100. Privileged instructions may, for example, turn the paging and interrupt mechanisms on and off.

- The instruction uses the register contents of the calling program. The registers may be changed.

See also IOInstruction.

PARAMETERS

→ The machine instruction as an octal number. For example, specify 150412B for the machine instruction PION. See the ND-100 REFERENCE MANUAL (ND-06.014) for the other privileged instructions.

PASCAL

```
Instruction : INTEGER2;
...
PrivInstruction(Instruction);
```

COBOL

```
01  Instruction  COMP.
    ...
    MONITOR-CALL "PrivInstruction" USING Instruction.
```

FORTRAN

```
      INTEGER  Instruction
      ...
      Monitor_Call('PrivInstruction', Instruction)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

146B PRIVINSTRUCTION IPRIV

Continued from previous page...

PLANC

```
INTEGER : Instruction
...
Monitor_Call('PrivInstruction', Instruction)
```

ASSEMBLY-500

```
TReg : W BLOCK 1
AReg : W BLOCK 1
DReg : W BLOCK 1
XReg : W BLOCK 1
ErrCode : W BLOCK 1
PrivInstruction : EQU 37B9 + 146B
        ...
        CALLG PrivInstruction, 4, TReg, AReg, DReg, XReg
        IF K GO Error
        ...
Error,  W1 =: ErrCode
```

MAC

```
        LDT     INSTR   %Instruction to be executed.
        MON     146     %Monitor call PrivInstruction.
        ...
INSTR,  150404          %Turn off memory management system. Dangerous!
```

A1BDW READADCHANNEL 37B

Reads an analog to digital channel.

PARAMETERS

← Standard error code. See appendix A.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 and ND-500 | All users | All programs

37B **READADCHANNEL** **AIBDN**

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
NoOfChannels : W  BLOCK  1
Channel : W  BLOCK  512
Buffer : W BLOCK  512
ReturnValue : W  BLOCK  1
ReadADChannel : EQU 37B9 + 37B
        ...
        CALLG ReadADChannel, 4, NoOfChannels, Channel, Buffer, ReturnValue
        IF K GO Error
        ...
Error,  ...
```

MAC

```
        MON       37        %Monitor call ReadADChannel.
```

ND-100 and ND-500 **All users** **All users**

BPAGE **READBLOCK** 7B

Reads randomly from a file. You read one block at a time. The file must be opened for random read access.

- The standard block size is 512 bytes. You can change this with SetBlockSize. The first block is number 0.

See also SetStartBlock, SetBlockSize, ReadDiskPage, ReadFromFile, and WriteBlock.

PARAMETERS

→ File number. See OpenFile.
→ Block number.
← Transferred block.
← Standard error code. See appendix A.

PASCAL

```
FileNumber, BlockNo : INTEGER2;
DataDestination : ARRAY [0..15] OF BITMAP;
...
ReadBlock(FileNumber, BlockNo, DataDestination);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNumber  COMP.
01  BlockNo  COMP.
01  DataDestination.
    02  array COMP OCCURS 256 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "ReadBlock" USING FileNumber, BlockNo, DataDestination.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNumber, BlockNo
      INTEGER  DataDestination(256)
      ...
      Monitor_Call('ReadBlock', FileNumber, BlockNo, DataDestination(1))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 All users All programs

7B **READBLOCK** RPAGE

Continued from previous page...

PLANC

```
INTEGER : FileNumber, BlockNo
BYTE ARRAY : DataDestination(0:511)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('ReadBlock', FileNumber, BlockNo, DataDestination(0))
```

ASSEMBLY-500

Not available.

MAC

```
        LDT     FILNO    %File number returned from earlier open.
        LDA     BLKNO    %Block number.
        LDX     (BUFF    %Address of buffer to receive block read.
        MON     7        %Monitor call ReadBlock.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
BLKNO,  ...
BUFF,   0
*+400/                   %Make a buffer of 256 words.
```

ND-100 All users All users

RDPAG — READDISKPAGE — 270B

Reads one or more directory pages. Any page can be read.

- The directory must be reserved with ReserveDir.

See also WriteDiskPage.

PARAMETERS

→ Directory index. See GetDirUserIndexes.
→ Buffer to receive pages.
→ Address of the destination pages on the disk.
→ Number of pages to transfer. Each page is 2048 bytes.

PASCAL

```
DirIndex, NoOfPages : INTEGER2;
PageAddr : LONGINT;
Buffer : ARRAY [0..63] OF BITMAP;
...
ReadDiskPage(DirIndex, Buffer, PageAddr, NoOfPages);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DirIndex  COMP.
01  NoOfPages  COMP.
01  PageAddr  COMP PIC S9(10).
01  Buffer.
    02  array COMP OCCURS 1024 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "ReadDiskPage" USING DirIndex, Buffer, PageAddr, NoOfPages.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DirIndex, NoOfPages
      INTEGER*4  PageAddr
      INTEGER  Buffer(1024)
      ...
      Monitor_Call('ReadDiskPage', DirIndex, Buffer(1), PageAddr, NoOfPages)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | **User RT and user SYSTEM** | **Background programs**

270B **READDISKPAGE** **RDPAG**

Continued from previous page...

PLANC

```
INTEGER : DirIndex, NoOfPages
INTEGER4 : PageAddr
BYTE ARRAY : Buffer(0:2047)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('ReadDiskPage', DirIndex, Buffer(0), PageAddr, NoOfPages)
```

ASSEMBLY-500

```
DirIndex : W  BLOCK  1
PageAddr : W  BLOCK  1
NoOfPages : W  BLOCK  1
Buffer : H  BLOCK  1024   %Must start on an even byte addresss.
ErrCode : W  BLOCK  1
ReadDiskPage : EQU 37B9 + 270B
        ...
        CALLG ReadDiskPage, 4, DirIndex, Buffer, PageAddr, NoOfPages
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DIRIX    %Directory index.
        LDX     (BUFF    %Address of buffer to receive data read.
        LDA     COUNT    %Number of pages to transfer.
        COPY    SA  DD
        LDA     (PAGNO   %Address of double word with disk page address.
        MON     270      %Monitor call ReadDiskPage.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
DIRIX,  ...
BUFF,   0
*+4000/                  %Make a buffer of 2048 words.
PAGNO,  ...              %A double word.
        ...              %
COUNT,  2                %Read 2 pages of 1024 words.
```

RPFILE — READFROMFILE — 117B

Reads any number of bytes from a file. The read operation must start at the beginning of a block. The file must be opened for random read access.

- The standard block size is 512 bytes. You can change this with SetBlockSize. The first block is number 0.
- You may use access code D for direct transfer. Then the block size must be a multiple of the page size. The number of bytes to transfer must be a multiple of the block size.
- Peripheral files are always read sequentially.
- Data transfer across segment or RT common limits is illegal.

See also SetStartBlock, SetBlockSize, ReadDiskPage, ReadBlock, and WriteBlock.

PARAMETERS

→ File number. See OpenFile.
→ Wait flag. Use 0 to suspend the program until the transfer is completed. Other values makes the program continue. You may check that the transfer is completed by AwaitFileTransfer.
← Transferred data.
→ Block number to start the read operation. Use -1 to read the next block.
→ Number of bytes to read.
← Standard error code. See appendix A.

PASCAL

```
FileNo, WaitFlag, BlockNo : INTEGER2;
Buff : ARRAY [0..15] OF BITMAP;
NoOfBytes : LONGINT;
...
ReadFromFile(FileNo, WaitFlag, Buff, BlockNo, NoOfBytes);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNo  COMP.
01  WaitFlag  COMP.
01  Buff.
    02  array COMP OCCURS 256 TIMES.
01  BlockNo  COMP.
01  NoOfBytes  COMP PIC S9(10).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "ReadFromFile" USING FileNo, WaitFlag, Buff,
                                      BlockNo, NoOfBytes.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNo, WaitFlag, BlockNo
      INTEGER  Buff(256)
      INTEGER*4  NoOfBytes
      ...
      Monitor_Call('ReadFromFile', FileNo, WaitFlag, Buff(1),
     C                             BlockNo, NoOfBytes)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

117B **READFROMFILE** **RFILE**

Continued from previous page...

PLANC

```
INTEGER : FileNo, RetFlag, BlockNo
BYTE ARRAY : Buff(0:511)
INTEGER4 : NoOfBytes
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('ReadFromFile', FileNo, RetFlag, Buff(0), BlockNo, NoOfBytes)
```

ASSEMBLY-500

```
FileNo : W  BLOCK  1
WaitFlag : W  BLOCK  1
Buff : W  BLOCK  256      %Must start on an even byte address.
BlockNo : W  BLOCK  1
NoOfBytes : W  BLOCK  1
ErrCode : W  BLOCK  1
ReadFromFile : EQU 37B9 + 117B
        ...
        CALLG ReadFromFile, 5, FileNo, WaitFlag, Buff, BlockNo, NoOfBytes
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDA     (PAR      %Load register A with address of parameter list.
        MON     117       %Monitor call ReadFromFile.
        JAF     ERROR     %Handle error if register A is non-zreo.
        ...
ERROR,  ...               %Error number in register A.
        ...
PAR,    FILNO             %File number returned from earlier call to
OpenFile.
        RETUR             %Return flag.
        BUFF              %Buffer to receive data.
        BLKNO             %Block number in file where data starts.
        COUNT             %Number of words to be read.
        ...
FILNO,  ...
RETUR,  0
BUFF,   0
*+400/                    %Make a buffer of 256 words.
BLKNO,  ...
COUNT,  ...
```

ND-100 and ND-500 **All users** **All users**

ROBJE READOBJECTENTRY 41B

Gets information about an opened file. An object entry describes each file. It contains the file name, the access rights, the date last opened for read and write, the size, and more. See the file system description in the SINTRAN III SYSTEM SUPERVISOR (ND-30.003). You specify the file number.

- There is one object entry for each version of a file.
- The device number location in the object entry contains the logical device number and the unit number where the mass storage file resides. The logical device number is placed in bit 11-0. The unit number in bit 15-12. The location contains the logical device number for peripheral files.

See also GetObjectEntry and SetObjectEntry.

PARAMETERS

→ The file number. See OpenFile.
← The 64 byte object entry. See appendix C.
← Standard error code. See appendix A.

PASCAL

```
FileNumber : INTEGER2;
Buff : ARRAY [0..1] OF BITMAP;
...
ReadObjectEntry(FileNumber, Buff);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNumber  COMP.
01  Buff.
    02  array COMP OCCURS 32 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "ReadObjectEntry" USING FileNumber, Buff.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNumber
      INTEGER  Buff(32)
      ...
      Monitor_Call('ReadObjectEntry', FileNumber, Buff(1))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

41B **READOBJECTENTRY** **ROBJE**

Continued from previous page...

PLANC

```
INTEGER : FileNumber
BYTE ARRAY : Buff(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('ReadObjectEntry', FileNumber, Buff(0))
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
Buff : H  BLOCK  40B
ErrCode : W  BLOCK  1
ReadObjectEntry : EQU 37B9 + 41B
        ...
        CALLG ReadObjectEntry, 2, FileNumber, Buff
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     FILNO    %File number returned from earlier open.
        LDA     (BUFF    %Address of buffer to receive object entry.
        MON     41       %Monitor call ReadObjectEntry.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
BUFF,   0
*+40/                    %Make a 32 words large buffer.
```

ND-100 and ND-500 **All users** **All users**

RDISK — READSCRATCHFILE — 5B

Reads randomly from the scratch file. One block is transferred. There is one scratch file connected to each terminal. It is opened for random read and write access when you log in. Its file number is 100B.

- The standard block size is 512 bytes. You can change this with SetBlockSize. The first block is number 0.

See also ReadBlock and ReadFromFile. ReadFromFile is the most efficient monitor call.

PARAMETERS

→ Block number to start the reading.
← The transferred data.
← Standard error code. See appendix A.

PASCAL

```
BlockNumber : INTEGER2;
DataDestination : BITMAP;
...
ReadScratchFile(BlockNumber, DataDestination);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  BlockNumber  COMP.
01  DataDestination.
    02  array COMP OCCURS 256 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "ReadScratchFile" USING BlockNumber, DataDestination.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  BlockNumber
      INTEGER  DataDestination(256)
      ...
      Monitor_Call('ReadScratchFile', BlockNumber, DataDestination(1))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 | All users | Background programs

5B **READSCRATCHFILE** **RDISK**

Continued from previous page...

PLANC

```
INTEGER : BlockNumber
BYTE ARRAY : DataDestination(0:511)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('ReadScratchFile', BlockNumber, DataDestination(0))
```

ASSEMBLY-500

Not available.

MAC

```
        LDT     BLKNO    %Block number to be read.
        LDX     (BUFF    %Address of buffer to receive block read.
        MON     5        %Monitor call ReadScratchFile.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
BLKNO,  ...
BUFF,   0
*+400/                   %Make a buffer of 256 words, 1 block.
```

ND-100 **All users** **Background programs**

SBERN — REENTRANTSEGMENT — 212B

Connects a reentrant segment to your two current segments. All modified pages of your current segments are written to the segment file before the reentrant segment is fetched. This is almost equivalent to SaveSegment followed by AttachSegment. However, ReentrantSegment is more efficient. Only the modified pages overlapping the reentrant segment are written back.

See also AttachSegment.

PARAMETERS

→ Segment number to attach.

PASCAL

```
SegmentNumber : INTEGER2;
...
ReentrantSegment(SegmentNumber);
```

COBOL

```
01  SegmentNumber  COMP.
    ...
    MONITOR-CALL "ReentrantSegment" USING SegmentNumber.
```

FORTRAN

```
      INTEGER  SegmentNumber
      ...
      Monitor_Call('ReentrantSegment', SegmentNumber)
```

ND-100 | All users | All programs

212B **REENTRANTSEGMENT** **SBEEN**

Continued from previous page...

PLANC

```
INTEGER : SegmentNumber
...
Monitor_Call('ReentrantSegment', SegmentNumber)
```

ASSEMBLY-500

Not available.

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     212     %Monitor call ReentrantSegment.
        ...
PAR,    SEGNO           %Segment number.
        ...
SEGNO,  ...
```

ND-100 **All users** **All users**

BLDIR — RELEASEDIR — 247B

Releases a directory. The directory must have been reserved with ReserveDir.

See also ReleaseResource.

PARAMETERS

→ Directory index. Use @LIST-DIRECTORIES to find the directory index.
← Standard error code. See appendix A.

PASCAL

```
DirectoryIndex : INTEGER2;
...
RelDirectory(DirectoryIndex);      [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DirectoryIndex  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "ReleaseDir" USING DirectoryIndex.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DirectoryIndex
      ...
      Monitor_Call('ReleaseDir', DirectoryIndex)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

247B **RELEASEDIR** **RLDIR**

Continued from previous page...

PLANC

```
INTEGER : DirectoryIndex
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('ReleaseDir', DirectoryIndex)
```

ASSEMBLY-500

```
DirectoryIndex : W  BLOCK  1
ErrCode : W  BLOCK  1
ReleaseDir : EQU 37B9 + 247B
        ...
        CALLG ReleaseDir, 1, DirectoryIndex
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DIRIX    %Directory index.
        MON     247      %Monitor call ReleaseDir.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
DIRIX,  ...
```

ND-100 and ND-500 **All users** **All users**

RELEASERESOURCE 123B

Releases a reserved device or file. The resource can then be used by another program. You reserve a device or opened file with ReserveResource. Some devices, eg., terminals, have both an input and output part. You can only release one part with each ReleaseResource.

- A normal termination of an RT program release all resources.
- Reserve the device with ReserveResource or ForceReserve.
- CloseFile or @CLOSE-FILE releases reserved files.

See also ForceRelease, ReserveResource, ReleaseDir, @RESRV, @RELEASE-FILE, and @RELEASE-DEVICE-UNIT.

PARAMETERS

→ Logical device number. See appendix B.
→ Input or output flag. Use 0 for the input part and 1 for the output part.

PASCAL

```
DeviceNumber, IOFlag : INTEGER2;
...
ReleaseResource(DeviceNumber, IOFlag);
```

COBOL

```
01  DeviceNumber  COMP.
01  IOFlag  COMP.
    ...
    MONITOR-CALL "ReleaseResource" USING DeviceNumber, IOFlag.
```

FORTRAN

```
      INTEGER  DeviceNumber, IOFlag
      ...
      Monitor_Call('ReleaseResource', DeviceNumber, IOFlag)
```

ND-100 and ND-500 | All users | All programs

123B **RELEASERESOURCE** **RELES**

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, IOFlag
...
Monitor_Call('ReleaseResource', DeviceNumber, IOFlag)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
IOFlag : W  BLOCK  1
ReleaseResource : EQU 37B9 + 123B
        ...
        CALLG ReleaseResource, 2, DeviceNumber, IOFlag
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     123     %Monitor call ReleaseResource.
        ...
PAR,    DEVNO           %Logical device number.
        IOF             %Input/output flag.
        ...
DEVNO,  ...
IOF,    ...
```

ND-100 and ND-500 **All users** **All users**

RENAMEFILE 2328

Renames a file. You need directory access to the file.

- All versions of a file are renamed unless you specify a version number.

See also @RENAME-FILE.

PARAMETERS

→ Old file name.
→ New file name with file type, eg., ADDRESS-LIST:TEXT. Do not use the directory name, the user name, or the version number. You may change the file type only. For example, specify :SYMB only. Include the colon.
← Standard error code. See appendix A.

PASCAL

```
OldFileName, NewFileName : PACKED ARRAY [0..63] OF CHAR;
...
RenameFile(OldFileName, NewFileName);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  OldFileName    PIC X(64).
01  NewFileName    PIC X(64).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "RenameFile" USING OldFileName, NewFileName.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      CHARACTER  OldFileName*64, NewFileName*64
      ...
      Monitor_Call('RenameFile', OldFileName(1:64), NewFileName(1:64))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

232B **RENAMEFILE** **MENP1**

Continued from previous page...

PLANC

```
BYTES : OldFileName(0:63), NewFileName(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('RenameFile', OldFileName, NewFileName)
```

ASSEMBLY-500

```
OldFileName : STRINGDATA  'TEXT:TEXT'''
NewFileName : STRINGDATA  'EXAMPLE:SYMB'''
ErrCode : W  BLOCK  1
RenameFile : EQU 37B9 + 232B
        ...
        CALLG RenameFile, 2, OldFileName, NewFileName
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (OLDFI   %Address of string with old file name.
        LDA     (NEWFI   %Address of string with new file name.
        MON     232      %Monitor call RenameFile.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
OLDFI,  'TEXT:TEXT'      %Change name of file TEXT:TEXT to EXAMPLE:SYMB.
NEWFI,  'EXAMPLE:SYMB'
```

ND-100 and ND-500 **All users** **All users**

WHODEV — RESERVATIONINFO — 140B

Checks that a device is not reserved. If it is reserved, you will receive information about which RT program that reserves it.

- Some devices has both an input and an output part. You have to use ReservationInfo for each part.

See also ReserveResource, ReleaseResource, and @LIST-DEVICE.

PARAMETERS

→ Logical device number. See appendix B.
→ Input or output flag. 0 means the input part. 1 means the output part.
← RT description address of reserving RT program. 0 means not reserved.
Errors in the parameters return -1.

PASCAL

```
DeviceNumber, IOFlag, ReturnValue : INTEGER2;
...
ReservationInfo(DeviceNumber, IOFlag, ReturnValue);
```

COBOL

```
01  DeviceNumber  COMP.
01  IOFlag  COMP.
01  ReturnValue  COMP.
    ...
    MONITOR-CALL "ReservationInfo" USING DeviceNumber, IOFlag, ReturnValue.
```

FORTRAN

```
      INTEGER  DeviceNo, IOFlag, ReturnValue
      ...
      Monitor_Call('ReservationInfo', DeviceNo, IOFlag, ReturnValue)
```

ND-100 and ND-500 | All users | All programs

140B	RESERVATIONINFO	WHDKV

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, IOFlag, ReturnValue
...
Monitor_Call('ReservationInfo', DeviceNumber, IOFlag, ReturnValue)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
IOFlag : W  BLOCK  1
ReturnValue : W  BLOCK  1
ReservationInfo : EQU 37B9 + 140B
        ...
        CALLG ReservationInfo, 2, DeviceNumber, IOFlag
        W1 =: ReturnValue               %Result is returned in W1 register.
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     140     %Monitor call ReservationInfo.
        STA     STAT    %Store status returned.
        AAA     -1      %Test if -1. JAN not applicable because of RT
        JAZ     ERROR   %            addresses above 100000.
        ...
ERROR,  ...
        ...
STAT,   0               %RT description address if any.
PAR,    DEVNO           %Logical device number.
        IOF             %Input or output flag.
DEVNO,  ...
IOF,    ...
```

ND-100 and ND-500	All users	All users

RESERVEDIR — RESDIR — 246B

Reserves a directory for special use. The directory must be entered. Other users will not be able to open files on a reserved directory.

- All files in the directory must be closed.
- Only user RT and the current user may be logged in if main directory.
- Use ReleaseDir to release the directory.

See also ReleaseDir.

PARAMETERS

→ Directory index. Use @LIST-DIRECTORIES to find the directory index.
← Standard error code. See appendix A.

PASCAL

```
DirectoryIndex : INTEGER2;
...
ResDirectory(DirectoryIndex);      [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DirectoryIndex  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "ReserveDir" USING DirectoryIndex.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DirectoryIndex
      ...
      Monitor_Call('ReserveDir', DirectoryIndex)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | User RT and user SYSTEM | All programs

246B **RESERVEDIR** **REDIR**

Continued from previous page...

PLANC

```
INTEGER : DirectoryIndex
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('ReserveDir', DirectoryIndex)
```

ASSEMBLY-500

```
DirectoryIndex : W  BLOCK  1
ErrCode : W  BLOCK  1
ReserveDir : EQU 37B9 + 246B
        ...
        CALLG ReserveDir, 1, DirectoryIndex
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DIRIX    %Directory index.
        MON     246      %Monitor call ReserveDir.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
DIRIX,  ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **All users**

RESERV — RESERVERESOURCE — 122B

Reserves a device or file for your program only. You release it with ReleaseResource. Some devices, eg., terminals, have both an input and output part. You can only reserve one part with each ReleaseResource.

- A normal termination of an RT program releases all resources.
- Release the device with ReleaseResource or ForceRelease.
- A background program does not release a resource when you press the ESCAPE key.

See also ForceReserve, ReserveDir, @RESRV, @RESERVE-FILE, and @RESERVE-DEVICE-UNIT.

PARAMETERS

→ Logical device number. See appendix B.
→ Input or output flag. Use 0 for the input part and 1 for the output part.
← Wait flag. Use 0 to make the program wait if the resource is already reserved. Use 1 to return a status value.
→ Status. Only used if the wait flag is 1. A negative value is returned if the resource is already reserved.

PASCAL

```
DeviceNo, IOFlag, WaitFlag, ReturnStatus : INTEGER2;
...
ReserveResource(DeviceNo, IOFlag, WaitFlag, ReturnStatus);
```

COBOL

```
01  DevNo  COMP.
01  IOFlag  COMP.
01  WaitFlag  COMP.
01  RetStatus  COMP.
    ...
    MONITOR-CALL "ReserveResource" USING DevNo, IOFlag, WaitFlag, RetStatus.
```

FORTRAN

```
      INTEGER  DevNo, IOFlag, WaitFlag, ReturnStatus
      ...
      Monitor_Call('ReserveResource', DevNo, IOFlag, WaitFlag, ReturnStatus)
```

ND-100 and ND-500 | All users | All programs

122B **RESERVERESOURCE** **RESRV**

Continued from previous page...

PLANC

```
INTEGER : DeviceNo, IOFlag, WaitFlag, ReturnStatus
...
Monitor_Call('ReserveResource', DeviceNo, IOFlag, WaitFlag, ReturnStatus)
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
IOFlag : W  BLOCK  1
WaitFlag : W  BLOCK  1
Status : W  BLOCK  1
ReserveResource : EQU 37B9 + 122B
        ...
        CALLG ReserveResource, 3, DeviceNo, IOFlag, WaitFlag
        W1 =: Status
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     123     %Monitor call ReserveResource.
        ...
PAR,    DEVNO           %Logical device number.
        IOF             %Input/output flag.
        WFLAG           %Wait flag
        STAT            %Status
DEVNO,  ...
IOF,    ...
WFLAG,  ...
STAT,   ...
```

ND-100 and ND-500 **All users** **All users**

SAVEND500SEGMENT 416B

Writes all modified pages of a segment back to the disk.

- Not allowed when fixed in memory.

See also SaveSegment.

PARAMETERS

→ Logical segment number in the domain. If 0, the segment number is retrieved from the parameter address.
→ Start page in the segment.
→ Last page in the segment.
← Standard error code. See appendix A.

PASCAL

```
LogSegmentNo, FirstPage, LastPage : LONGINT;
...
SaveND500Segment(LogicalSegmentNo, FirstPage, LastPage);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  LogSegmentNo  COMP.
01  FirstPage  COMP.
01  LastPage  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "SaveND500Segment" USING LogSegmentNo, FirstPage,
                                          LastPage.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  LogSegmentNo, FirstPage, LastPage
      ...
      Monitor_Call('SaveND500Segment', LogSegmentNo, FirstPage, LastPage)
      IF (ErrCode .NE. 0) THEN ...
```

ND-500 | All users | All programs

416B **SAVEND500SEGMENT** **WSEGM**

Continued from previous page...

PLANC

```
INTEGER : LogSegmentNo, FirstPage, LastPage
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('SaveND500Segment', LogSegmentNo, FirstPage, LastPage)
```

ASSEMBLY-500

```
LogSegmentNo : W  BLOCK  1
FirstPage : W  BLOCK  1
LastPage : W  BLOCK  1
ErrCode : W  BLOCK  1
SaveND500Segment : EQU 37B9 + 416B
        ...
        CALLG SaveND500Segment, 3, LogSegmentNo, FirstPage, LastPage
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

Not available.

ND-500 **All users** **All users**

WSBG SAVESEGMENT 164B

Saves a segment in the ND-100. All pages in physical memory which have been changed, are written back to the disk.

See also SaveND500Segment.

PARAMETERS

→ Segment number.

PASCAL

```
SegmentNumber : INTEGER2;
...
SaveSegment(SegmentNumber);
```

COBOL

```
01  SegmentNumber  COMP.
    ...
    MONITOR-CALL "SaveSegment" USING SegmentNumber.
```

FORTRAN

```
      INTEGER  SegmentNumber
      ...
      Monitor_Call('SaveSegment', SegmentNumber)
```

ND-100 and ND-500 | All users | All programs

164B **SAVESEGMENT** **WSEG**

Continued from previous page...

PLANC

```
INTEGER : SegmentNumber
...
Monitor_Call('SaveSegment', SegmentNumber)
```

ASSEMBLY-500

```
SegmentNumber : W  BLOCK  1
SaveSegment : EQU 37B9 + 164B
        ...
        CALLG SaveSegment, 1, SegmentNumber
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     164     %Monitor call SaveSegment.
        ...
PAR,    SEGNO           %Segment number.
        ...
SEGNO,  ...
```

ND-100 and ND-500 **All users** **All users**

SCROP ScratchOpen 2358

Opens a file as a scratch file. A maximum of 64 pages of the file is kept when you close the file. Use SET-CLOSED-FILE-SIZE in the SINTRAN-SERVICE-PROGRAM to change this.

- The file is closed as any other opened file.

See also OpenFile, SetPermanentOpen, DirectOpen, CloseFile, and @SCRATCH-OPEN.

PARAMETERS

← File number. Monitor calls for input and output use this number.
→ Access code. The legal values are shown below.
0: Sequential write.
1: Sequential read.
2: Random read or write.
3: Random read only.
4: Sequential read or write.
5: Sequential write append.
6: Random read or write common on contiguous files.
7: Random read common on contiguous files.
8: Random read or write on contiguous files. Direct transfer for ReadFromFile, WriteToFile and DeviceFunction in RT programs.
9: Random read, write append for WriteToFile and ReadFromFile.
→ File name.
→ Default file type. Do not include the colon.
← Standard error code. See appendix A.

PASCAL

```
FileNo, AccessCode : INTEGER2;
FileName : PACKED ARRAY [0..63] OF CHAR;
FileType : PACKED ARRAY [0..3] OF CHAR;
...
ScratchOpen(FileNo, AccessCode, FileName, FileType);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNo  COMP.
01  AccessCode  COMP.
01  FileName   PIC X(64).
01  FileType   PIC X(4).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "ScratchOpen" USING FileNo, AccessCode, FileName, FileType.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNo, AccessCode
      CHARACTER  FileName*64, FileType*4
      ...
      Monitor_Call('ScratchOpen', FileNo, AccessCode, FileName(1:64),
     C                  FileType(1:4))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

235B **SCRATCHOPEN** **SCBOP**

Continued from previous page...

PLANC

```
INTEGER : FileNo, AccessCode
BYTES : FileName(0:63), FileType(0:3)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('ScratchOpen', FileNo, AccessCode, FileName, FileType)
```

ASSEMBLY-500

```
FileNo : W  BLOCK  1
AccessCode : W  BLOCK  1
FileName : STRINGDATA  'EXAMPLE'''
FileType : STRINGDATA  'SYMB'''
ErrCode : W  BLOCK  1
ScratchOpen : EQU 37B9 + 235B
        ...
        CALLG ScratchOpen, 4, FileNo, AccessCode, FileName, FileType
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (FILE    %Address of file name string.
        LDA     (TYPE    %Address of default file type string.
        LDT     ACODE    %Access code.
        MON     235      %Monitor call ScratchOpen.
        JMP     ERROR    %Error return from monitor call.
        STA     FILNO    %Normal return, store file number.
        ...
ERROR,  ...              %Error number in register A.
        ...
FILNO,  0
ACODE,  ...
FILE,   'EXAMPLE'        %Open EXAMPLE:SYMB as a scratch file.
TYPE,   'SYMB'
```

ND-100 and ND-500 **All users** **All users**

SPLEE SEGMENTOVERLAY 323B

Used to build multisegment programs in the ND-100. It is mainly for internal use. A new reentrant segment and two address areas in this segment are specified.

- The building of multisegment programs is taken care of automatically by the BRF-LINKER. See the BRF-LINKER USER MANUAL (ND-60.196).

PARAMETERS

→ Segment number. The segment must be reentrant.
→ Start of page area 1. Normally in the program bank.
→ End of page area 1.
→ Start of page area 2. Normally in the data bank.
→ End of page area 2.
→ Clear flag. If not 0, the earlier specified overlay areas are cleared. Use 0 the first time SegmentOverlay is called.
← Standard error code. See appendix A.

PASCAL

```
SegmentNo, Page1A1, NoPageA1, Page1A2, NoPageA2, ClearFlag : INTEGER2;
...
SegmentOverlay(SegmentNo, Page1A1, NoPageA1, Page1A2, NoPageA2, ClearFlag);
```

COBOL

```
01  SegmentNo  COMP.
01  Page1A1  COMP.
01  NoPageA1  COMP.
01  Page1A2  COMP.
01  NoPageA2  COMP.
01  ClearFlag  COMP.
    ...
    MONITOR-CALL "SegmentOverlay" USING SegmentNo, Page1A1, NoPageA1,
                                        Page1A2, NoPageA2, ClearFlag.
```

FORTRAN

```
      INTEGER  SegmentNo, Page1A1, NoPageA1, Page1A2, NoPageA2, ClearFlag
      ...
      Monitor_Call('SegmentOverlay', SegmentNo, Page1A1, NoPageA1,
     C                               Page1A2, NoPageA2, ClearFlag)
```

ND-100 All users Background programs

323B **SEGMENTOVERLAY** SPLIB

Continued from previous page...

PLANC

```
INTEGER : SegmentNo, Page1A1, NoPageA1, Page1A2, NoPageA2, ClearFlag
...
Monitor_Call('SegmentOverlay', SegmentNo, Page1A1, NoPageA1, &
                               Page1A2, NoPageA2, ClearFlag)
```

ASSEMBLY-500

Not available.

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     323     %Monitor call SegmentOverlay.
        ...
PAR,    SEGNO           %Segment number.
        AREA1           %First page in area 1.
        NUM1            %Number of pages in area 1.
        AREA2           %First page in area 2.
        NUM2            %Number of pages in area 2.
        CLEAR           %Clear flag.
        ...
SEGNO,  ...
AREA1,  ...
NUM1,   ...
AREA2,  ...
NUM2,   ...
CLEAR,  ...
```

ND-100 All users Background programs

ENTSEG SEGMENTTOPAGETABLE 157B

Enters a routine as a direct task or as a device driver. Such routines are connected to the interrupt system. They are loaded with the RT LOADER or by DMAC.

- The segment where the routine resides must be fixed in memory. See FixScattered.

See also @ENTSEG.

PARAMETERS

→ Segment number where the routine resides.
→ Page table to use for the segment. In practice, this should be 3. For SINTRAN III VSX, version K, this should be 17.
→ The interrupt level where the direct task should run. You must specify one of the free levels 2B, 6B, 7B, 10B or 11B. Do not use level 2B on SINTRAN III VSX, version K.
→ Start address of the routine.

PASCAL

```
SegmentNo, PageTable, InterruptLevel, StartAddr : INTEGER2;
...       [Note routine name.]
EnterSegment(SegmentNo, PageTable, InterruptLevel);
```

COBOL

```
01  SegmentNo  COMP.
01  PageTable  COMP.
01  InterruptLevel  COMP.
01  StartAddress  COMP.
    ...
    MONITOR-CALL "SegmentToPageTable" USING SegmentNo, PageTable,
                                            InterruptLevel, StartAddress.
```

FORTRAN

```
      INTEGER  SegmentNo, PageTable
      INTEGER  InterLevel,StartAddress
      ...
      Monitor_Call('SegmentToPageTable', SegmentNo, PageTable,
     C                   InterLevel,StartAddress)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

157B **SEGMENTTOPAGETABLE** **BMTSEG**

Continued from previous page...

PLANC

```
INTEGER : SegNo, PageTable, InterruptLevel,StartAddress
...
Monitor_Call('SegmentToPageTable', SegNo, PageTable,&
                InterruptLevel, StartAddress)
```

ASSEMBLY-500

```
SegNo : W  BLOCK  1
PageTable : W  BLOCK  1
InterLevel : W  BLOCK  1
StartAddr : W  BLOCK  1
SegmentToPageTable : EQU 37B9 + 157B
        ...
        CALLG SegmentToPageTable, 4, SegNo, PageTable, InterLevel, StartAddr
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     157     %Monitor call SegmentToPageTable.
        ...
PAR,    SEG             %The segment where the routine resides.
        PAGE            %Page table to be used.
        INTR            %Interupt level where the direct task will run.
        ENTRY           %Entry point, start address of direct task.
        ...
SEG,    ...
PAGE,   ...
INTR,   ...
ENTRY,  ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

MAPSIB — SENDSIBASMESSAGE — 304B

Sends a request to the SIBAS database system. SIBAS is activated and retuns an answer.

- This is a special monitor call for SIBAS.
- RT common is used for communication if SIBAS runs on the ND-100.

See also GetSIBASMessage, SIBASFunction, and AnswerSIBAS.

PARAMETERS

→ SIBAS number.
→ Message to be sent to SIBAS.
← Message returned from SIBAS.
← Standard error code. See appendix A.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

304B **SENDSIBASMESSAGE** **MAPSIB**

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
SibasNo : W  BLOCK  1
MsgToSibas : W  BLOCK  256
AnswerBuffer : W BLOCK 256
ErrCode : W  BLOCK 1
SendSIBASMessage : EQU 37B + 304B
        ...
        CALLG SendSIBASMessage, 3, SibasNo, MsgToSibas, AnswerBuffer
        IF K GO Error
        ...
Error,  W1 =: ErrCode
```

MAC

```
        LDT     SIBNO    %Load register T with SIBAS number.
        LDX     MESG     %Load register X with address of message to SIBAS.
        LDA     BUFF     %Address of message returned from SIBAS.
        COPY    SA  DD
        MON     304      %Monitor call SendSIBASMessage.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
SIBNO,  ...              %SIBAS number (0-5).
MESG,   ...

BUFF,   0
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

SETBS — SETBLOCKSIZE — 76B

Sets the block size of an opened file. Monitor calls which read or write randomly to a file operates on blocks. See ReadFromFile and WriteToFile.

- The standard block size is 512 bytes. This block size is set when the file is opened.
- The block size is reset when the file is closed.
- Factors of 2048 bytes are the most efficient block sizes.

See also SetStartBlock, ReadFromFile, WriteToFile, and @SET-BLOCK-SIZE.

PARAMETERS

→ File number. See OpenFile.
→ Block size in bytes. It must be an even number.
← Standard error code. See appendix A.

PASCAL

```
FileNumber : INTEGER2;
BlockSize : LONGINT;
...
SetBlockSize(FileNumber, BlockSize);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNumber  COMP.
01  BlockSize  COMP PIC S9(10).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "SetBlockSize" USING FileNumber, BlockSize.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNumber
      INTEGER*4  BlockSize
      ...
      Monitor_Call('SetBlockSize', FileNumber, BlockSize)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

76B **SETBLOCKSIZE** **SETBS**

Continued from previous page...

PLANC

```
INTEGER : FileNumber
INTEGER4 : BlockSize
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('SetBlockSize', FileNumber, BlockSize)
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
BlockSize : W  BLOCK  1
ErrCode : W  BLOCK  1
SetBlockSize : EQU 37B9 + 76B
        ...
        CALLG SetBlockSize, 2, FileNumber, BlockSize
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     FILNO    %File number returned from earlier open.
        LDA     SIZE     %Block size in words.
        MON     76       %Monitor call SetBlockSize.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
SIZE,   ...              %New block size.
```

ND-100 and ND-500 **All users** **All users**

BRKM SetBreak 4B

Sets the break characters for a terminal. Normally, a program waits for input. When a break character is typed, the program restarts. For example, most subsystems restarts when you press the RETURN-key after a command. The subsystems have defined the RETURN-key as a break character.

- SINTRAN III has some predefined break tables.
- You may define your own break table. This is a 256 bit array where each bit represents an ASCII character. Use 1 for the characters you want as break characters. The ability to define your own break tables is optional in older versions of SINTRAN III.

See also SetEcho.

PARAMETERS

→ Logical device number. See appendix B. Only used by RT programs. Your own terminal is always selected for background programs.
→ Break table. Negative values gives break on no characters.
0: Break on all characters.
1: Break on control characters, ie., ASCII values less than 32 and DEL. This is the default.
2: Special break table used by the MAC assembler.
3-6: System defined break strategy.
7: User defined break table. See the next parameter.
8: Last user defined break table.
9: Change the maximum numbers of characters before break only.
→ User defined break table, ie., 256 bits which represent the ASCII characters.
→ Maximum number of characters before break. Not used by break table 0-2.

PASCAL

```
DeviceNo, BreakStrategy, NoOfChar : INTEGER2;
Table : BITMAP;
...
SetBreak(DeviceNo, BreakStrategy, Table, NoOfChar);
```

COBOL

```
01  DeviceNo  COMP.
01  BreakStrategy  COMP.
01  NoOfChar  COMP.
01  Table.
    02  array COMP OCCURS 8 TIMES.
    ...
    MONITOR-CALL "SetBreak" USING DeviceNo, BreakStrategy, Table, NoOfChar.
```

FORTRAN

```
      INTEGER  DeviceNo, Strategy, NoOfChar
      INTEGER  Table(8)
      ...
      Monitor_Call('SetBreak', DeviceNo, Strategy, Table(1), NoOfChar)
```

ND-100 and ND-500 | All users | All programs

4B **SETBREAK** BEGIN

Continued from previous page...

PLANC

```
INTEGER : DeviceNo, Strategy, NoOfChar
BYTE ARRAY : Table(0:15)
...
Monitor_Call('SetBreak', DeviceNo, Strategy, Table(0), NoOfChar)
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
BreakStrategy : W  BLOCK  1
NoOfChar : W  BLOCK  1
Table : H  BLOCK 8
SetBreak : EQU 37B9 + 4B
        ...
        CALLG SetBreak, 4, DeviceNo, BreakStrategy, Table, NoOfChar
```

MAC

```
        LDT     DEVNO    %Logical device number.
        LDA     NOCHR    %Number of characters on input before break,
        COPY    SA  DD   %   only if STRAT greater or equal to 3.
        LDX     (TABLE   %Address of 8 word large bit map.
        LDA     STRAT    %Break strategy.
        MON     4        %Monitor call SetBreak.
        ...
DEVNO,  ...
STRAT,  1                %Break only on control characters.
TABLE,  ...
*+10/                    %Make a 8 word large buffer.
NOCHR,  ...
```

ND-100 and ND-500 All users All users

SETBREAKPOINT 47B

Starts the program to be debugged. The program stops at the breakpoint defined by DefineBreakpoint. The debug program then starts.

- Both the program and the debug program must be loaded to the same address space.

See also DefineBreakpoint and GetBreakpointInfo.

PARAMETERS

→ Address of register block containing the register values of the program being debugged.
← Standard error code. See appendix A.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 | All users | All programs

47B SETBREAKPOINT SBRK

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Not available.

MAC

```
        LDT     (REGBL
        MON     47          %Monitor call SetBreakpoint.
        ...
REGBL,  0                   %Register block.
*+10/
```

ND-100 All users All users

UPDAT	SETCLOCK	111B

Gives new values to the computer's clock and calendar. If the computer panel has a clock, it is updated.

- The startup time for RT programs can be set by StartupTime. Such RT programs start according to the new time.
- Illegal time values, eg., 61 minutes, stop the program and output a message.

See also AdjustClock, GetCurrentTime, and GetBasicTime.

PARAMETERS

→ Minutes.
→ Hours.
→ Days.
→ Months.
→ Years.

PASCAL

```
Minute, Hour, Day, Month, Year : INTEGER2;
...
SetClock(Minute, Hour, Day, Month, Year);
```

COBOL

```
01  Minute  COMP.
01  Hour  COMP.
01  Day  COMP.
01  Month  COMP.
01  Year  COMP.
    ...
    MONITOR-CALL "SetClock" USING Minute, Hour, Day, Month, Year.
```

FORTRAN

```
      INTEGER  Minute, Hour, Day, Month, Year
      ...
      Monitor_Call('SetClock', Minute, Hour, Day, Month, Year)
```

ND-100 and ND-500	User RT and user SYSTEM	RT programs

111B	SETCLOCK	UPDAT

Continued from previous page...

PLANC

```
INTEGER : Minute, Hour, Day, Month, Year
...
Monitor_Call('SetClock', Minute, Hour, Day, Month, Year)
```

ASSEMBLY-500

```
Minute : W  BLOCK  1
Hour : W  BLOCK  1
Day : W  BLOCK  1
Month : W  BLOCK  1
Year : W  BLOCK  1
SetClock : EQU 37B9 + 111B
        ...
        CALLG SetClock, 5, Minute, Hour, Day, Month, Year
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     111     %Monitor call SetClock.
        ...
PAR,    MIN             %New minutes of hour.
        HOUR            %New hour of day.
        DAY             %New day of month.
        MONTH           %New month of year.
        YEAR            %New value for year.
        ...
MIN,    ...
HOUR,   ...
DAY,    ...
MONTH,  ...
YEAR,   ...
```

ND-100 and ND-500	User RT and user SYSTEM	RT programs

SETCM — SetCommandBuffer — 12B

Transfers a string to the command buffer. The command buffer contains the last command input from the terminal. You may read the command buffer by reading from logical device number 0. See InByte.

- The command @TERMINAL-STATISTICS lists the command buffer.
- You may apply the SINTRAN III command editing characters to the command buffer when the program has terminated.
- The parameter is fetched through the alternative page table.
- You may use this monitor call to erase sensitive information in the command buffer, eg., password parameters.

See also ExecuteCommand and CallCommand.

PARAMETERS

→ String to be put in the command buffer.

PASCAL

```
Command : PACKED ARRAY [0..31] OF CHAR;
...
SetCommandBuffer(Command);
```

COBOL

```
01  Command   PIC X(32).
    ...
    MONITOR-CALL "SetCommandBuffer" USING Command.
```

FORTRAN

```
      CHARACTER  Command*32
      ...
      Monitor_Call('SetCommandBuffer', Command(1:32))
```

ND-100 and ND-500 | All users | Background programs

12B **SETCOMMANDBUFFER** **SETCB**

Continued from previous page...

PLANC

```
BYTES : Command(0:31)
...
Monitor_Call('SetCommandBuffer', Command)
```

ASSEMBLY-500

```
Command : STRINGDATA  'CLOSE-FILE 102'''
SetCommandBuffer : EQU 37B9 + 12B
        ...
        CALLG SetCommandBuffer, 1, Command
```

MAC

```
        LDA     (CMND    %Address of string with command.
        MON     12       %Monitor call SetCommandBuffer.
        ...
CMND,   'CLOSE-FILE 102' %Transfer 'CLOSE-FILE 102' to the command buffer.
```

ND-100 and ND-500 **All users** **Background programs**

ECHOM SetEcho 3B

When you press a key on the terminal, a character is normally displayed. This is called echo. You modify a terminal's echo with this monitor call.

See also SetBreak.

PARAMETERS

→ The terminal's logical device number. See appendix B. Only needed for RT programs. Background programs ignore this parameter. The user's terminal is assumed.

→ Echo. Less than 0: No echo, eg., password fields.
0: Echo on all characters.
1: Echo on all characters except control characters.
2: Special echo used by MAC.
3-6: System defined echo.
7: User defined echo. See the next parameter.

→ User defined echo table. Ignored if the echo is different from 7. Use 256 bits to represents the ASCII characters. Use 0 for the character that should have echo. For example, if bit number 65 is 0, the character A gives echo. See the ASCII table in appendix F.

PASCAL

```
DeviceNumber, EchoStrategy : INTEGER2;
Table : BITMAP;
...
SetEcho(DeviceNumber, EchoStrategy, Table);
```

COBOL

```
01  DeviceNumber  COMP.
01  EchoStrategy  COMP.
01  Table.
    02  array COMP OCCURS 8 TIMES.
    ...
    MONITOR-CALL "SetEcho" USING DeviceNumber, EchoStrategy, Table.
```

FORTRAN

```
      INTEGER  DeviceNumber, EchoStrategy
      INTEGER  Table(8)
      ...
      Monitor_Call('SetEcho', DeviceNumber, EchoStrategy, Table(1))
```

ND-100 and ND-500 | **All users** | **All programs**

3B	SETECHO	ECHOM

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, EchoStrategy
BYTE ARRAY : Table(0:15)
...
Monitor_Call('SetEcho', DeviceNumber, EchoStrategy, Table(0))
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
EchoStrategy : W  BLOCK  1
Table : H  BLOCK 8
SetEcho : EQU 37B9 + 3B
        ...
        CALLG SetEcho, 3, DeviceNumber, EchoStrategy, Table
```

MAC

```
        LDT     DEVNO    %Logical device number.
        LDA     STRAT    %Echo strategy.
        LDX     (TABLE   %Eight word large bit map.
        MON     3        %Monitor call SetEcho.
        ...
DEVNO,  ...
STRAT,  ...
TABLE,  ...
*+10/                    %Make a buffer of 8 words.
```

ND-100 and ND-500	All users	All users

BUSEL **SETESCAPEHANDLING** 300B

Enables user defined escape handling. When the ESCAPE key is pressed, execution continues at the specified address in your program.

- Disable the user defined escape handling with StopEscapeHandling.
- The normal escape handling is reset when the program aborts.

See also EnableEscape and OffEscLocalFunction.

PARAMETERS

→ Program address where escape handler routine starts.
← Standard error code. See appendix A.

PASCAL

```
EscapeHandler : INTEGER2;
...
SetEscapeHandling(EscapeHandler);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  EscapeHandler  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "SetEscapeHandling" USING EscapeHandler.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  EscapeHandler
      ...
      Monitor_Call('SetEscapeHandling', EscapeHandler)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 | All users | Background programs

300B **SETESCAPEHANDLING** **BUSEL**

Continued from previous page...

PLANC

```
INTEGER : EscapeHandler
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('SetEscapeHandling', EscapeHandler)
```

ASSEMBLY-500

Not available.

MAC

```
        LDA     PROG    %Address of user escape handler routine.
        MON     300     %Monitor call SetEscapeHandling.
        JMP     ERROR   %Error return from monitor call.
        ...             %Normal return.
ERROR,  ...             %Error number in register A.
        ...
PROG,   ...
```

ND-100 **All users** **Background programs**

MSDAE SETESCLOCALCHARS 227B

You can terminate most programs with the ESCAPE key. A LOCAL key has a similar function. It terminates a connection to a remote computer in a network. This monitor call allows you to select other keys for these functions.

- The local function is only used by COSMOS.
- Appendix F shows the ASCII values of the characters.

See also SetEscapeHandling, GetEscLocalChars, OnEscLocalChars, and OffEscLocalChars.

PARAMETERS

→ Logical device number. See appendix B. Only used by RT programs. Your own terminal is always used in background program.
← The local character.
← The escape character.

PASCAL

```
DeviceNo, DisconnectChar, EscapeChar : INTEGER2;
...
SetEscLocalChar(DeviceNo, DisconnectChar, EscapeChar);
```

COBOL

```
01  DeviceNo  COMP.
01  DisconChar  COMP.
01  EscapeChar  COMP.
    ...
    MONITOR-CALL "SetEscLocalChar" USING DeviceNo, DisconChar, EscapeChar.
```

FORTRAN

```
      INTEGER  DeviceNo, DisconChar, EscapeChar
      ...
      Monitor_Call('SetEscLocalChar', DeviceNo, DisconChar, EscapeChar)
```

ND-100 and ND-500 | All users | All programs

227B **SETESCLOCALCHARS** **MSDAR**

Continued from previous page...

PLANC

```
INTEGER : DeviceNo, DisconnectChar, EscapeChar
...
Monitor_Call('SetEscLocalChar', DeviceNo, DisconnectChar, EscapeChar)
```

ASSEMBLY-500

```
DeviceNo : W  BLOCK  1
DisconnectChar : W  BLOCK  1
EscapeChar : W  BLOCK  1
SetEscLocalChar : EQU 37B9 + 227B
        ...
        CALLG SetEscLocalChar, 3, DeviceNo, DisconnectChar, EscapeChar
```

MAC

```
        LDT     DEVNO    %Logical device number.
        LDA     CHAR     %Left byte: disconnect char,
                         % right byte: escape char.
        MON     227      %Monitor call SetEscLocalHandling.
        ...
DEVNO,  ...
CHAR,   ...
```

ND-100 and ND-500 **All users** **All users**

SFACC SETFILEACCESS 237B

Sets the access protection for a file. You should specify the access for yourself, friends, and other users. The default file access for yourself is all kinds of access. Your friends have read access only. Other users have no access.

- You need directory access to a file to change the fila access. User SYSTEM and RT may set the access protection for any files.
- Use the characters R, W, A, C, D, and N to specify the legal file access. R means Read. W means Write. A means Append to the end of a file. C means Common, ie., more than one user may access the file at a time. D means Directory access, ie., the file may be deleted, new versions created, etc. N means No access.
- Use @FILE-STATISTICS to check the file access.

See also CreateFriend, SetObjectEntry, @SET-FILE-ACCESS, and @SET-DEFAULT-FILE-ACCESS.

PARAMETERS

→ File name. It is most efficient to use unabbreviated file names, eg., EXAMPLE:TEXT. The default file type is :SYMB.
→ Public access. Use N or a combination of R, W, A, C, and D, eg., N.
→ Friend access. Use N or a combination of R, W, A, C, and D, eg., RWA.
→ Own access. Use N or a combination of R, W, A, C, and D, eg., RWACD.
← Standard error code. See appendix A.

PASCAL

```
FileName : PACKED ARRAY [0..63] OF CHAR;
PubAccess, FriendAccess, OwnAccess : PACKED ARRAY [0..4] OF CHAR;
...
SetFileAccess(FileName, PubAccess, FriendAccess, OwnAccess);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileName    PIC X(64).
01  PubAcc    PIC X(5).
01  FriendAcc    PIC X(5).
01  OwnAcc    PIC X(5).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "SetFileAccess" USING FileName, PubAcc, FriendAcc, OwnAcc.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      CHARACTER  FileName*64, PubAcc*5, FriendAcc*5, OwnAcc*5
      ...
      Monitor_Call('SetFileAccess', FileName(1:64), PubAcc(1:5),
     C                  FriendAcc(1:5), OwnAcc(1:5))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

237B SETFILEACCESS SPACC

Continued from previous page...

PLANC

```
BYTES : FileName(0:63), PubAccess(0:5), FriendAccess(0:5), OwnAccess(0:5)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('SetFileAccess', FileName, PubAccess, FriendAccess, OwnAccess)
```

ASSEMBLY-500

```
FileName : STRINGDATA  'EXAMPLE:SYMB'''  %Set access of file EXAMPLE:SYMB to
PubAccess : STRINGDATA  'N'''            % no public access
FriendAccess : STRINGDATA  'RA'''        % read and write append for friends
OwnAccess : STRINGDATA  'RWACD'''        % full own access
ErrCode : W  BLOCK  1
SetFileAccess : EQU 37B9 + 237B
        ...
        CALLG SetFileAccess, 4, FileName, PubAccess, FriendAccess, OwnAccess
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (FILE    %Address of string with file name.
        LDT     (PUBAC   %Address of string with public access characters.
        LDA     (FRIAC   %Address of string with own access characters.
        COPY    SA  DD
        LDA     (OWNAC   %Address of string with friend access characters.
        MON     237      %Monitor call SetFileAccess.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILE,   'EXAMPLE:SYMB'   %Change access for EXAMPLE:SYMB.
PUBAC,  'N'              %No public access.
FRIAC,  'RA'             %Read and write append access for friend.
OWNAC,  'RWACD'          %Full own access.
```

ND-100 and ND-500	All users	All users

SBAX SetMaxBytes 73B

Sets the number of bytes in an opened file. The specified number of bytes are stored when the file is closed. The error code 3 is returned if you later try to read beyond this size. Error code 3 means end of file.

- The file must be opened for write.
- This monitor call is only relevant for sequential access.

See also GetBytesInfile and SetStartByte.

PARAMETERS

→ File number. See OpenFile.
→ Maximum file size in bytes.
← Standard error code. See appendix A.

PASCAL

```
FileNumber : INTEGER2;
MaxBytePointer : LONGINT;
...
SetMaxBytes(FileNumber, MaxBytePointer);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNumber  COMP.
01  MaxBytePointer  COMP PIC S9(10).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "SetMaxBytes" USING FileNumber, MaxBytePointer.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNumber
      INTEGER*4  MaxBytePointer
      ...
      Monitor_Call('SetMaxBytes', FileNumber, MaxBytePointer)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

73B **SETMAXBYTES** **SMAX**

Continued from previous page...

PLANC

```
INTEGER : FileNumber
INTEGER4 : MaxBytePointer
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('SetMaxBytes', FileNumber, MaxBytePointer)
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
MaxBytePointer : W  BLOCK  1
ErrCode : W  BLOCK  1
SetMaxBytes : EQU 37B9 + 73B
        ...
        CALLG SetMaxBytes, 2, FileNumber, MaxBytePointer
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     FILNO    %File number returned from earlier open.
        LDD     (POINT   %Maximum byte pointer.
        MON     73       %Monitor call SetMaxBytes.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
POINT,  ...              %A double word.
        ...              %
```

ND-100 and ND-500 **All users** **All users**

5PASBT	SETND500PARAM	436B

Sets information about an ND-500 program. Use GetND500Param to read the 5 parameters when a program is terminated.

- SINTRAN III sets some of the parameter values if you give the command @ENABLE-TERMINATION-HANDLING first.

See also TerminationHandling, GetUserParam, SetUserParam, and GetND500Param.

PARAMETERS

→ The five user parameters as an array. SINTRAN III's termination handling returns the following:

Parameter 1: Bit 24:16 contains the user index. Bit 15:0 contains the directory index.
2: Logical device number of the terminal.
3: Fatal error or the monitor call ErrorMessage returns the error number. If ESCAPE was pressed, -1 is returned.
4: Set by SetND500Param.
5: Set by SetND500Param.

The parameters are returned if the user press the ESCAPE key, if the monitor calls ExitFromProgram or ErrorMessage are executed, or if a fatal error occurs. You can set all parameters if no termination handling is enabled.

PASCAL

```
Buffer : BITMAP;
...
SetND500Param(Buffer);
```

COBOL

```
01  Buffer.
    02  array COMP OCCURS 5 TIMES.
    ...
    MONITOR-CALL "SetND500Param" USING Buffer.
```

FORTRAN

```
      INTEGER  Buffer(5)
      ...
      Monitor_Call('SetND500Param', Buffer(1))
```

ND-500	All users	All programs

436B **SETND500PARAM** SPASET

Continued from previous page...

PLANC

```
INTEGER ARRAY : Buffer(0:4)
...
Monitor_Call('SetND500Param', Buffer(0))
```

ASSEMBLY-500

```
Buffer : W  BLOCK 5
SetND500Param : EQU 37B9 + 436B
        ...
        CALLG SetND500Param, 1, Buffer
```

MAC

This monitor call is not available on the ND-100. See SetUserParam.

ND-500 All users All users

D90OBJ SETOBJECTENTRY 216B

Changes the description of a file. An object entry describes each file. It contains the file name, the access rights, the date last opened for read and write, the size, and more. You may use GetObjectEntry to read an object entry. Change parts of it. Then write it back with SetObjectEntry.

- You specify the directory index, the user index, and the object index.
- There is one object entry for each version of a file.
- Only user SYSTEM can change the object entry of a file without read or write access to the file.
- On the ND-100 you may access files on remote computer systems if the computers are connected through a COSMOS network.

See also GetObjectEntry, GetAllFileIndexes, and @CHANGE-OBJECT-ENTRY.

PARAMETERS

→ The 64 byte object entry. See appendix C.
→ The directory index. See GetAllFileIndexes.
→ The user index. See GetAllFileIndexes.
→ The object index. See GetAllFileIndexes.
→ Remote flag. Use 0 for the local computer and 1 a for remote computer.
→ Remote system identification if remote flag is 1.
← Standard error code. See appendix A.

PASCAL

```
Buff : ARRAY [0..1] OF BITMAP;
DirIndex, UserIndex, ObjectIndex : INTEGER2;
...
SetObjectEntry(Buff, DirIndex, UserIndex, ObjectIndex);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  Buff.
    02  array COMP OCCURS 32 TIMES.
01  DirIndex  COMP.
01  UserIndex  COMP.
01  ObjectIndex  COMP.
01  RemoteFlag  COMP.
01  RemoteSystem  PIC X(64).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "SetObjectEntry" USING Buff, DirIndex, UserIndex, ObjIndex,
                                        RemoteFlag, RemoteSystem.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  Buff(32)
      INTEGER  DirIndex, UserIndex, ObjIndex, RemoteFlag
      CHARACTER RemoteSystem*64
      ...
      Monitor_Call('SetObjectEntry', Buff(1), DirIndex, UserIndex, ObjIndex,
     C                  RemoteFlag, RemoteSystem(1:64))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

216B SETOBJECTENTRY DMOBJ

Continued from previous page...

PLANC

```
BYTE ARRAY : Buff(0:63)
INTEGER : DirIndex, UserIndex, ObjIndex, RemoteFlag
BYTES : RemoteSystem(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('SetObjectEntry', Buff(0), DirIndex, UserIndex, ObjectIndex,&
                RemoteFlag, RemoteSystem)
```

ASSEMBLY-500

```
Buff : H  BLOCK 32
DirIndex : W  BLOCK  1              %SysId used if bit 7 is set to 1.
UserIndex : W  BLOCK  1
ObjIndex : W  BLOCK  1
SysId : STRINGARRAY 20B             %Dummy if not remote system
ErrCode : W  BLOCK  1
SetObjectEntry : EQU 37B9 + 216B
        ...
        CALLG SetObjectEntry, 5, Buff, DirIndex, UserIndex, ObjIndex, SysId
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDA     (REMID   %Remote identification in register D.
        COPY    SA  DD   %Used if bit 15 in register X is set.
        LDA     (BUFF    %Address of buffer containing source object entry.
        LDT     INDEX    %Left byte: Dir. index, right byte: User index.
        LDX     OBJIX    %Object index.
        MON     216      %Monitor call SetObjectEntry.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
BUFF,   0                %
*+40/                    %Make a buffer of 32 words.
INDEX,  ...              %Set bit 15 if remote file.
OBJIX,  ...
REMID,  0                %Remote system identification string.
*+32/                    %Space for the string.
```

ND-100 and ND-500 All users All users

WFLAG SETOUTPUTFLAGS 403B

ND-100 and ND-500 programs may communicate through two 32-bit flag arrays. You can use the flags as you want. SetOutputFlags writes to the output flags. The ND-100 reads these flags with the monitor call ND500Function. See the ND-500 LOADER/MONITOR (ND-60.136).

- You store the last values written to the flags. There is no queue.

See also SetOutputFlags.

PARAMETERS

← Flag values as a 32-bit integer.

PASCAL

```
Value : LONGINT;
...
SetOutputFlags(Value);
```

COBOL

```
01  Value  COMP.
    ...
    MONITOR-CALL "SetOutputFlags" USING Value.
```

FORTRAN

```
      INTEGER  Value
      ...
      Monitor_Call('SetOutputFlags', Value)
```

ND-500 | All users | All programs

403B	SETOUTPUTFLAGS	WFLAG

Continued from previous page...

PLANC

```
INTEGER : Value
...
Monitor_Call('SetOutputFlags', Value)
```

ASSEMBLY-500

```
Value : W  BLOCK  1
SetOutputFlags : EQU 37B9 + 403B
        ...
        CALLG SetOutputFlags, 1, Value
```

MAC

Not available.

ND-500	All users	All users

DOLW **SETOUTREGISTERS** 164B

Sets the registers of a device interface. Mainly for internal use.

See also GetInRegisters.

PARAMETERS

→ Number of registers.
→ Buffer with logical device.
← Data buffer.
← Error indicator.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 and ND-500 User RT and user SYSTEM RT programs

166B SETOUTREGISTERS DOLW

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
NoOfReg : W  BLOCK  1
Buffer : H BLOCK 256
DataBuff : H  BLOCK  256
Buff : H BLOCK 256
RetValue : W  BLOCK  1
SetOutRegisters : EQU  37B9 + 166B
        ...
        CALLG SetOutRegisters, 5, NoOfReg, Buffer, DataBuff, Buff, RetValue
        IF K GO Error
        ...
Error,  ...
```

MAC

```
        MON      166      %Monitor call SetOutRegisters.
```

Trailer

ND-100 and ND-500 User RT and user SYSTEM RT programs

SPEP1 SETPERIPHERALNAME 2348

Defines a peripheral file, eg., a printer. You connect a file name to the logical device number of the peripheral.

- The file name should exist in advance, but with no file type. Otherwise you may include the file name in double quotes ("..."). An empty file type is default.

See also SetTerminalFile and @SET-PERIPHERAL-FILE.

PARAMETERS

→ File name for the peripheral. See appendix G.
→ Logical device number. See appendix B.
← Standard error code. See appendix A.

PASCAL

```
FileName : PACKED ARRAY [0..63] OF CHAR;
DeviceNumber : INTEGER2;
...
SetPeripheralName(FileName, DeviceNumber);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileName   PIC X(64).
01  DeviceNumber  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "SetPeripheralName" USING FileName, DeviceNumber.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      CHARACTER  FileName*64
      INTEGER  DeviceNumber
      ...
      Monitor_Call('SetPeripheralName', FileName(1:64), DeviceNumber)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 User SYSTEM All programs

234B SETPERIPHERALNAME SPEFI

Continued from previous page...

PLANC

```
BYTES : FileName(0:63)
INTEGER : DeviceNumber
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('SetPeripheralName', FileName, DeviceNumber)
```

ASSEMBLY-500

```
FileName : STRINGDATA  'LINE-PRINTER'''   %Use LINE-PRINTER as name of dev.
DeviceNumber : W  DATA  5                 % no. 5 (line printer no. 1).
ErrCode : W  BLOCK  1
SetPeripheralName : EQU 37B9 + 234B
        ...
        CALLG SetPeripheralName, 2, FileName, DeviceNumber
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (FILE    %Address of string with file name.
        LDA     DEVNO    %Logical device number.
        MON     234      %Monitor call SetPeripheralFile.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILE,   'LINE-PRINTER'   %Use LINE-PRINTER as name of device number 5.
DEVNO,  5                %Device number of line printer no. 1.
```

ND-100 and ND-500 User SYSTEM All users

SPEED **SETPERMANENTOPEN** 2362

Sets a file permanently open. The file is not closed by CloseFile with -1 as file number. You have to specify the file number or -2.

- The file must already be open.
- Only mass storage files can be set permanently open.
- The file is not closed when your program terminates.

See also OpenFile, CloseFile, and @SET-PERMANENT-OPEN.

PARAMETERS

→ File number. See OpenFile.
← Standard error code. See appendix A.

PASCAL

```
FileNumber : INTEGER2;
...
SetPermanentOpen(FileNumber);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNumber  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "SetPermanentOpen" USING FileNumber.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNumber
      ...
      Monitor_Call('SetPermanentOpen', FileNumber)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

236B **SETPERMANENTOPEN** **SPECIAL**

Continued from previous page...

PLANC

```
INTEGER : FileNumber
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('SetPermanentOpen', FileNumber)
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
ErrCode : W  BLOCK  1
SetPermanentOpen : EQU 37B9 + 236B
        ...
        CALLG SetPermanentOpen, 1, FileNumber
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     FILNO    %File number returned from earlier open.
        MON     236      %Monitor call SetPermanentOpen.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
```

ND-100 and ND-500 **All users** **All users**

SPRIAM SetProcessName 425B

Defines a new name for your process.

- Process names may be up to 16 characters and contain an additional user name, eg., (P-HANSEN)WP-PROCESS.

See also GetProcessNo and GetOwnProcessInfo.

PARAMETERS

→ Process name.

PASCAL

```
ProcessName : PACKED ARRAY [0..33] OF CHAR;
...
SetProcessName(ProcessName);
```

COBOL

```
01  ProcessName    PIC X(34).
    ...
    MONITOR-CALL "SetProcessName" USING ProcessName.
```

FORTRAN

```
      CHARACTER  ProcessName*34
      ...
      Monitor_Call('SetProcessName', ProcessName(1:34))
```

ND-500 All users All programs

425B **SETPROCESSNAME** SPRNAM

Continued from previous page...

PLANC

```
BYTES : ProcessName(0:33)
...
Monitor_Call('SetProcessName', ProcessName)
```

ASSEMBLY-500

```
ProcessName : STRINGDATA 'WP-PROCESS'''
SetProcessName : EQU 37B9 + 425B
        ...
        CALLG SetProcessName, 1, ProcessName
```

MAC

Not available.

ND-500 All users All users

SPRIO SetProcessPriority 507B

Sets the priority for a process in the ND-500. The priorities vary from 0 to 255. The process with the highest priority is executed first.

- The priorities of background programs normally vary between 20 and 64. SINTRAN III modifies the priorities all the time. This is done to allow several jobs to share the CPU. Specify 0 to execute a process in the same way.
- With SetProcessPriority, you may fix the priority.

See also SetRTPriority.

PARAMETERS

→ Priority.

PASCAL

```
NewPriority : LONGINT;
...
SetProcessPriority(NewPriority);
```

COBOL

```
01  NewPriority  COMP.
    ...
    MONITOR-CALL "SetProcessPriority" USING NewPriority.
```

FORTRAN

```
      INTEGER  NewPriority
      ...
      Monitor_Call('SetProcessPriority', NewPriority)
```

ND-500 | User RT and user SYSTEM | All programs

507B	SETPROCESSPRIORITY	SPRIO

Continued from previous page...

PLANC

```
INTEGER : NewPriority
...
Monitor_Call('SetProcessPriority', NewPriority)
```

ASSEMBLY-500

```
NewPriority : W  BLOCK  1
SetProcessPriority : EQU 37B9 + 507B
        ...
        CALLG SetProcessPriority, 1, NewPriority
```

MAC

Not available.

ND-500	User RT and user SYSTEM	All users

SETLMD — SETREMOTEACCESS — 316B

Switches remote file access on and off. The COSMOS network allows you to access files in remote computers directly. Use DefaultRemoteSystem or @SET-DEFAULT-REMOTE-SYSTEM to specify a default remote system. If a file does not exist in the local system, the default remote system is searched. SetRemoteAccess switches this function on and off.

- You may include a remote system identification in the file name. Only the specified system is searched.
- This monitor call is only available with COSMOS.

See also @SET-LOCAL-MODE or @SET-REMOTE-MODE.

PARAMETERS

→ Mode. Use 0 to switch remote mode off. Switch it on with 1.
← Standard error code. See appendix A.

PASCAL

```
Mode : INTEGER2;
...
SetRemoteAccess(Mode);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  Mode  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "SetRemoteAccess" USING Mode.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  Mode
      ...
      Monitor_Call('SetRemoteAccess', Mode)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

316B **SetRemoteAccess** SELND

Continued from previous page...

PLANC

```
INTEGER : Mode
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('SetRemoteAccess', Mode)
```

ASSEMBLY-500

```
Mode : W  BLOCK  1
SetRemoteAccess : EQU 37B9 + 316B
        ...
        CALLG SetRemoteAccess, 1, Mode
```

MAC

```
        LDA     MODE     %Local / Remote Flag.
        MON     316      %Monitor call SetRemoteAccess.
        ...
MODE,   ...
```

ND-100 and ND-500 | All users | All users

PRIOR — SETRTPRIORITY — 110B

Sets the priority of an RT program. RT programs may be given priorities from 0 to 255. SINTRAN III executes the RT program with the highest priority.

- The priority of background programs vary between 20 and 60.
- Programs with priority 0 will never start. You may use this to suspend programs.

See also SetND500Priority and @PRIOR.

PARAMETERS

→ Address of the RT description. You may use 0 for your own program.
→ Priority.
← The previous priority.

PASCAL

```
RTProgram, PriorityLevel, OldPriority : INTEGER2;
...
SetRTPriority(RTProgram, PriorityLevel, OldPriority);
```

COBOL

```
01  RTProgram  COMP.
01  PriorLevel  COMP.
01  OldPriority  COMP.
    ...
    MONITOR-CALL "SetRTPriority" USING RTProgram, PriorLevel, OldPriority.
```

FORTRAN

```
      INTEGER  RTProgram, PriorityLevel, OldPriority
      ...
      Monitor_Call('SetRTPriority', RTProgram, PriorityLevel, OldPriority)
```

ND-100 | User RT and user SYSTEM | RT programs

110B SETRTPRIORITY PRIOR

Continued from previous page...

PLANC

```
INTEGER : RTProgram, PriorityLevel, OldPriority
...
Monitor_Call('SetRTPriority', RTProgram, PriorityLevel, OldPriority)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
PriorityLevel : W  BLOCK  1
OldPriority : W  BLOCK  1
SetRTPriority : EQU 37B9 + 110B
        ...
        CALLG SetRTPriority, 2, RTProgram, PriorityLevel
        W1 =: OldPriority                %Result is returned in W1 register.
```

MAC

```
        LDA     (PAR     %Load register A with address of parameter list.
        MON     110      %Monitor call SetRTPriority.
        STA     OLDP     %Store old priority returned.
        ...
OLDP,   0
PAR,    RTPRO            %Address of RT description.
        PRIOR            %New priority.
        ...
RTPRO,  ...
PRIOR,  ...
```

ND-100 User RT and user SYSTEM RT programs

SETBL SetStartBlock 77B

Sets the next block to be read or written in an opened file. You may access the first bytes in the block with the monitor calls to read and write bytes.

- The standard block size is 512 bytes. This block size is set when the file is opened. You can change this with SetBlockSize.
- The first block of a file is number 0.

See also SetStartByte and @SET-BLOCK-POINTER. The monitor call is identical to SetStartByte with the block number multiplied by the block size as parameter.

PARAMETERS

→ File number. See OpenFile.
→ Block number.
← Standard error code. See appendix A.

PASCAL

```
FileNumber, BlockNumber : INTEGER2;
...
SetStartBlock(FileNumber, BlockNumber);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNumber  COMP.
01  BlockNumber  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "SetStartBlock" USING FileNumber, BlockNumber.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNumber, BlockNumber
      ...
      Monitor_Call('SetStartBlock', FileNumber, BlockNumber)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

77B **SETSTARTBLOCK** SETSBL

Continued from previous page...

PLANC

```
INTEGER : FileNumber, BlockNumber
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('SetStartBlock', FileNumber, BlockNumber)
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
BlockNumber : W  BLOCK  1
ErrCode : W  BLOCK  1
SetStartBlock : EQU 37B9 + 77B
        ...
        CALLG SetStartBlock, 2, FileNumber, BlockNumber
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     FILNO    %File number returned from earlier open.
        LDA     BLKNO    %Block number.
        MON     77       %Monitor call SetStartBlock.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
BLKNO,  ...
```

ND-100 and ND-500 All users All users

SETBT SetStartByte 74B

Sets the next byte to be read or written in an opened mass storage file.

- The bytes in a file are numbered upwards from 0.

See also GetStartByte, SetStartBlock, SetMaxBytes, and @SET-BYTE-POINTER.

PARAMETERS

→ File number. See OpenFile.
→ Start byte in the file.
← Standard error code. See appendix A.

PASCAL

```
FileNumber : INTEGER2;
BytePointer : LONGINT;
...
SetBytePointer(FileNumber, BytePointer);      [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNumber  COMP.
01  BytePointer  COMP PIC S9(10).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "SetStartByte" USING FileNumber, BytePointer.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNumber
      INTEGER*4  BytePointer
      ...
      Monitor_Call('SetStartByte', FileNumber, BytePointer)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

74B SETSTARTBYTE SETBT

Continued from previous page...

PLANC

```
INTEGER : FileNumber
INTEGER4 : BytePointer
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('SetStartByte', FileNumber, BytePointer)
```

ASSEMBLY-500

```
FileNumber : W  BLOCK  1
BytePointer : W  BLOCK  1
ErrCode : W  BLOCK  1
SetStartByte : EQU 37B9 + 74B
        ...
        CALLG SetStartByte, 2, FileNumber, BytePointer
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     FILNO    %File number returned from earlier open.
        LDD     POINT    %Byte pointer.
        MON     74       %Monitor call SetStartByte.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
POINT,  ...              %A double word.
        ...              %
```

ND-100 and ND-500 All users All users

SYSCP1 SETTEMPORARYFILE 233B

Defines> a file to store information temporarily. The file can be read once. When it is closed, its contents are deleted. The empty file exists.

- GetObjectEntry and @FILE-STATISTICS shows whether a file is temporary or not.
- Files to be printed are commonly defined as temporary. Their contents exist until they have been printed.

See also CreateFile and @SET-TEMPORARY-FILE.

PARAMETERS

→ File name.
← Standard error code. See appendix A.

PASCAL

```
FileName : PACKED ARRAY [0..63] OF CHAR;
...
SetTemporaryFile(FileName);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileName   PIC X(64).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "SetTemporaryFile" USING FileName.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      CHARACTER  FileName*64
      ...
      Monitor_Call('SetTemporaryFile', FileName(1:64))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 All users All programs

233B **SETTEMPORARYFILE** STEF1

Continued from previous page...

PLANC

```
BYTES : FileName(0:63)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('SetTemporaryFile', FileName)
```

ASSEMBLY-500

```
FileName : STRINGDATA  'TEMP-FILE:SYMB'''
ErrCode : W  BLOCK  1
SetTemporaryFile : EQU 37B9 + 233B
        ...
        CALLG SetTemporaryFile, 1, FileName
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (FILE    %Address of string with file name.
        MON     233      %Monitor call SetTemporaryFile.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILE,   'TEMP-FILE:SYMB' %Treat TEMP-FILE:SYMB as a temporary file.
```

ND-100 and ND-500 All users All users

STRFI SETTERMINALNAME 27SB

Defines the file name to be used for terminals. This is normally TERMINAL:. Background users identify their own terminal with this file name.

- You may use this monitor call more than once. Each file name will identify the terminals.

See also SetPeripheralName and @SET-TERMINAL-FILE.

PARAMETERS

→ File name.
← Standard error code. See appendix A.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 and ND-500 | User SYSTEM | All programs

275B **SETTERMINALNAME** SYSPT

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
FileName : STRINGDATA  'TERMINAL'''
ErrCode : W  BLOCK  1
        ...
SetTerminalName : EQU  37B9 + 275B
        CALLG SetTerminalName, 1, FileName
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDX     (NAME   %Address of string containing terminal name.
        MON     275     %Monitor call SetTerminalName.
        ...
NAME,   'TERMINAL'      %Set terminal name to TERMINAL.
```

ND-100 and ND-500 User SYSTEM All users

MSTTY SETTERMINALTYPE 178

Sets the type of a terminal. The terminal type tells SINTRAN III how to handle a particular terminal. A wrong terminal type normally distorts the screen. The function keys cannot be used.

- Appendix H lists the terminal types.

See also GetTerminalType, and @SET-TERMINAL-TYPE.

PARAMETERS

→ The logical device number of the terminal. You may use 1 for your own terminal in background programs.
→ The terminal type.
← Standard error code. See appendix A.

PASCAL

```
DeviceNumber, TerminalType : INTEGER2;
...
SetTerminalType(DeviceNumber, TerminalType);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  TerminalType  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "SetTerminalType" USING DeviceNumber, TerminalType.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, TerminalType
      ...
      Monitor_Call('SetTerminalType', DeviceNumber, TerminalType)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

17B SETTERMINALTYPE MSTTY

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, TerminalType
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('SetTerminalType', DeviceNumber, TerminalType)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
TerminalType : W  BLOCK  1
ErrCode : W  BLOCK  1
SetTerminalType : EQU 37B9 + 17B
        ...
        CALLG SetTerminalType, 2, DeviceNumber, TerminalType
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DEVNO    %Logical device number.
        LDA     TYPE     %Terminal type number.
        MON     17       %Monitor call SetTerminalType.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
DEVNO,  ...
TYPE,   ...
```

ND-100 and ND-500 | All users | All users

PASET **SETUSERPARAM** **SGB**

Sets information about a background program. Use GetUserParam to read the 5 parameters when a program is terminated.

- SINTRAN III sets some of the parameter values if you give the command @ENABLE-TERMINATION-HANDLING first.

See also TerminationHandling, GetUserParam, @DEFINE-TERMINATION-HANDLING, @DISABLE-TERMINATION-HANDLING, and @SET-USER-PARAMETERS.

PARAMETERS

→ The five user parameters. This is an array of 16-bit integers on the ND-100. ND-500 uses an array of 32-bit integers. SINTRAN III's termination handling returns the following:

Parameter 1: The last byte contains the user index. The byte in front of it contains the directory index.
2: Logical device number of the terminal.
3: Fatal error or the monitor call ErrorMessage returns the error number. If ESCAPE was pressed, -1 is returned.
4: Set by SetUserParam.
5: Set by SetUserParam.

The parameters are returned if the user press the ESCAPE key, if the monitor calls ExitFromProgram or ErrorMessage are executed, or if a fatal error occurs. You can set all parameters if no termination handling is enabled.

PASCAL

```
Buff : BITMAP;
...
SetUserParam(Buff);
```

COBOL

```
01  Buff.
    02  array COMP OCCURS 5 TIMES.
    ...
    MONITOR-CALL "SetUserParam" USING Buff.
```

FORTRAN

```
      INTEGER  Buff(5)
      ...
      Monitor_Call('SetUserParam', Buff(1))
```

ND-100 and ND-500 **All users** **All programs**

56B **SETUSERPARAM** **PASET**

Continued from previous page...

PLANC

```
INTEGER ARRAY : Buff(0:4)
...
Monitor_Call('SetUserParam', Buff(0))
```

ASSEMBLY-500

```
Buff : H  BLOCK 5
SetUserParam : EQU 37B9 + 56B
        ...
        CALLG SetUserParam, 1, Buff
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     56      %Monitor call SetUserParam.
        ...
PAR,    BUFF            %Buffer of 5 words for setting user parameters.
        ...
BUFF,   ...             %Left byte: directory index, Right byte: user
index.
        ...             %Logical device number, terminal number.
        ...             % -1 if escape, otherwise error number.
        ...             %User defined.
        ...             %User defined.
```

ND-100 and ND-500 **All users** **All users**

SIBFU SIBASFUNCTION 432B

Reserves and releases communication facilities for the SIBAS database system.

- This monitor call is intended for internal use only.

See also AnswerSIBAS, GetSIBASMessage, and SendSIBASMessage.

PARAMETERS

→ Function. Use 0 to reserve. Other values releases the communication facility.
→ SIBAS number. See the SIBAS documentation.

PASCAL

```
Func, SIBASNumber : INTEGER2;
...
SIBASFunction(Func, SIBASNumber);
```

COBOL

Not available.

FORTRAN

Not available.

ND-500 User RT and user SYSTEM RT programs

432B SIBASFUNCTION SIBPU

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
Func : W  BLOCK  1
SIBASNumber : W  BLOCK  1
SIBASFunction : EQU 37B9 + 432B
        ...
        CALLG SIBASFunction, 2, Func, SIBASNumber
```

MAC

Not available.

ND-500 User RT and user SYSTEM RT programs

CONCT — STARTONINTERRUPT — 106B

StartOnInterrupt connects an RT program to interrupts from a device. The RT program starts when an interrupt occurs. Use either WaitForRestart or SuspendProgram to make the program wait.

- You remove this connection with NoInterruptStart.
- Several devices may be connected to one program.
- It is impossible to connect to some devices. Connection can be made if SINTRAN III have been generated with a connect driver routine for the device.

See also NoInterruptStart and @CONCT.

PARAMETERS

→ Address of RT description. Use 0 for your own RT description address.
→ Logical device number. See appendix B.

PASCAL

```
RTProgram, DeviceNumber : INTEGER2;
...
StartOnInterrupt(RTProgram, DeviceNumber);
```

COBOL

```
01  RTProgram  COMP.
01  DeviceNumber  COMP.
    ...
    MONITOR-CALL "StartOnInterrupt" USING RTProgram, DeviceNumber.
```

FORTRAN

```
      INTEGER  RTProgram, DeviceNumber
      ...
      Monitor_Call('StartOnInterrupt', RTProgram, DeviceNumber)
```

ND-100 and ND-500 | **User RT and user SYSTEM** | **RT programs**

106B **StartOnInterrupt** **CONCT**

Continued from previous page...

PLANC

```
INTEGER : RTProgram, DeviceNumber
...
Monitor_Call('StartOnInterrupt', RTProgram, DeviceNumber)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
DeviceNumber : W  BLOCK  1
StartOnInterrupt : EQU 37B9 + 106B
        ...
        CALLG StartOnInterrupt, 2, RTProgram, DeviceNumber
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     106     %Monitor call StartOnInterrupt.
        ...
PAR,    RTPRO           %Address of RT description.
        DEVNO           %Logical device number.
        ...
RTPRO,  ...
DEVNO,  ...
```

ND-100 and ND-500 **User RT and user SYSTEM** **RT programs**

STARTP **StartProcess** **500B**

Starts a process in the ND-500. You identify the process with the process number.

- The process may be active. The process then restarts as soon as it terminates.

See also StopProcess and SwitchProcess.

PARAMETERS

→ Process number.
← Standard error code. See appendix A.

PASCAL

```
ProcessNumber : LONGINT;
...
StartProcess(ProcessNumber);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  ProcessNumber  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "StartProcess" USING ProcessNumber.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  ProcessNumber
      ...
      Monitor_Call('StartProcess', ProcessNumber)
      IF (ErrCode .NE. 0) THEN ...
```

ND-500 **All users** **All programs**

500B **STARTPROCESS** **STARTP**

Continued from previous page...

PLANC

```
INTEGER : ProcessNumber
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('StartProcess', ProcessNumber)
```

ASSEMBLY-500

```
ProcessNumber : W  BLOCK  1
ErrCode : W  BLOCK  1
StartProcess : EQU 37B9 + 500B
        ...
        CALLG StartProcess, 1, ProcessNumber
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

Not available.

ND-500 **All users** **All users**

RT StartRTProgram 100B

Starts an RT program. The program is moved to the execution queue. It is executed according to its priority.

- The program may be in the execution queue. Then it restarts as soon as it terminates.
- You can terminate RT programs with StopRTProgram.

See also StartupTime, StartupInterval, DelayStart, and @RT.

PARAMETERS

→ Address of RT description. Use 0 for your own RT description address.

PASCAL

```
RTProgram : INTEGER2;
...
StartRTProgram(RTProgram);
```

COBOL

```
01  RTProgram  COMP.
    ...
    MONITOR-CALL "StartRTProgram" USING RTProgram.
```

FORTRAN

```
      INTEGER  RTProgram
      ...
      Monitor_Call('StartRTProgram', RTProgram)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

100B STARTRTPROGRAM RT

Continued from previous page...

PLANC

```
INTEGER : RTProgram
...
Monitor_Call('StartRTProgram', RTProgram)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
StartRTProgram : EQU 37B9 + 100B
        ...
        CALLG StartRTProgram, 1, RTProgram
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     100     %Monitor call StartRTProgram.
        ...
PAR,    RTPRO           %Address of RT description.
        ...
RTPRO,  ...
```

ND-100 and ND-500 User RT and user SYSTEM RT programs

INTV STARTUPINTERVAL 103B

Prepares an RT program for periodic execution. The interval between the executions can be specified in hours, minutes, seconds, and basic time units. A basic time unit is 1/50th of a second.

- The RT program is not started. Use StartRTProgram, @RT, or similar to start it.
- StopRTProgram, Disconnect, @ABORT or @DSCNT cancel this monitor call.
- One execution may be unfinished when it is time for the next execution. In that case the program's restart flag is set. If the delay becomes as long as two intervals, one execution is lost.
- The interval replaces any earlier specified intervals.
- AdjustClock and @CLADJ do not affect the interval.

See also ExactInterval and @INTV. ExactInterval allows you to specify intervals between 0 and 4294967647 basic time units.

PARAMETERS

→ Address of an RT description. 0 means calling program. GetRtAddress gives RT description addresses.
→ Number of time units between executions of the program.
→ The type of time units. 1 = basic time units, ie., 1/50th of a second, 2 = seconds, 3 = minutes, 4 = hours.

PASCAL

```
RTProgram, Time,Units : INTEGER2;
...
StartInterval(RTProgram, Time, Units);     [Note routine name.]
```

COBOL

```
01  RTProgram  COMP.
01  Time  COMP.
01  Units  COMP.
    ...
    MONITOR-CALL "StartupInterval" USING RTProgram, Time, Units.
```

FORTRAN

```
      INTEGER  RTProgram, Time, Units
      ...
      Monitor_Call('StartupInterval', RTProgram, Time, Units)
```

ND-100 and ND-500 User RT and user SYSTEM RT programs

103B STARTUPINTERVAL INTV

Continued from previous page...

PLANC

```
INTEGER : RTProgram, Time, Units
...
Monitor_Call('StartupInterval', RTProgram, Time, Units)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
Time : W  BLOCK  1
Units : W  BLOCK  1
StartupInterval : EQU 37B9 + 103B
        ...
        CALLG StartupInterval, 3, RTProgram, Time, Units
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     103     %Monitor call StartupInterval.
        ...
PAR,    RTPRO           %Address of RT description.
        TIME            %Interval between each execution of program.
        UNIT            %Time units.
        ...
RTPRO,  ...
UNIT,   ...
BASE,   ...
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

ABSET — StartupTime — 102B

Starts an RT program at a specified time of the day. The RT program is then put in the time queue. It is moved to the execution queue at the specified time.

- The time of the day may have passed. In that case, the program starts the next day.
- RT programs already in the time queue are reinserted according to the new specifications.
- AdjustClock, and @CLADJ affect the system's clock. The RT program starts according to the new time.

See also ExactStartup, DelayStartup, and @ABSET.

PARAMETERS

→ Address of the RT description. Use 0 for the calling program.
→ Seconds.
→ Minutes.
→ Hours.

PASCAL

```
RTProgram, Seconds, Minutes, Hours : INTEGER2;
...
StartTime(RTProgram, Seconds, Minutes, Hours);     [Note routine name.]
```

COBOL

```
01  RTProgram  COMP.
01  Seconds  COMP.
01  Minutes  COMP.
01  Hours  COMP.
    ...
    MONITOR-CALL "StartupTime" USING RTProgram, Seconds, Minutes, Hours.
```

FORTRAN

```
      INTEGER  RTProgram, Seconds, Minutes, Hours
      ...
      Monitor_Call('StartupTime', RTProgram, Seconds, Minutes, Hours)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

102B	STARTUPTIME	ABSET

Continued from previous page...

PLANC

```
INTEGER : RTProgram, Seconds, Minutes, Hours
...
Monitor_Call('StartupTime', RTProgram, Seconds, Minutes, Hours)
```

ASSEMBLY-500

```
RTProgram : W  BLOCK  1
Seconds : W  BLOCK  1
Minutes : W  BLOCK  1
Hours : W  BLOCK  1
StartupTime : EQU 37B9 + 102B
        ...
        CALLG StartupTime, 4, RTProgram, Seconds, Minutes, Hours
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     102     %Monitor call StartupTime.
        ...
PAR,    RTPRO           %Address of RT description.
        SEC             %Seconds.
        MIN             %Minutes.
        HOUR            %Hour of day.
        ...
RTPRO,  ...
SEC,    ...
MIN,    ...
HOUR,   ...
```

ND-100 and ND-500	User RT and user SYSTEM	RT programs

DUSEL **StopEscapeHandling** 301B

Disables user defined escape handling. The ESCAPE key terminates the program as normal. StartEscapeHandling starts user defined escape handling.

- User defined escape handling stops when a program terminates.

See also StartEscapeHandling and DisableEscape.

PARAMETERS

← Standard error code. See appendix A.

PASCAL

Not available.

COBOL

```
MONITOR-CALL "StopEscapeHandling".
CALL "CbError" USING ErrCode.
IF ErrCode NOT 0 GO ....
```

FORTRAN

```
Monitor_Call ('StopEscapeHandling')
IF (ErrCode .NE. 0) THEN...
```

ND-100 | All users | Background programs

301B **StopEscapeHandling** **DUSEL**

Continued from previous page...

PLANC

```
ON ROUTINEERROR DO
  IF ErrCode >< 0 THEN...
ENDON
...
Monitor_Call ('StopEscapeHandling')
```

ASSEMBLY-500

Not available.

MAC

```
        MON     301     %Monitor call StopEscapeHandling.
        JMP     ERROR   %Error return from monitor call.
        ...             %Normal return.
ERROR,  ...             %Error number in register A.
        ...
```

ND-100 **All users** **Background programs**

STOPPR	StopProcess	501B

Sets the current process in a wait state. StartProcess restarts the process. Execution continues after the monitor call. The ESCAPE key terminates the waiting program.

- The process restarts immediately if it is scheduled for repeated execution.
- Use ExitFromProgram to terminate the execution.

See also StartProcess and SwitchProcess.

PARAMETERS

This monitor call has no parameters.

PASCAL

Not available.

COBOL

Not available.

FORTRAM

Not available.

ND-500	All users	All programs

501B	StopProcess	STOPPR

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
StopProcess :  EQU  37B9 + 501B
        CALLG StopProcess
```

MAC

Not available.

ND-500	All users	All users

ABORT — StopRTProgram — 105B

Stops an RT program. It is removed from the time or execution queue. All reserved devices and files are released. Periodic execution stops.

- Nothing happens if the RT program is already stopped.
- The RT program restarts immediately if its restart flag is set.

See also ExitRTProgram and @ABORT.

PARAMETERS

→ Address of the RT description. Use 0 for the calling program.

PASCAL

Not available.

COBOL

```
INTEGER RTProgram
Monitor_Call('StopRTProgram',RTProgram)
```

FORTRAN

```
INTEGER : RTProgram
Monitor_Call('StopRTProgram',RTProgram)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

105B StopRTProgram ABORT

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
RTDescr : W  BLOCK  1
        ...
StopRTProgram :  EQU  37B9 + 105B
        CALLG StopRTProgram, 1, RTDescr
```

MAC

```
        LDA     (PAR      %Load register A with address of parameter list.
        MON     105       %Monitor call StopRTProgram
        ...
PAR,    RTPRO             %Address of RT description.
        ...
RTPRO,  ...
```

ND-100 and ND-500 User RT and user SYSTEM RT programs

HOLD — SUSPENDPROGRAM — 104B

Suspends the execution of your program for a given time. The execution then continues after the time specified by the monitor call.

- No reserved files or devices are released.
- The execution continues immediately if the program has the restart flag set.
- You may use NoWaitSwitch. Then the program restarts when a break occurs.

See also TimeOut, WaitForRestart and @HOLD.

PARAMETERS

→ Number of time units to suspend the program.
→ The type of time units. 1 = basic time units, ie., 1/50th of a second, 2 = seconds, 3 = minutes, 4 = hours.

PASCAL

```
TimeUnits, UnitType : INTEGER2;
...
SuspendProgram(TimeUnits, UnitType);
```

COBOL

```
01  TimeUnits  COMP.
01  UnitType  COMP.
    ...
    MONITOR-CALL "SuspendProgram" USING TimeUnits, UnitType.
```

FORTRAN

```
      INTEGER  TimeUnits, UnitType
      ...
      Monitor_Call('SuspendProgram', TimeUnits, UnitType)
```

ND-100 and ND-500 | All users | All programs

104B **SUSPENDPROGRAM** **HOLD**

Continued from previous page...

PLANC

```
INTEGER : TimeUnits, UnitType
...
Monitor_Call('SuspendProgram', TimeUnits, UnitType)
```

ASSEMBLY-500

```
TimeUnits : W  BLOCK  1
UnitType : W  BLOCK  1
SuspendProgram : EQU 37B9 + 104B
        ...
        CALLG SuspendProgram, 2, TimeUnits, UnitType
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     104     %Monitor call SuspendProgram.
        ...
PAR,    TIME            %Number of time units the program has to wait.
        BASE            %Base time units.
        ...
TIME,   ...
BASE,   ...
```

ND-100 and ND-500 **All users** **All users**

SWITCHP **SwitchProcess** **502B**

Sets the current process in a wait state. Restarts another process. This is similar to executing a StartProcess followed by a StopProcess.

- The stopped process restarts immediately if it is scheduled for repeated execution.

See also StartProcess and StopProcess.

PARAMETERS

→ Process number to start.

PASCAL

```
ProcessNumber : LONGINT;
...
SwitchProcess(ProcessNumber);
```

COBOL

```
01  ProcessNumber  COMP.
    ...
    MONITOR-CALL "SwitchProcess" USING ProcessNumber.
```

FORTRAN

```
      INTEGER  ProcessNumber
      ...
      Monitor_Call('SwitchProcess', ProcessNumber)
```

ND-500 **All users** **All programs**

502B	SwitchProcess	SWITCHP

Continued from previous page...

PLANC

```
INTEGER : ProcessNumber
...
Monitor_Call('SwitchProcess', ProcessNumber)
```

ASSEMBLY-500

```
ProcessNumber : W  BLOCK  1
SwitchProcess : EQU 37B9 + 502B
        ...
        CALLG SwitchProcess, 1, ProcessNumber
```

MAC

Not available.

ND-500	All users	All users

USTBRK — SwitchUserBreak — 405B

Switches user defined escape handling on and off. The user defined escape handling transfers control to a routine when you press the ESCAPE key.

See also GetUserRegisters.

PARAMETERS

→ On off flag. Use 1 for on and 0 for off.
→ Program address to start at when you press the ESCAPE key.

PASCAL

```
Func, Address : LONGINT;
...
SwitchUserBreak(Func, Address);
```

COBOL

```
01  Func  COMP.
01  Address  COMP.
    ...
    MONITOR-CALL "SwitchUserBreak" USING Func, Address.
```

FORTRAN

```
      INTEGER  Func, Address
      ...
      Monitor_Call('SwitchUserBreak', Func, Address)
```

ND-500 | All users | All programs

405B **SwitchUserBreak** USTBRK

Continued from previous page...

PLANC

```
INTEGER : Func, Address
...
Monitor_Call('SwitchUserBreak', Func, Address)
```

ASSEMBLY-500

```
Func : W  BLOCK  1
Address : W  BLOCK  1
SwitchUserBreak : EQU 37B9 + 405B
        ...
        CALLG SwitchUserBreak, 2, Func, Address
```

MAC

Not available.

ND-500 All users All users

SYSCT **SYSTEMCONTROL** 261B

System control of device.

See also UserControl.

PARAMETERS

→ Function.
→ Device number.
←
← Standard error code. See appendix A.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 All users All programs

261B **SYSTEMCONTROL** SYSCT

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Not available.

MAC

```
MON     261     %Monitor call SystemControl.
```

ND-100 All users All users

TRMPP TERMINALLINEINFO 3328

Gets information about a terminal line. You may also enable programs to continue in spite of errors with the terminal line.

See also TerminalStatus.

PARAMETERS

→ Function code. Use 1 to enable program to continue after line errors. 0 disables this function. Use 2 to get terminal line information.
→ Logical device number of a terminal. Use 1 for your own terminal.
← Terminal line information. Dummy for function code 0 and 1. The following bits are used:

Bit 0: Set if the terminal line does not function.
1: Set if logout waits for ExitFromProgram.
2: Overflow in input buffer. Some characters are lost.
3: Parity error on input.
4: Framing error on input.

← Standard error code. See appendix A.

PASCAL

```
FuncCode, DeviceNo, ReturnInfo : INTEGER2;
...
TermLineInfo(FuncCode, DeviceNo, ReturnInfo);      [note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FuncCode  COMP.
01  DeviceNo  COMP.
01  ReturnInfo  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "TerminalLineInfo" USING FuncCode, DeviceNo, ReturnInfo.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FuncCode, DeviceNo, ReturnInfo
      ...
      Monitor_Call('TerminalLineInfo', FuncCode, DeviceNo, ReturnInfo)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | Background programs

332B TERMINALLINEINFO

Continued from previous page...

PLANC

```
INTEGER : FuncCode, DeviceNo, ReturnInfo
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('TerminalLineInfo', FuncCode, DeviceNo, ReturnInfo)
```

ASSEMBLY-500

```
FuncCode : W  BLOCK  1
DeviceNo : W  BLOCK  1
ReturnInfo : W  BLOCK  1
ErrCode : W  BLOCK  1
TerminalLineInfo : EQU 37B9 + 332B
        ...
        CALLG TerminalLineInfo, 2, FuncCode, DeviceNo
        IF K GO ERROR
        W1 =: ReturnInfo
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     LDN     %Load register T with the logical device number.
        LDA     FUNC    %Load register A with function code.
        MON 332         %Monitor call TerminalStatus.
        STA     STAT    %Returned status. Standard error code.
        STA     LINFO   %Returns here if function code = 2 and no
        ...             %errors have occurred.
LDN,    1
FUNC,   0
LINFO,  0
STAT,   0
```

ND-100 and ND-500 | All users | Background programs

TERMINALMODE

Selects various terminal functions. You may stop output on full page. Input may be converted to uppercase letters. A delay after carriage return can be set. You may also set automatic logout if the line between the terminal and the computer is broken.

- Terminal mode can also be set on TADs.

See also GetTerminalMode and @TERMINAL-MODE.

PARAMETERS

→ The logical device number of the terminal. See appendix B. Use 1 for your own terminal.
← The terminal mode. The numbers below are used.

Terminal mode	Capital letters?	Delay after return?	Stop on full page?	Logout on missing carrier?
0	No	No	No	No
1	Yes	No	No	No
2	No	Yes	No	No
3	Yes	Yes	No	No
4	No	No	Yes	No
5	Yes	No	Yes	No
6	No	Yes	Yes	No
7	Yes	Yes	Yes	No
8	No	No	No	Yes
9	Yes	No	No	Yes
10	No	Yes	No	Yes
11	Yes	Yes	No	Yes
12	No	No	Yes	Yes
13	Yes	No	Yes	Yes
14	No	Yes	Yes	Yes
15	Yes	Yes	Yes	Yes

← Standard error code. See appendix A. If you try to use the monitor call through ND-NET, -1 is retuned.

PASCAL

```
DeviceNumber, Mode : INTEGER2;
...
TermMode(DeviceNumber, Mode);      [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP. 01  Mode  COMP.
01  ErrCode  COMP.
    MONITOR-CALL "TerminalMode" USING DeviceNumber, Mode.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber, Mode
      ...
      Monitor_Call('TerminalMode', DeviceNumber, Mode)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | Background programs

52B **TERMINALMODE** TRMMD

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, Mode
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('TerminalMode', DeviceNumber, Mode)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
Mode : W  BLOCK  1
ErrCode : W  BLOCK  1
TerminalMode : EQU 37B9 + 52B
        ...
        CALLG TerminalMode, 2, DeviceNumber, Mode
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

PMAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     52      %Monitor call TerminalMode.
        JAF     ERROR   %Handle error if register A is non-zero.
        ...
ERROR,  ...             %Error number in register A.
        ...
PAR,    DEVNO           %Logical device number.
        MODE            %Communication mode.
        ...
DEVNO,  ...
MODE,   ...
```

ND-100 and ND-500 | All users | Background programs

TERMINALNOWAIT

Switches No Wait on and off. No Wait is useful for input from and output to character devices, eg., terminals. In No Wait, the program does not wait for input or output. Monitor calls like InByte returns the error code 3 instead.

- SuspendProgram or WaitForRestart may passivate the program afterwards. The program then restarts when the device detects a break.
- The input buffer must be emtied before passivating the program.

See also InByte, OutByte, and NoWaitSwitch.

PARAMETERS

→ Logical device number of a character device. See appendix B.
→ Input or output flag. Use 0 for input and 1 for output.
→ No Wait flag. Use 0 to switch No Wait on, and 1 to switch it off.
← Status.

PASCAL

```
DevNo, IOFlag, NoWaitFlag, RetStatus : INTEGER2;
...
TermNoWait(DevNo, IOFlag, NoWaitFlag, RetStatus);   [Note routine name.]
```

COBOL

```
01  DevNo  COMP.
01  IOFlag  COMP.
01  NoWaitFlag  COMP.
01  RetStat  COMP.
    ...
    MONITOR-CALL "TerminalNoWait" USING DevNo, IOFlag, NoWaitFlag, RetStat.
```

FORTRAN

```
      INTEGER  DevNo, IOFlag, NoWaitFlag, RetStatus
      ...
      Monitor_Call('TerminalNoWait', DevNo, IOFlag, NoWaitFlag, RetStatus)
```

ND-100 and ND-500 | All users | All programs

307B **TERMINALNOWAIT** TERMWAIT

Continued from previous page...

PLANC

```
INTEGER : DevNo, IOFlag, NoWaitFlag, RetStatus
...
Monitor_Call('TerminalNoWait', DevNo, IOFlag, NoWaitFlag, RetStatus)
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
IOFlag : W  BLOCK  1
NoWaitFlag : W  BLOCK  1
TerminalNoWait : EQU 37B9 + 307B
        ...
        CALLG TerminalNoWait, 3, DeviceNumber, IOFlag, NoWaitFlag
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     307     %Monitor call TerminalNoWait.
        STA     STAT    %Store status returned.
        JAF     ERROR   %Handle error if register A is non-zero.
        ...
ERROR,  ...             %Error: illegal input.
        ...
STAT,   0
PAR,    DEVNO           %Logical device number of a terminal.
        IOF             %Input/output flag.
        FLAG            %No Wait flag.
        ...
DEVNO,  ...
IOF,    ...
FLAG,   ...
```

ND-100 and ND-500 All users All users

TERMST TerminalStatus 23CB

Gets information about a terminal. The user logged in, the time logged in, the CPU time used, the job being executed, and more is returned.

- You may use the monitor call for batch jobs.

See also @TERMINAL-STATUS.

PARAMETERS

→ Logical device number of a terminal. Use 1 for your own terminal.
← Information about the terminal. The 44 bytes are used as follows:

Byte 0:15	Name of user logged in . Terminated by ' if less than 16 bytes.
16:17	Mode. 1 means command. 2 means program running. 3 means that WaitForRestart has been executed. 4 means that SuspendProgram has been executed.
18:19	State. -1 means no one logged in. 0 means idle batch processor. 1 means active terminal.
20:21	CPU time used in minutes.
22:23	Time logged in in minutes.
24:43	The last command executed. It is terminated by an '.

← Standard error code. See appendix A.

PASCAL

```
DeviceNumber : INTEGER2;
Buffer : ARRAY [0..1] OF BITMAP;
...
TermStatus(DeviceNumber, Buffer);      [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DeviceNumber  COMP.
01  Buffer.
    02  array COMP OCCURS 22 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "TerminalStatus" USING DeviceNumber, Buffer.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DeviceNumber
      INTEGER  Buffer(22)
      ...
      Monitor_Call('TerminalStatus', DeviceNumber, Buffer(1))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

TerminalStatus

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber
BYTE ARRAY : Buffer(0:43)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('TerminalStatus', DeviceNumber, Buffer(0))
```

ASSEMBLY-500

```
DeviceNumber : W  BLOCK  1
Buffer : H  BLOCK  25B
ErrCode : W  BLOCK  1
TerminalStatus : EQU 37B9 + 330B
        ...
        CALLG TerminalStatus, 2, DeviceNumber, Buffer
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     LDN     %Load register T with the logical device number.
        LDA     (BUF    %Load register A with address of return buffer.
        MON     330     %Monitor call TerminalStatus.
        STA     STAT    %Returned status.
        ...
LDN,    1
BUF,    0
*+26/                   %Buffer of 44 bytes.
STAT,   0
```

ND-100 and ND-500 | All users | All users

EXTRN TERMINATIONHANDLING 20CB

Switches termination handling on and off.

- Termination handling for RT programs is either on or off. Background programs may have termination handling on or off for either user break or fatal errors.

See also @ENABLE-TERMINATION-HANDLING and @DISABLE-TERMINATION-HANDLING.

PARAMETERS

→ On or off flag. Use 1 for on and 0 for off.
← Termination. Use 1 for termination handling on user break. Use 2 for fatal errors. Specify 0 for both. Allways use 0 for RT programs.

PASCAL

```
EnDisFlag, Flag : INTEGER2;
...
TerminationHandling(EnDisFlag, Flag);
```

COBOL

```
01  EnDisFlag  COMP.
01  Flag  COMP.
    ...
    MONITOR-CALL "TerminationHandling" USING EnDisFlag, Flag.
```

FORTRAN

```
      INTEGER  EnDisFlag, Flag
      ...
      Monitor_Call('TerminationHandling', EnDisFlag, Flag)
```

ND-100 All users All programs

206B TERMINATIONHANDLING

Continued from previous page...

PLANC

```
INTEGER : EnDisFlag, Flag
...
Monitor_Call('TerminationHandling', EnDisFlag, Flag)
```

ASSEMBLY-500

Not available.

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     206     %Monitor call TerminationHandling.
        ...
PAR,    EDFLAG          %Enable / Disable flag.
        FLAG            %Flag telling user break or fatal error.
        ...
EDFLAG, ...
FLAG,   ...
```

ND-100 All users All users

TROUT TimeOut 267B

Suspends the execution of your program for a given time. The execution then continues after the monitor call. The restart cause is indicated.

- No reserved files or devices are released.
- The execution continues immediately if the program has its restart flag set.
- You may use NoWaitSwitch. Then the program restarts when a break occurs.

See also SuspendProgram, WaitForRestart, and ND500TimeOut.

PARAMETERS

→ Number of time units to suspend the program.
→ The type of time units. 1 = basic time units, ie., 1/50th of a second, 2 = seconds, 3 = minutes, 4 = hours.
← Restart cause. 0 means that the defined time has elapsed. 1 means that a break restarted the program. -1 means that the RT program was scheduled for repeated execution.

PASCAL

```
NoTimeUnits, UnitType, ReturnStatus : INTEGER2;
...
TimeOut(NoTimeUnits, UnitType, ReturnStatus);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  NoTimeUnits  COMP.
01  UnitType  COMP.
01  ReturnStatus  COMP.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "TimeOut" USING NoTimeUnits, UnitType, ReturnStatus.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  NoTimeUnits, UnitType, ReturnStatus
      ...
      Monitor_Call('TimeOut', NoTimeUnits, UnitType, ReturnStatus)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | All programs

267B TimeOut TIMOUT

Continued from previous page...

PLANC

```
INTEGER : NoTimeUnits, UnitType, ReturnStatus
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('TimeOut', NoTimeUnits, UnitType, ReturnStatus)
```

ASSEMBLY-500

```
NoTimeUnits : W  BLOCK  1
UnitType : W  BLOCK  1
RestartReason : W  BLOCK  1
ErrCode : W  BLOCK  1
TimeOut : EQU 37B9 + 267B
        ...
        CALLG TimeOut, 3, NoTimeUnits, UnitType, RestartReason
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     267     %Monitor call TimeOut.
        STA     STAT    %Store status returned.
        ...
STAT,   0
PAR,    TIME            %Number of time units.
        BASE            %Unit base type.
        ...
TIME,   ...
BASE,   ...
```

ND-100 and ND-500 | All users | All users

ERROR ToErrorDevice 1428

Outputs a user defined, real time error. The error message is output on the error device, ie., normally the console. The following is an example of such a message: 23.10.59 ERROR 59 AT XPROG AT 134562; USER ERROR, SUBERROR 4. See appendix A.

- The real time errors number 50-69 can be used this way.

See also WarningMessage, ErrorMessage, and OutMessage.

PARAMETERS

→ Error number. You may use error number 50 to 69. This number is output following ERROR.
→ Suberror number.

PASCAL

```
ErrorNumber, SubErrorNumber : INTEGER2;
...
ToErrorDevice(ErrorNumber, SubErrorNumber);
```

COBOL

```
01  ErrorNumber  COMP.
01  SubErrorNumber  COMP.
    ...
    MONITOR-CALL "ToErrorDevice" USING ErrorNumber, SubErrorNumber.
```

FORTRAN

```
      INTEGER  ErrorNumber, SubErrorNumber
      ...
      Monitor_Call('ToErrorDevice', ErrorNumber, SubErrorNumber)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

142B ToErrorDevice

Continued from previous page...

PLANC

```
INTEGER : ErrorNumber, SubErrorNumber
...
Monitor_Call('ToErrorDevice', ErrorNumber, SubErrorNumber)
```

ASSEMBLY-500

```
ErrorNumber : W  BLOCK  1          %In ASCII characters.
SubErrorNumber : W  BLOCK  1
ToErrorDevice : EQU 37B9 + 142B
        ...
        CALLG ToErrorDevice, 2, ErrorNumber, SubErrorNumber
```

MAC

```
        LDA     ERRNO    %Error number.
        LDT     SUBER    %Sub-error number.
        MON     142      %Monitor call ToErrorDevice.
        ...
ERRNO,  ...              %Error number as two ASCII characters.
SUBER,  ...              %Suberror number as two ASCII characters.
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

TransferData

Transfers data between physical memory and a mass storage device, eg., a disk. You may perform various device control functions. This monitor call is mainly used by the operating system itself.

- The program must run on ring 2. You may use any page table. Both demand and non-demand segments are legal. Memory areas for DMA transfer must be fixed contiguously.
- For calls to magnetic tape or other devices where the last parameter contains number of bytes/words read or other return parameters, the parameter list must be in resident memory, ie., an area with fixed page table contents.
- Parameters are fetched via the normal page table.
- The physical memory area must be contiguous. Older versions of magnetic tapes or disk controllers cannot cross physical memory bank boundaries of 128 Kbytes. These magnetic tapes have ND numbers less than ND-537. The disks have ND numbers less than ND-559.
- Only two programs may execute TransferData at a time.

See also DataTransfer, ReadFromFile, and WriteToFile.

PARAMETERS

→ Logical device number. See appendix B.
→ Function code. See the tables on the following pages.
→ Physical memory address.
→ Sector address on the disk. See the tables on the following pages.
→ Number of sectors to transfer.
← Error code. Negative if error. Contains a hardware status.

PASCAL

```
DeviceNumber, Func, RetStatus : INTEGER2;
MemAddr, BlockAddr, NoOfBlocks : LONGINT;
...
TransferData(DeviceNumber, Func, MemAddr, BlockAddr, NoOfBlocks, RetStatus);
```

COBOL

```
01  DeviceNumber  COMP.
01  Func  COMP.
01  ReturnStatus  COMP.
01  MemAddr  COMP PIC S9(10).
01  BlockAddr  COMP PIC S9(10).
01  NoOfBlocks  COMP PIC S9(10).
    ...
    MONITOR-CALL "TransferData" USING DeviceNumber, Func, MemAddr,
                                BlockAddr, NoOfBlocks, ReturnStatus.
```

FORTRAN

```
      INTEGER  DeviceNumber, Func, ReturnStatus
      INTEGER*4  MemAddr, BlockAddr, NoOfBlocks
      ...
      Monitor_Call('TransferData', DeviceNumber, Func, MemAddr,
     C                             BlockAddr, NoOfBlocks, ReturnStatus)
```

ND-100 | User RT and user SYSTEM | RT programs

335B TRANSFERDATA EXCEPT

Continued from previous page...

PLANC

```
INTEGER : DeviceNumber, Func, ReturnStatus
INTEGER4 : MemAddr, BlockAddr, NoOfBlocks
...
Monitor_Call('TransferData', DeviceNumber, Func, MemAddr, &
                             BlockAddr, NoOfBlocks, ReturnStatus)
```

ASSEMBLY-500

Not available.

MAC

```
        LDT     DEVNO     %Logical device number.
        LDA     (PAR      %Load register A with address of parameter list.
        MON     335       %Monitor call DataTransfer.
        JAN     ERROR     %Error if register A is negative.
        ...               %Continue with processing.
ERROR,  ...               %Error number in register A.
        ...
DEVNO,  ...
PAR,    FUNC              %Function code etc.
        DMEM              %Memory address.
        BLOCK             %Block address.
        NOBLK             %Number of blocks to transfer.
FUNC,   ...
DMEM,   0;0               %These three parameters are 32-bit long. Use
BLOCK,  0;0               %the most significant word for 16-bit parameters.
NOBLK,  0;0               %
```

The function code for ECC and HAWK disks:

Bits	Value	Explanation
0-5	0B	Read
	1B	Write
	2B	Read block for parity check
	3B	Compare block
6-10	(0-3B)	Mass storage unit number.
11-13		Most significant 3 bits of the sector address. Not PHOENIX.
11-13	(0-4B)	Subunit number. PHOENIX only.
14	1B	PHOENIX
	0B	NOT PHOENIX.

The sector address for ECC and HAWK disks:

Bit	Value	Meaning
17	0	Removable surface.
	1	Fixed surface. HAWK and PHOENIX only.

The function code for floppy disk controllers. The asterisk (*) means that code applies to the new controllers only.

Bits	Value	Explanation
0-5	0B	Read.
	1B	Write.
	2B	Read without transfering data, ie., find end-of-file mark.
	3B	Compare block.
	5B	Write end-of-file mark, (deleted record). *
	10B	Advance to end-of-file mark.
	11B	Reverse to end-of-file mark.
	13B	Rewind.
	15B	Backspace one record.
	16B	Advance one record.
	20B	Read status.
	24B	Read last status.
	40B	Select format.
	41B	Format floppy
	42B	Read format.
	43B	Read deleted record.
	44B	Write deleted record.
	54B	Copy floppy disk. *
	55B	Format floppy disk track. *
	56B	Check floppy disk. *
6-10	(0-3B)	Unit number.

The sector address should contain the number of blocks to transfer if read or write functions. The select format function requires the format number in the sector address.

Function code of VERSATEC on DMA:

Bits	Value	Explanation
0-5	20B	Read status.
	21B	Clear VERSATEC.
	24B	Read last status.
	30B	Set alphanumeric mode.
	31B	Set graphic mode.
	32B	Give form feed.

Magnetic tape function code, floppy disk & disk

32 bits	61 - read	w/32 bits disk add
word count	61 - write	w/32 bits disk add
	63 - compare	w/32 bits disk add

ADR100 TRANSLATEADDRESS 430B

Translates an ND-500 logical address to an ND-100 physical address. Use this monitor call to set up communication areas between the two CPUs.

- Applied only to data addresses, not program addresses.

PARAMETERS

→ ND-500 array.
← Physical memory in the ND-100.

PASCAL

```
ND500Array, ND100PhysWordAddr : LONGINT;
...
TranslateAddress(ND500Array, ND100PhysWordAddr);
```

COBOL

```
01  ND500Array  COMP.
01  ND100PhysWordAddr  COMP.
    ...
    MONITOR-CALL "TranslateAddress" USING ND500Array, ND100PhysWordAddr.
```

FORTRAN

```
      INTEGER  ND500Array, ND100PhysWordAddr
      ...
      Monitor_Call('TranslateAddress', ND500Array, ND100PhysWordAddr)
```

ND-500 | All users | All programs

430B **TRANSLATEADDRESS** ADR100

Continued from previous page...

PLANC

```
INTEGER : ND500Array, ND100PhysWordAddr
...
Monitor_Call('TranslateAddress', ND500Array, ND100PhysWordAddr)
```

ASSEMBLY-500

```
ND500Array : W  BLOCK  1
ND100PhysWordAddr : W  BLOCK  1
TranslateAddress : EQU 37B9 + 430B
        ...
        CALLG TranslateAddress, 2, ND500Array, ND100PhysWordAddr
```

MAC

Not available.

ND-500 All users All users

UEADM UEADMINISTRATOR 3231B

Controls functions for User Environment. This monitor call should not be used in ordinary programs.

PARAMETERS

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

32XB **UEADMINISTRATOR** UEADM

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Internal use only.

MAC

Internal use only.

ND-100 and ND-500 User RT and user SYSTEM RT programs

UELOG	UELOGIN	320B

This monitor call controls log in under USER-ENVIRONMENT. It is not intended for ordinary use.

PARAMETERS

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 and ND-500	All users	All programs

32MB	UELOGIN	UELOGI

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Internal use only.

MAC

Internal use only.

ND-100 and ND-500	All users	All users

UnFixSegment

Releases a fixed segment. Its pages may then be swapped from physical memory to the disk. Use FixScattered or FixContiguous to fix the segment.

- You must use UnFixSegment or @UNFIX to allow the RT-LOADER to clear the segment.

See also FixInMemory, MemoryAllocation, FixIOArea, and @UNFIX.

PARAMETERS

→ Segment number to be released.

PASCAL

```
SegmentNumber : INTEGER2;
...
UnFixSegment(SegmentNumber);
```

COBOL

```
01  SegmentNumber  COMP.
    ...
    MONITOR-CALL "UnFixSegment" USING SegmentNumber.
```

FORTRAN

```
      INTEGER  SegmentNumber
      ...
      Monitor_Call('UnFixSegment', SegmentNumber)
```

ND-100 and ND-500 | User RT and user SYSTEM | RT programs

116B UnFixSegment UNFIX

Continued from previous page...

PLANC

```
INTEGER : SegmentNumber
...
Monitor_Call('UnFixSegment', SegmentNumber)
```

ASSEMBLY-500

```
SegmentNumber : W  BLOCK  1
UnFixSegment : EQU 37B9 + 116B
        ...
        CALLG UnFixSegment, 1, SegmentNumber
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     116     %Monitor call UnFixSegment.
        ...
PAR,    SEGNO           %Segment number to be unfixed.
        ...
SEGNO,  ...
```

ND-100 and ND-500 User RT and user SYSTEM RT programs

USCNT UserControl 26GM

Sets user control of a device.

- This monitor call is only available with COSMOS.

See also SystemControl.

PARAMETERS

← Standard error code. See appendix A.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 and ND-500 All users All programs

260B **USERCONTROL** USCNT

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

```
DeviceNo : W  BLOCK 1
Code : W BLOCk 1
ErrCode : W BLOCK 1
UserControl : EQU 37B9 + 260B
        ...
        CALLG UserControl, 2, DeviceNo, Code
        IF K GO Error
        ...
Error,  W1 =: ErrCode
```

MAC

```
        MON      260      %Monitor call UserControl.
```

ND-100 and ND-500 All users All users

USD UserDef0 170B

User defined monitor call. You can implement up to 8 monitor calls yourself. These are named UserDef0, UserDef1,... UserDef7. The short names are US0, US1,...US7. To do this you need good knowledge of SINTRAN III. See the SINTRAN III listing part one and two.

- Use the DMAC subsystem or the @LOOK-AT command to patch in the MAC instructions of your monitor call. Place the code in physical memory after the SINTRAN III symbol 7ENDC. You ought to do this from a mode file. The monitor call is lost in a warm start.
- Use the SINTRAN-SERVICE-PROGRAM command DEFINE-USER-MONITOR-CALL to define the entry point of the monitor call.
- There are other ways of implementing user defined monitor calls.
- Some monitor calls are no longer in use. You may use these for user defined monitor calls.
- If the subsystems TPS is installed, the user defined monitor calls 170 to 175 may not be used.

PARAMETERS

Both input and output parameters are user defined. You are adviced to transfer parameters in the same way as SuspendProgram, InString, WriteToFile, etc.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 All users All programs

17CB	USERDEF0	000

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Not available.

MAC

Depends on the monitor call.

ND-100	All users	All users

RTVT WaitForRestart 135B

Sets the RT program in a waiting state. It is restarted by StartRTProgram or @RT. Execution continues after the monitor call.

- The RT program restarts immediately if its restart flag is set.
- No reserved devices or files are released.

See also SuspendProgram. It makes programs wait for a specified time.

PARAMETERS

This monitor call has no parameters.

PASCAL

```
WaitForRestart;
```

COBOL

```
    MONITOR-CALL "WaitForRestart".
```

FORTRAN

```
      Monitor_Call('WaitForRestart')
```

ND-100 and ND-500 User RT and user SYSTEM RT programs

135B **WaitForRestart** RTWT

Continued from previous page...

PLANC

```
Monitor_Call('WaitForRestart')
```

ASSEMBLY-500

```
WaitForRestart : EQU 37B9 + 135B
        CALLG WaitForRestart, 0
```

MAC

```
        MON     135     %Monitor call WaitForRestart
```

ND-100 and ND-500 User RT and user SYSTEM RT programs

ERRMSG WarningMessage 64B

Outputs a file system error message. Appendix A shows the messages connected to each error code. The error code is input. The program continues.

- The error message is output to the terminal. In batch jobs, mode jobs, and RT programs it is output to the error device. The error device is normally the console.
- Error code 0 is illegal.

See also GetErrorMessage and ErrorMessage. ErrorMessage writes out the error message and terminates the program.

PARAMETERS

→ Error code of the message to be printed. Use octal numbers.

PASCAL

```
ErrCode : INTEGER2;
...
WarningMessage(ErrCode);
```

COBOL

```
01  ErrCode  COMP.
    ...
    MONITOR-CALL "WarningMessage" USING ErrCode.
```

FORTRAN

```
      INTEGER  ErrCode
      ...
      Monitor_Call('WarningMessage', ErrCode)
```

ND-100 and ND-500 All users All programs

64B **WARNINGMESSAGE** ERRNO

Continued from previous page...

PLANC

```
INTEGER : ErrCode
...
Monitor_Call('WarningMessage', ErrCode)
```

ASSEMBLY-500

```
ErrCode : W  BLOCK  1
WarningMessage : EQU 37B9 + 64B
        ...
        CALLG WarningMessage, 1, ErrCode
```

MAC

```
        LDA     ERRNO    %Error number of error message to be printed.
        MON     64       %Monitor call WarningMessage.
        ...
ERRNO,  ...
```

ND-100 and ND-500 All users All users

WPAGE **WRITEBLOCK** 108

Writes randomly to a file. You Write one block at a time. The file must be opened for random write access.

- The standard block size is 512 bytes. You can change this with SetBlockSize. The first block is number 0.

See also SetStartBlock, SetBlockSize, WriteDiskPage, WriteFromFile, and ReadBlock. WriteFromFile is the most efficient way to read randomly.

PARAMETERS

→ File number. See OpenFile.
→ Block number.
→ Transferred block.
← Standard error code. See appendix A.

PASCAL

```
FileNumber, BlockNumber : INTEGER2;
Buffer : ARRAY [0..15] OF BITMAP;
...
WriteBlock(FileNumber, BlockNumber, Buffer);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNumber  COMP.
01  BlockNumber  COMP.
01  Buffer.
    02  array COMP OCCURS 256 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "WriteBlock" USING FileNumber, BlockNumber, Buffer.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNumber, BlockNumber
      INTEGER  Buffer(256)
      ...
      Monitor_Call('WriteBlock', FileNumber, BlockNumber, Buffer(1))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 All users All programs

WRITEBLOCK

Continued from previous page...

PLANC

```
INTEGER : FileNumber, BlockNumber
BYTE ARRAY : Buffer(0:511)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('WriteBlock', FileNumber, BlockNumber, Buffer(0))
```

ASSEMBLY-500

Not available.

MAC

```
        LDT     FILNO    %File number returned from earlier open.
        LDA     BLKNO    %Block number.
        LDX     (BUFF    %Address of buffer to receive block read.
        MON     10       %Monitor call WriteBlock.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
FILNO,  ...
BLKNO,  ...
BUFF,   0
*+400/                   %Make a buffer of 256 words, 1 block.
```

ND-100 All users All users

WDIRM — WRITEDIRENTRY — 311B

Changes the information about a directory. The complete contents of the directory entry is set. The SINTRAN III SYSTEM SUPERVISOR (ND-30.003) describes the file system in more detail.

- The directory must be entered.
- The directory must be reserved.

See also GetDirUserIndexes, and GetDirEntry.

PARAMETERS

→ The directory index. See GetDirUserIndexes.
→ The 48 byte directory entry. See appendix C.
← Standard error code. See appendix A.

PASCAL

```
DirIndex : INTEGER2;
DirEntry : ARRAY [0..1] OF BITMAP;
...
SetDirEntry(DirIndex, DirEntry);      [Note routine name.]
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DirIndex  COMP.
01  DirEntry.
    02  array COMP OCCURS 24 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "WriteDirEntry" USING DirIndex, DirEntry.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DirIndex
      INTEGER  DirEntry(24)
      ...
      Monitor_Call('WriteDirEntry', DirIndex, DirEntry(1))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | User SYSTEM | All programs

311B **WRITEDIRENTRY** WDIREN

Continued from previous page...

PLANC

```
INTEGER : DirIndex
BYTE ARRAY : DirEntry(0:47)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('WriteDirEntry', DirIndex, DirEntry(0))
```

ASSEMBLY-500

```
DirIndex : W  BLOCK  1
DirEntry : W  ARRAY 24
ErrCode : W  BLOCK  1
WriteDirEntry : EQU 37B9 + 311B
        ...
        CALLG WriteDirEntry, 2, DirIndex, DirEntry
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DIRIX    %Directory index.
        LDX     (ENTRY   %Address of buffer containing directory entry.
        MON     311      %Monitor call WriteDirEntry.
        JMP     ERROR    %Error return from monitor call.
        ...              %Normal return.
ERROR,  ...              %Error number in register A.
        ...
DIRIX,  ...
ENTRY,  ...              %
        ...              %A buffer of 24 word.
```

ND-100 and ND-500 User SYSTEM All users

WRITEDISKPAGE

Writes to one or more pages in a directory. Any page can be written to.

- The directory must be reserved with ReseveDir.

See also ReadDiskPage.

PARAMETERS

→ Directory index. See GetDirUserIndexes.
→ Address of buffer with pages to transfer.
→ Address of the destination pages on the disk.
→ Number of pages to transfer. Each page is 2048 bytes.

PASCAL

```
DirIndex, NoOfPages : INTEGER2;
Buffer : ARRAY [0..63] OF BITMAP;
PageAddr : LONGINT;
...
WriteDiskPage(DirIndex, Buffer, PageAddr, NoOfPages);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  DirIndex  COMP.
01  NoOfPages  COMP.
01  PageAddr  COMP PIC S9(10).
01  Buffer.
    02  array COMP OCCURS 1024 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL   "WriteDiskPage"   USING   DirIndex,   Buffer,   PageAddr,
NoOfPages.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  DirIndex, NoOfPages
      INTEGER*4  PageAddr
      INTEGER  Buffer(1024)
      ...
      Monitor___Call('WriteDiskPage',   DirIndex,   Buffer(1),   PageAddr,
NoOfPages)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | User RT and user SYSTEM | Background programs

271B WRITEDISKPAGE

Continued from previous page...

PLANC

```
INTEGER : DirIndex, NoOfPages
INTEGER4 : PageAddr
BYTE ARRAY : Buffer(0:2047)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('WriteDiskPage', DirIndex, Buffer(0), PageAddr, NoOfPages)
```

ASSEMBLY-500

```
DirIndex : W  BLOCK  1
PageAddr : W  BLOCK  1
NoOfPages : W  BLOCK  1
Buffer : W  BLOCK 1024    %Must start on an even byte address.
ErrCode : W  BLOCK  1
WriteDiskPage : EQU 37B9 + 271B
        ...
        CALLG WriteDiskPage, 4, DirIndex, Buffer, PageAddr, NoOfPages
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDT     DIRIX   %Directory index.
        LDX     (BUFF   %Address of buffer containing data to be written.
        LDA     PAGES   %Number of pages to transfer.
        COPY    SA  DD
        LDA     (PAGNO  %Address of double word with disk page address.
        MON     271     %Monitor call WriteDiskPage.
        JMP     ERROR   %Error return from monitor call.
        ...             %Normal return.
ERROR,  ...             %Error number in register A.
        ...
DIRIX,  ...
BUFF,   0
*+4000/                 %Make a buffer of 2048 words, 2 pages.
PAGNO,  ...             %A double word.
        ...             %
PAGES,  2               %Transfer 2 pages.
```

ND-100 and ND-500 | User RT and user SYSTEM | Background programs

WRITESCRATCHFILE

Writes randomly to the scratch file. One block is transferred. There is one scratch file connected to each terminal. It is opened for random read and write access when you log in. Its file number is 100B.

- The standard block size is 512 bytes. You can change this with SetBlockSize. The first block is number 0.

See also WriteBlock, WriteToFile, and ReadScratchFile.

PARAMETERS

→ Block number to start the writing from.
→ The data to be transferred.
← Standard error code. See appendix A.

PASCAL

```
BlockNumber : INTEGER2;
Buffer : ARRAY [0..15] OF BITMAP;
...
WriteScratchFile(BlockNumber, Buffer);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  BlockNumber  COMP.
01  Buffer.
    02  array COMP OCCURS 256 TIMES.
01  ErrCode  COMP.
    ...
    MONITOR-CALL "WriteScratchFile" USING BlockNumber, Buffer.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  BlockNumber
      INTEGER  Buffer(256)
      ...
      Monitor_Call('WriteScratchFile', BlockNumber, Buffer(1))
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 | All users | Background programs

WRITESCRATCHFILE

Continued from previous page...

PLANC

```
INTEGER : BlockNumber
BYTE ARRAY : Buffer(0:511)
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('WriteScratchFile', BlockNumber, Buffer(0))
```

ASSEMBLY-500

Not available.

MAC

```
        LDT     BLKNO     %Block number to be written into.
        LDX     (BUFF     %Address of buffer containing data to be written.
        MON     6         %Monitor call WriteScratchFile.
        JMP     ERROR     %Error return from monitor call.
        ...               %Normal return.
ERROR,  ...               %Error number in register A.
        ...
BLKNO,  ...
BUFF,   0
*+400/                    %Make a buffer of 256 words, 1 block.
```

ND-100 All users Background programs

WRITE — WRITETOFILE — 12CB

Writes any number of bytes to a file. The read operation must start at the beginning of a block. The file must be opened for random write access.

- The standard block size is 512 bytes. You can change this with SetBlockSize. The first block is number 0.
- You may use access code D for direct transfer. Then the block size must be a multiple of the page size. The number of bytes to transfer must be a multiple of the block size. The data must be fixed contiguously in memory.
- Peripheral files are always written to sequentially.
- Data transfer across segment or RT common limits is illegal.

See also SetStartBlock, SetBlockSize, WriteDiskPage, WriteBlock, and ReadFromFile.

PARAMETERS

→ File number. See OpenFile.
→ Wait flag. Use 0 to suspend the program until the transfer is completed. Other values make the program continue. You may check that the transfer is completed by AwaitFileTransfer. Background programs must always wait until data is transferred.
→ Data to be transferred.
→ Block number to start writing from. Use -1 to write to the next block.
→ Number of bytes to read.
← Standard error code. See appendix A.

PASCAL

```
FileNo, ReturnFlag, BlockNo : INTEGER2;
NoOfBytes : LONGINT;
Buff : ARRAY [0..15] OF BITMAP;
...
WriteToFile(FileNo, ReturnFlag, Buff, BlockNo, NoOfBytes);
IF ErrCode >< 0 THEN ...
```

COBOL

```
01  FileNo  COMP. 01  ReturnFlag  COMP.
01  Buff.
    02  array COMP OCCURS 256 TIMES.
01  BlockNo  COMP. 01  NoOfBytes  COMP PIC S9(10).
01  ErrCode  COMP.
    ...
    MONITOR-CALL "WriteToFile" USING FileNo, ReturnFlag, Buff,
                                     BlockNo, NoOfBytes.
    CALL "CbError" USING ErrCode.
    IF ErrCode NOT = 0 GO ...
```

FORTRAN

```
      INTEGER  FileNo, ReturnFlag, BlockNo
      INTEGER  Buff(256)
      INTEGER*4  NoOfBytes
      ...
      Monitor_Call('WriteToFile', FileNo, ReturnFlag, Buff(1),
     C                            BlockNo, NoOfBytes)
      IF (ErrCode .NE. 0) THEN ...
```

ND-100 and ND-500 | All users | Background programs

120B WRITETOFILE WFILE

Continued from previous page...

PLANC

```
INTEGER : FileNo, ReturnFlag, BlockNo
BYTE ARRAY : Buff(0:511)
INTEGER4 : NoOfBytes
...
ON ROUTINEERROR DO
    IF ErrCode >< 0 THEN ...
ENDON
Monitor_Call('WriteToFile', FileNo, ReturnFlag, Buff(0), BlockNo, NoOfBytes)
```

ASSEMBLY-500

```
FileNo : W  BLOCK  1
ReturnFlag : W  BLOCK  1
Buff : W  BLOCK 256
BlockNo : W  BLOCK  1
NoOfBytes : W  BLOCK  1
ErrCode : W  BLOCK  1
WriteToFile : EQU 37B9 + 120B
        ...
        CALLG WriteToFile, 5, FileNo, ReturnFlag, Buff, BlockNo, NoOfBytes
        IF K GO ERROR
        ...
ERROR : W1 =: ErrCode                          %ErrorCode in W1 register.
```

MAC

```
        LDA     (PAR    %Load register A with address of parameter list.
        MON     120     %Monitor call WriteToFile.
        JAF     ERROR   %Do error handling if register A is non-zero.
        ...
ERROR,  ...             %Error number in register A.
        ...
PAR,    FILNO           %File number returned from earlier open.
        FLAG            %Wait flag.
        BUFF            %Buffer containing data to be written.
        BLKNO           %Block number where writing is started.
        COUNT           %Number of words to transfer.
        ...
FILNO,  ...
FLAG,   ...
BUFF,   0
*+1000/                 %Make a buffer of 512 words, 4 blocks.
BLKNO   ...
COUNT,  0               %A double word.
        1000            %Transfer 512 words.
```

ND-100 and ND-500 | All users | Background programs

XMSG XMSGFUNCTION 200B

Performs various data communication functions. All types of programs may communicate through this monitor call. The programs may be in different computers in a network.

- The monitor call operates by sending and receiving messages.
- The SINTRAN III COMMUNICATION GUIDE (60.134) describes XMSGFunction.
- The COSMOS PROGRAMMER GUIDE (60.164) describes the communication facilities offered to high-level languages.

See also HDLCFunction.

PARAMETERS

The parameters varies from function to function.

PASCAL

Not available.

COBOL

Not available.

FORTRAN

Not available.

ND-100 All users All programs

200B XMSGFUNCTION XMSG

Continued from previous page...

PLANC

Not available.

ASSEMBLY-500

Not available.

MAC

See the SINTRAN III COMMUNICATION GUIDE (60.134).

ND-100 All users All users

APPENDIX A: ERROR MESSAGES

This is the SINTRAN III error messages. Real time errors are output as shown below. The message is displayed on the terminal if the error relates to an interactive program. Other messages are written to the error device, normally the console.

```
23.59.57 ERROR 14 IN BAK07 AT 114721; OUTSIDE SEGMENT BOUNDS
```

Error number 14 occurred at 23.59.57 o'clock. The RT program BAK07 caused the error at the memory address 114721. Some error messages output additional information. The erroneous RT program is in most cases aborted. Error messages from background programs are displayed as texts only, eg., NO SUCH FILE.

Error no	Meaning	Additional information		Program Aborted
00	Illegal monitor call			yes
01	Bad RT program address			yes
02	Wrong priority in PRIOR			yes
03	Bad memory page	page no.		
04	Internal interrupt on direct task level	level	bit no.	
06	Batch input error	error no.		yes
07	Batch output error	error no.		yes
08	Batch system error	error no.	L register	yes
09	Illegal parameter in CLOCK			yes
10	Illegal parameter in ABSET			yes
11	Illegal parameter in UPDAT			yes
12	Illegal time parameters			yes
13	Page fault for non-demand	page no.		yes
14	Outside segment bounds	page no.		yes
15	Illegal segment number	segment no.		yes
16	Segment not loaded	segment no.		yes
17	Fixing demand	segment no.		yes
18	Too many fixed pages	segment no.		yes
19	Too big segment	segment no.		yes
20	Disk transfer error	hardware device no.	unit	no yes if segment transfer
21	Disk transfer error	last 16 bits of sector address	hardware status	no
22	False interrupt	level	ident code	no
23	Device error	hardware device no.	hardware status	no
25	Already fixed	segment no.		yes

Error no	Meaning	Additional information		Program Aborted
26	Device time-out	hardware device	unit no.	no no.
27	Illegal parameter in CONCT			yes
28	Space not available	segment no.		yes
29	MON 64 and MON 65	error no.	(see NORD File System ND-60.122)	yes
30	Divide by zero			yes
31	Permit violation			yes
32	Ring violation			yes
33	HDLC driver, fatal error			yes
34	Illegal instruction			yes
35	REENTRANT-FTN stack error			yes
36	Privileged instructon			yes
37	IOX error address level			no
38	Memory parity error	PEA reg.	PES reg.	yes
39	Memory out of range	PEA reg.	PES reg.	yes
40	Power fail			no
41	Illegal error code in ERMON			yes
42	Overlapping segments	segments		yes
44	Corrected memory error	PEA reg.	PES reg.	no
45	Not demand segments			yes
46	XMSG fatal error, internal error or inconsistency	XMSG error code	physical address	yes
47	XMSG user error	calling level		yes
48	False BEX interrupt			
49	Remote power fail interrupt			
50-69	User defined error (MON 142)	error no.	suberror no	no
70	BEX parity error			
71	False MPM4 interrupt	busc no.	hardware status	no
72	MPM4 power fail interrupt	busc no.		no
73	MPM4 memory out of range	busc no.	lower limit	no
74	MPM4 memory error	local PES	local PEA	no
75	MPM4 parity error	busc no.	lower limit	no
76	MPM4 write parity error	busc no.	port code no	
90	FORTRAN run-time error	error no.		no
91	FORTRAN I/O error	error no.		no
100	FTN library error			

The following table shows the SINTRAN III background errors. The octal error numbers are returned from monitor calls. Use the monitor calls ErrorMessage and WarningMessage to display the corresponding error message. Error numbers above 1000B are only returned by programs running on the ND-500.

Error number Octal	Decimal	Meaning
000	000	Not used
001	001	Not used
002	002	Bad file number
003	003	End of file
004	004	Card Reader Error (card read)
005	005	Device not reserved
006	006	Not used
007	007	Card Reader Error (card not read)
010	008	Not used
011	009	Not used
012	010	End of device (time-out)
013	011	Not used
014	012	Not used
015	013	Not used
016	014	Not used
017	015	Not used
020	016	Not used
021	017	Illegal character in parameter
022	018	No such page
023	019	Not decimal number
024	020	Not octal number
025	021	You are not authorized to do this
026	022	Directory not entered
027	023	Ambiguous directory name
030	024	No such device name
031	025	Ambiguous device name
032	026	Directory entered
033	027	No such logical unit
034	028	Unit occupied
035	029	Master block transfer error
036	030	Bit file transfer error
037	031	No more tracks available
040	032	Directory not on specified unit
041	033	Files opened on this directory
042	034	Main directory not last one released
043	035	No main directory
044	036	Too long parameter
045	037	Ambiguous user name
046	038	No such user name
047	039	No such user name in main directory(s)
050	040	Attempt to create too many users
051	041	User already exists
052	042	User has files

Error number Octal	Decimal	Meaning
053	043	User is entered
054	044	Not so much space unreserved in directory
055	045	Reserved space already used
056	046	No such file name
057	047	Ambiguous file name
060	048	Wrong password
061	049	User already entered
062	050	No user entered
063	051	Friend already exists
064	052	No such friend
065	053	Attempt to create too many friends
066	054	Attempt to create yourself as friend
067	055	Continuous space not available
070	056	Not directory access
071	057	Space not available to expand file
072	058	Space already allocated
073	059	No space in default directories
074	060	No such file version
075	061	No more pages available for this user
076	062	File already exists
077	063	Attempt to create too many files
100	064	Outside device limits
101	065	No previous version
102	066	File not continuous
103	067	File type already defined
104	068	No such access code
105	069	File already opened
106	070	Not write access
107	071	Attempt to open too many files
110	072	Not write and append access
111	073	Not read access
112	074	Not read, write and common access
113	075	Not read and write access
114	076	Not read and common access
115	077	File reserved by another user
116	078	File already opened for write by you
117	079	No such user index
120	080	Not append access
121	081	Attempt to open too many mass-storage files
122	082	Attempt to open too many files
123	083	Not opened for sequential write
124	084	Not opened for sequential read
125	085	Not opened for random write
126	086	Not opened for random read
127	087	File number out of range
130	088	File number already used
131	089	No more buffer space
132	090	No file opened with this number
133	091	Not mass-storage file
134	092	File used for write

Error number Octal	Decimal	Meaning
135	093	File used for read
136	094	File only opened for sequential read or write
137	095	No scratch file opened
140	096	File not reserved by you
141	097	Transfer error
142	098	File already reserved
143	099	No such block
144	100	Source and destination equal
145	101	Illegal on tape device
146	102	End of tape
147	103	Device unit reserved for special use
150	104	Not random access on tape files
151	105	Not last file on tape
152	106	Not tape device
153	107	Illegal address reference in monitor call
154	108	Source empty
155	109	File already open by another user
156	110	File already open for write by another user
157	111	Missing parameter
160	112	Two pages must be left unreserved
161	113	No answer from remote computer
162	114	Device cannot be reserved
163	115	Overflow in read
164	116	DMA error
165	117	Bad data block
166	118	CONTROL MODUS word error
167	119	Parity error
170	120	LRC error
171	121	Device error (read-last-status to get status)
172	122	No device buffer available
173	123	Illegal mass-storage unit number
174	124	Illegal parameter
175	125	Write protect violation
176	126	Error detected by read after write
177	127	No EOF mark found
200	128	Cassette not in position
201	129	Illegal function code
202	130	Time-out (no data block found)
203	131	Paper fault
204	132	Device not ready
205	133	Device already reserved
206	134	Not peripheral file
207	135	No such queue entry
210	136	Not so much space left
211	137	No spooling for this device
212	138	No such queue
213	139	Queue empty
214	140	Queue full
215	141	Not last used by you
216	142	No such channel name

Error number Octal	Decimal	Meaning
217	143	No remote connection
220	144	Illegal channel
221	145	Channel already reserved on remote computer
222	146	No remote file processor
223	147	Formatting error
224	148	Incompatible device sizes
225	149	Remote processor not available
226	150	Tape format error 227 151 Block count error
230	152	Volume not on specified unit
231	153	Not deleted record
232	154	Device error
233	155	Error in object entry
234	156	Odd number of bytes (right byte in last word insignificant)
235	157	Error in backspace/forward space print
236	148	Block format error
237	159	Overflow in write
240	160	Illegal device type
241	161	Segment not contiguously fixed
242	162	Segment not fixed
243	163	Approaching end of accounting file
244	164	End of accounting file encountered
245	165	No more unused spooling files available
246	166	Inconsistent directory
247	167	Object entry not used
250	168	User does not exist
251	169	Directory not reserved
252	170	Not a multiplum of hardware block size
253	171	Not indexed file
254	172	Illegal floppy format
255	173	File not opened
256	174	File already opened for write by you
257	175	User does not exist in the same main directory as you
300-377		Reserved for special use
1000	512	ND-500 open file table is full
1001	513	File is neither contiguous nor magnetic tape
1002	514	ND-500 open file table for direct transfer is full
1003	515	Error in monitor call
1004	516	Odd byte address
1005	517	Odd bytecount
1006	518	Too big bytecount
1007	519	Bytecount not modulo sector size in direct transfer
1010	520	Address outside file limits in direct transfer
1011	521	Block address not modulo sector size in direct transfer
1012	522	Hardware status error in direct transfer
1013	523	Illegal monitor call number
1014	524	DC address not legal on magnetic tape

Error number Octal	Decimal	Meaning
1015	525	Wrong number of parameters in monitor call
1016	526	Byte pointer not modulo sector size in direct transfer
1017	527	Data area cannot be put in a 64K SINTRAN III segment
1020	528	Segment not modifiable
1021	529	Bytecount not modulo block size in direct transfer
1022	530	Illegal operation on file connected to a segment
1023	531	File already connected to a segment
1024	532	All logical data segments used
1025	533	Logical data segment already used
1026	534	Block size not modulo sector size
1027	535	Address outside program segment
1030	536	Address outside data segment
1031	537	Trying to write segment back on system swap file
1032	538	Illegal memory type of specified area
1033	539	Max global fix
1034	540	Error in absolute fix
1035	541	Other segments has user fixed pages in the specified area
1036	542	Other segments has system fixed pages in the specified area
1037	543	Impossible to do fix contiguous because of pages already system fixed
1040	544	Impossible to do fix contiguous because of pages already user fixed
1041	545	No contiguous area available because of system fixing of other segments.
1042	546	No contiguous area available because of user fixing of other segments.
1043	547	Impossible to do contiguous fix. Area greater than physical memory.
1044	548	Not enough memory reserved by the ND-500
1045	549	Trying to fix pages reserved by the ND-500
1046	550	Segment not in use
1047	551	The process has no before image log segment
1050	552	No swap-file part available
1051	553	Swapping space not available
1052	554	No free physical segment
1053	555	Segment not modifiable
1054	556	Illegal process number
1055	557	Swap device error
1056	558	Privileged monitor call
1057	559	Illegal logical segment number
1060	560	No such process
1061	561	Illegal address

APPENDIX B: LOGICAL DEVICE NUMBERS

SINTRAN III uses the following logical device numbers. The first table is an overview of the following tables.

Octal logical device number	Decimal logical device number	Group of devices
0-77	0-63	Character devices, terminals
100-177	64-127	Mass storage files
200-277	128-191	Non-internal devices
300-377	192-255	User semaphores
400-477	256-319	Process control devices/connect devices
500-577	320-383	System devices
600-677	384-447	Communication devices, ND-NET
700-777		NORDCOM devices and other special devices
1000-1077		Extension of character devices
1100-1777		System devices
2000-2077		Terminals

Logical device number 100-177 is assigned to files when they are opened. These logical device numbers are called file numbers. See OpenFile.

Octal logical device number	Decimal logical device number	Character devices
0	0	In InByte input from command buffer, else dummy
1	1	Background, "own terminal" RT, Terminal 1
2	2	Tape reader 1 (Console)
3	3	Tape punch 1
4	4	Card reader 1
5	5	Line printer 1
6	6	Synchronous modem 1
7	7	Terminal 17
10	8	Plotter 1
11	9	Terminal 2
12	10	Tape reader 2
13	11	Tape punch 2
14	12	Bus switch device
15	13	Line printer 2
16	14	Synchronous modem 2
17	15	Terminal 18
20	16	Cassette drive 1
21	17	Cassette drive 2
22	18	Versatec on DMA printer/plotter 1
23	19	Versatec on DMA printer/plotter 2
24	20	Tektronix display
25	21	Magnetic tape 1 unit 2
26	22	Synchronous modem 5
27	23	Synchronous modem 6
30	24	Synchronous modem 3
31	25	Synchronous modem 4
32	26	Magnetic tape 2 unit 0
33	27	Magnetic tape 1 unit 3
34	28	Magnetic tape 2 unit 1
35	29	Card Punch 3
36	30	CDC link
37	31	Not used
40	32	Magnetic tape 1 unit 0
41	33	Magnetic tape 1 unit 1
42-47	34-39	Terminals 3-8
50	40	Card punch 1
51	41	Card punch 2
52-57	42-47	Terminals 19-24
60-67	48-55	Terminals 9-16
70-77	56-63	Terminals 25-32

Octal logical device number	Process control devices and connect devices
400-437	CAMAC interrupts or special process interface
440	Direct task level 6
441	Direct task level 7
442	Direct task level 8
443	Direct task level 9
450-467	CONNECT devices
470	ND 23 - programmed clock

Octal logical device number	Disk, spooling, and other system devices
500	Internal device for error message RT program
501	Semaphore for segment transfer
502	Disk 10Mb 1 datafield
503	RT loader command lock
504	General lock for file system
505	User-file-buffer lock
506	Object-file-buffer lock
507	RT-open-file-table lock
511	Disk 10Mb 1, unit 0, R-bit-file-buffer lock
512	Disk 10Mb 1, unit 0, F-bit-file-buffer lock
513	Disk 10Mb 1, unit 0, R-directory lock
514	Disk 10Mb 1, unit 0, F-directory lock
515	DF1, file-transfer for RT lock for disk 1, disk 2, disk3 and disk 4
516	DF2, for open-file monitor call from RT program datafield
517	RTFIL semaphore
520	NOTIS-IR semaphore 2
521	Device buffer allocation lock
522	Disk 10Mb 1, unit 1, R-directory lock
523	Disk 10Mb 1, unit 1, F-directory lock
524	Disk 10Mb 1, unit 1, R-bit-file-buffer lock
525	Disk 10Mb 1, unit 1, F-bit-file-buffer lock
526	DF3, transfer lock for magnetic tape 1
527	Spooling queue semaphore
530	Accounting semaphore
531	CDC link monitor call datafield
532	Spooling device 4, queue semaphore
533	Spooling device 4, queue I/O semaphore
534	Spooling device 5, queue semaphore
535	Spooling device 5, queue I/O semaphore
536	Spooling device 6, queue semaphore
537	Spooling device 6, queue I/O semaphore
540	Internal Device Remote Batch IBM
541	Internal Device Remote Batch UNIVAC
542	Internal Device Remote Batch Honeywell Bull
543	Internal Device Remote Batch CDC

Continued

Octal logical device number	Disk, spooling, and other system devices continued
544	Big disk 3, Unit 0 directory lock
545	Big disk 3, Unit 0 bit file buffer lock
546	Unit 1
550	(547) Unit 1
551	(550) Unit 2
552	(551) Unit 2
554	Disk 10Mb 1, unit 3, R-bit-file-buffer lock (552) Unit 3
555	Disk 10Mb 1, unit 3, F-bit-file-buffer lock (553) Unit 3
556	Disk 10Mb 1, unit 3, R-directory lock
557	Disk 10Mb 1, unit 3, F-directory lock
560	Magnetic tape 1, datafield
561	All magnetic tapes, directory lock
562	Spooling device 11, queue semaphore
563	Magtape 2, unit 2, I/O datafield
564	Magtape 2, unit 3, I/O datafield
565	Big disk 3, datafield
567	CDC link data field
570	Disk 10Mb 1, unit 2, R-directory lock
571	Disk 10Mb 1, unit 2, F-directory lock
572	Disk 10Mb 1, unit 2, R-bit-file-buffer lock 563 Magnetic tape 2, unit 2, I/O datafield
573	Disk 10Mb 1, unit 2, F-bit-file-buffer lock
574	Monitor call datafield for cassette
575	Cassette data field 564 Magnetic tape 2, unit 3 I/O datafield
576	DF5, monitor call data field for Versatec 1
577	Versatec data field

Octal logical device number	Floppy disks, printers, terminals, and other system devices
1000	Floppy disk 1, unit 0, I/O datafield
1001	Floppy disk 1, unit 1, I/O datafield
1002	Floppy disk 1, unit 2, I/O datafield
1003	Floppy disk 2, unit 0, I/O datafield
1004	Floppy disk 2, unit 1, I/O datafield
1005	Floppy disk 2, unit 2, I/O datafield
1006	Hasp DMA 1, I/O datafield
1007	Hasp DMA 2, I/O datafield
1010	Hasp DMA 3, I/O datafield
1011	Hasp DMA 4, I/O datafield
1012	Hasp DMA 5, I/O datafield
1013	Hasp DMA 6, I/O datafield
1014	Line printer 3, I/O datafield
1015	Line printer 4, I/O datafield
1040-1077	Terminals 33-64

Octal logical device number	Various system devices
1100	Big disk/big cartridge disk 1, datafield
1101	Big disk/big cartridge disk 1, unit 0, directory lock
1102	Big disk/big cartridge disk 1, unit 0, bit-file-buffer lock
1103	NORD-50 datafield
1104	Disk 10Mb 2, datafield
1105	Disk 10Mb 2, unit 0, R-directory lock
1106	Disk 10Mb 2, unit 0, F-directory lock
1107	Disk 10Mb 2, unit 0, R-bit-file-buffer lock
1110	Disk 10Mb 2, unit 0, F-bit-file-buffer lock
1111	Magnetic tape 2 datafield
1112	Big disk 4, unit 0, directory lock
1113	Floppy disk 1, unit 3, I/O datafield
1114	Big disk 4, unit 0, bitfile-buffer lock
1115	Floppy disk 2, unit 3, I/O datafield
1116	Dr 7, transfer lock for magnetic tape 2
1117	Big disk/big cartridge disk 1, unit 1, directory lock
1120	Big disk/big cartridge disk 1, unit 1, bit-file-buffer lock
1121	Big disk/big cartridge disk 1, unit 2, directory lock
1122	Big disk/big cartridge disk 1, unit 2, bit-file-buffer lock
1123	Big disk/big cartridge disk 1, unit 3, directory lock
1124	Big disk/big cartridge disk 1, unit 3, bit-file-buffer lock
1125	Versatec controller 2
1126	Monitor call datafield for Versatec controller 2
1127	DF 39, magnetic tape 3 monitor call datafield
1130	Disk 10Mb 2, unit 1, R-directory lock
1131	Disk 10Mb 2, unit 1, F-directory lock
1132	Disk 10Mb 2, unit 1, R-bit-file lock
1133	Disk 10Mb 2, unit 1, F-bit-file lock
1134	Floppy disk 1, unit 3, directory table lock
1135	Floppy disk 1, unit 3, bit-file-buffer lock
1136	Spooling device 1, queue semaphore
1137	Spooling device 1, queue I/O semaphore
1140	Spooling device 2, queue semaphore
1141	Spooling device 2, queue I/O semaphore
1142	Spooling system general semaphore
1143	Spooling system wait for used pages semaphore
1144	Spooling system wait for free pages semaphore
1145	Floppy disk 1, datafield
1146	Monitor call datafield for floppy disk 1
1147	Floppy disk 2, unit 3, directory table lock
1150	Floppy disk 1, unit 0, directory table lock
1151	Floppy disk 1, unit 0, bit-file-buffer lock

Continued

Octal Logical Device Number	Various system devices
1152	Floppy disk 1, unit 1, directory table lock
1153	Floppy disk 1, unit 1, bit-file-buffer lock
1154	Floppy disk 1, unit 2, directory table lock
1155	Floppy disk 1, unit 2, bit-file-buffer lock
1156	Floppy disk 2, datafield
1157	Monitor call datafield for floppy disk 2
1160	Floppy disk 2, unit 3, bit-file-buffer lock
1161	Floppy disk 2, unit 0, directory table lock
1162	Floppy disk 2, unit 0, bit-file-buffer lock
1163	Floppy disk 2, unit 1, directory table lock
1164	Floppy disk 2, unit 1, bit-file-buffer lock
1165	Floppy disk 2, unit 2, directory table lock
1166	Floppy disk 2, unit 2, bit-file-buffer lock
1167	Line printer 1, datafield
1170	Monitor call datafield for line printer 1
1171	Big disk 4, unit 2, directory lock
1172	Big disk 4, unit 2, bitfile buffer lock
1173	Semaphore for spooling device 3
1174	Semaphore for spooling device 3
1175	Line printer 2, datafield
1176	Monitor call datafield lock for line printer 2
1177	Spooling semaphore, for id data buffer lock
1200	ND TPS system semaphore
1201	DMAC command lock
1202	RT-PROGRAM-LOG semaphore
1203	Histogram commands lock
1204	SINTRAN-SERVICE-PROGRAM command lock
1205	Mail system lock
1206	Terminal 1, datafield
1207	Big disk/big cartridge disk 2, datafield
1210	Internal device 1, datafield
1211	Monitor call datafield for internal device 1
1212	Internal device 2, datafield
1213	Monitor call datafield for internal device 2
1214	Internal device 3, datafield
1215	Monitor call datafield for internal device 3
1216	Internal device 4, datafield
1217	Monitor call datafield for internal device 4
1220	Internal device 5, datafield
1221	Monitor call datafield for internal device 5
1222	Accounting semaphore
1223	NOTIS-IR semaphore
1224	Winchester disk, datafield or STC magtape controller 4
1225	Winchester disk 1, unit 0, directory table lock STC magtape 4, unit 0, I/O datafield
1226	Winchester disk 1, unit 0, bit-file-buffer lock STC magtape 4, unit 1, I/O datafield
1227	Winchester disk 1, unit 1, directory table lock STC magtape 4, unit 2,I/O datafield

Continued

Octal logical device number	Various system devices
1230	Winchester disk, unit 1, bit-file-buffer lock STC magtape 4, unit 3, I/O datafield
1231	Winchester disk 2, datafield or STC magtape controller 3
1232	Winchester disk 2, Unit 0, directory table lock
1233	Winchester disk 2, unit 0, bit-file-buffer lock STC magtape 3, unit 1, I/O datafield
1234	Winchester disk 2, unit 1, directory table lock STC magtape 3, unit 2, I/O datafield
1235	Winchester disk 2, unit 1, bit-file-buffer lock STC magtape 3, unit 3, I/O datafield
1236	Batch process 1, datafield
1237	Batch process 1, internal device
1240	Batch process 2, datafield
1241	Batch process 2, internal device
1242	Batch process 3, datafield
1243	Batch process 3, internal device
1244	Batch process 4, datafield
1245	Batch process 4, internal device
1246	Batch process 5, datafield
1247	Batch process 5, internal device
1250	Batch process 6, datafield
1251	Batch process 6, internal device
1252	Batch process 7, datafield
1253	Batch process 7, internal device
1254	Batch process 8, datafield
1255	Batch process 8, internal device
1256	Batch process 9, datafield
1257	Batch process 9, internal device
1260	Batch process 10, datafield
1261	Batch process 10, internal device
1262	Spooling device 7, queue semaphore
1263	Spooling device 7, queue I/O semaphore
1264	Spooling device 8, queue semaphore
1265	Spooling device 8, queue I/O semaphore
1266	Spooling device 9, queue semaphore
1267	Spooling device 9, queue I/O semaphore
1270	Spooling device 10, queue semaphore
1271	Spooling device 10, queue I/O semaphore
1272	Monitor call datafield for internal device 1
1273	Monitor call datafield for internal device 2
1274	Monitor call datafield for internal device 3
1275	Monitor call datafield for internal device 4
1276	Monitor call datafield for internal device 5
1277	DF 40, magnetic tape 4, monitor call datafield
1300	Big disk 4, unit 3, directory lock
1301	Big disk 4, unit 3, bit file buffer lock
1302	Device buffer lock
1303	Hasp DMA 1, datafield

Continued

Octal logical device number	Various system devices
1304	Hasp DMA 1, datafield 1300 bigdisk 4, unit 3, directory lock
1305	Monitor call datafield for Hasp DMA 1 1301 big disk 4, unit 3, bit-file-buffer lock
1306	Monitor call datafield for Hasp DMA 1 1302 not used
1307	Hasp DMA 2, datafield
1310	Hasp DMA 2, datafield
1311	Monitor call datafield for Hasp DMA 2
1312	Monitor call datafield for Hasp DMA 2
1313	Hasp DMA 3, datafield
1314	Hasp DMA 3, datafield
1315	Monitor call datafield for Hasp DMA 3
1316	Monitor call datafield for Hasp DMA 3
1317	Hasp DMA 4, datafield
1320	Hasp DMA 4, datafield
1321	Monitor call datafield for Hasp DMA 4
1322	Monitor call datafield for Hasp DMA 4
1323	Hasp DMA 5, datafield
1324	Hasp DMA 5, datafield
1325	Monitor call datafield for Hasp DMA 5
1326	Monitor call datafield for Hasp DMA 5
1327	Hasp DMA 6, datafield
1330	Hasp DMA 6, datafield
1331	Monitor call datafield for Hasp DMA 6
1332	Monitor call datafield for Hasp DMA 6
1333	Big disk/big cartridge disk 2, unit 0, directory table lock
1334	Big disk/big cartridge disk 2, unit 0, bit-file-buffer lock
1335	Big disk/big cartridge disk 2, unit 1, directory table lock
1336	Big disk/big cartridge disk 2, unit 1, bit-file-buffer lock
1337	Big disk/big cartridge disk 2, unit 2, directory table lock
1340	Big disk/big cartridge disk 2, unit 2, bit-file-buffer lock
1341	Big disk/big cartridge disk 2, unit 3, directory table lock
1342	Big disk/big cartridge disk 2, unit 3, bit-file-buffer lock
1343	Line printer 3, datafield
1344	Monitor call datafield for line printer 3
1345	Line printer 4, datafield
1346	Monitor call datafield for line printer 4
1347	Spooling device 11, queue I/O semaphore
1350	Spooling device 12, queue semaphore
1351	Spooling device 12, queue, I/O semaphore
1352	RT-PROGRAM-LOG command lock
1360	HDLC DMA, link 1, input; synchronous modem 1 for HDLC interface input/output

Continued

Octal Logical Device Number	Various system devices
1361	HDLC DMA, link 1, output
1362	HDLC DMA, link 2, input; synchronous modem 2 for HDLC interface input/output
1364	HDLC DMA, link 3, input; synchronous modem 3 for HDLC interface input/output
1366	HDLC DMA, link 4, input; synchronous modem 4 for HDLC interface input/output
1373	HDLC DMA, link 6, output; synchronous modem 6 for HDLC interface output.
1374	X21 logical number 1
1375	X21 logical number 2
1376	X21 logical number 3
1377	X21 logical number 4
1400-1537	Terminal access device (TAD) 1-96
1600-1677	DMA device buffer, header locks for header numbers 0-77
1722	Spooling device 13, queue semaphore
1723	Spooling device 13, queue I/O semaphore
1724	Spooling device 14, queue semaphore
1725	Spooling device 14, queue I/O semaphore
1726	Spooling device 15, queue semaphore
1727	Spooling device 15, queue I/O semaphore
2000-2077	Terminal number 65-127

APPENDIX C: FILE SYSTEM ENTRIES

Each directory, user, and file has a description in the file system. This appendix shows how their contents are organized. The followin table describes the directory. Bits are numbered from high to low, eg., from 15 to 0.

BYTE	DIRECTORY INFORMATION
0:1	Various, bit 15: directory is entered. bit 14: main directory. bit 13: default directory. bit 11: directory in special use, eg., @TEST-DIRECTORY. bit 10-0: Number of opened files on the directory.
2:3	Unit number in bit 15-12, device number in bit 11-0.
4:5	Subunit number i bit 15-9, name index i bit 7-0.
6:7	Logical device number for the directory semaphore.
8:9	Logical device number for the disk or floppy disk.
10:11	Number of users logged in with this as main directory.
12:13	Number of users logged in with this as default directory.
14:29	Directory name.
32:35	Object file pointer.
36:39	User file pointer.
40:43	Pointer to the bit file.
44:47	Number of unreserved pages in the directory.

The following table describes the user entry. Bits are numbered from high to low, eg., from 15 to 0.

BYTE	USER INFORMATION
0:1	Various, bit 15: set if the entry is used. bit 8: always set. bit 7-0: enter count.
2:17	User name.
18:19	Encrypted password.
20:23	Date created.
24:27	Last date logged in.
28:31	Number of reserved pages.
32:35	Number of pages in use.
36:37	User index.
38:39	Not used.
40:41	Default file access.
42:43	Address of the previous user entry on the directory.
44:45	Address of next user entry on the directory.
46:47	Not used.
48:63	Friend table. Two bytes for each of the 8 possible friends. The 16 bits for each friend are used as shown below. Bit 15: set if friend exists. Bit 12: set if friend has directory access. Bit 11: set if friend has common access. Bit 10: set if friend has append access. Bit 9: set if friend has write access. Bit 8: set if friend has read access. Bit 7-0: user index of friend.

The following table describes the object entry. There is on object entry for each version of a file. Bits are numbered from high to low, eg., from 15 to 0.

BYTE	OBJECT INFORMATION
0:1	Various, bit 15: set if object entry in use. bit 14: set if currently opened for write. bit 13: set if file is reserved. bit 12: set if the file is modified. bit 11: bit 10-0: terminal number of last user opening the file. 0 if opened by RT program.
2:17	File name.
18:21	File type.
22:23	Address of object entry of the next file version.
24:25	Address of object entry of the previous file version.
26:27	File access, bit 14-9: public access bit 9-4: friend access. bit 4-0: own access.
28:29	Attributes, bit 5: set if allocated file. bit 4: set if contiguous file. bit 3: set if indexed file. bit 2: set if spooling fil bit 1: set if peripheral file. bit 0: set if terminal file.
30:31	Device number.
32:33	User index in main directory of reserving user.
34:35	Object index of this object entry.
36:37	Current open count.
38:39	Total open count.
40:43	Date created.
44:47	Last opened for read.
48:51	Last opened for write.
52:55	Pages in file.
56:59	Bytes in file.
60:63	File pointer. Bit 35: set if index. Bit 34: set if subindex. Bit 33-0: set if file pointer.

APPENDIX D: RT PROGRAM DESCRIPTIONS

This appendix shows the contents of the RT descriptions. Bits are numbered from high to low, eg., from 15 to 0.

BYTE	RT PROGRAM INFORMATION
0:1	Time queue link. The last queue element contains -1.
2	Status. bit 7, set if program waits for input or output. bit 6, set if program should restart when terminated. bit 5, set if WaitForRestart is executed. bit 4, set if periodic program. bit 3, set if absolute startup time is specified. bit 2, set if TimeOut has been executed. bit 1:0, Initial execution ring.
3	Priority.
4:7	Startup time for scheduled program in basic time units.
8:11	Time interval for periodic execution in basic time unit.
12:13	Start address of the RT program.
14	First initial segment.
15	Second initial segment.
16:17	Link for wait and execution queue.
18	First active segment.
19	Second active segment.
20:21	Active page tables and ring bit 15, set for background processes, ie., BAK01, BAK02, etc. bit 14, set if startup inhibited, ie., after DisableRTStart. bit 13, set if program is in the swapping queue. bit 12:11, initial page table. bit 10:9, normal page table. bit 8:7, alternative page table. bit 6:3, interrupt level. bit 2, not used. bit 1:0, actual execution ring.
22:23	Head of reservation queue.
24:25	Reentrant segment number.
26:27	Window with file transfer page number.
28:29	Address of the RT descrition part in the paging off area.
30:31	Saved P register.
32:33	Saved X register.
34:35	Saved T register.
36:37	Saved A register.
38:39	Saved D register.
40:41	Saved L register.
42:43	Saved status register.
44:45	Saved B register.
46:53	Reentrant segment bitmap. One bit for each of the 128 pages.

APPENDIX E: SEGMENT DESCRIPTORS

This appendix shows contents of the segment descriptors. Bits are numbered from high to low, eg., from 15 to 0.

BYTE	SEGMENT INFORMATION
0:1	Segment queue link.
2:3	Page queue header.
4:5	Segment start and size. bit 15, not used. bit 14:8, segment size in number of pages. bit 7:6, page table number. bit 5:0, first logical page.
6:7	Position on segment file. bit 15:14, segment file, ie., SEGFIL0, SEGFIL1, etc. bit 13:0, start page within the segment file.
8:9	Flags. bit 15, write permitted. bit 14, read permitted. bit 13, instruction fetching permitted. bit 12, segment is written to. bit 11, not used. bit 10:9, protection ring. bit 8:7, not used. bit 6, reentrant subsystem. bit 5, segment protect flag. bit 4, system segment. bit 3, inhibit use of segment. bit 2, segment fixed in physical memory. bit 1, demand segment. bit 0, sufficient physical memory.

APPENDIX F: ASCII TABLE

The following numbers represent characters under SINTRAN III.

CHAR	Byte Left	Byte Right		CHAR	Byte Left	Byte Right	Decimal
NUL	000000	000000	0	SPACE	020000	000040	32
SOH	000400	000001	1	!	020400	000041	33
STX	001000	000002	2	"	021000	000042	34
ETX	001400	000003	3	#	021400	000043	35
EOT	002000	000004	4	$	022000	000044	36
ENQ	002400	000005	5	%	022400	000045	37
ACK	003000	000006	6	&	023000	000046	38
BEL	003400	000007	7	'	023400	000047	39
BS	004000	000010	8	(	024000	000050	40
HT	004400	000011	9	)	024400	000051	41
LF	005000	000012	10	*	025000	000052	42
VT	005400	000013	11	+	025400	000053	43
FF	006000	000014	12	,	026000	000054	44
CR	006400	000015	13	-	026400	000055	45
SO	007000	000016	14	.	027000	000056	46
SI	007400	000017	15	/	027400	000057	47
DLE	010000	000020	16	0	030000	000060	48
DC1	010400	000021	17	1	030400	000061	49
DC2	011000	000022	18	2	031000	000062	50
DC3	011400	000023	19	3	031400	000063	51
DC4	012000	000024	20	4	032000	000064	52
NAK	012400	000025	21	5	032400	000065	53
SYN	013000	000026	22	6	033000	000066	54
ETB	013400	000027	23	7	033400	000067	55
CAN	014000	000030	24	8	034000	000070	56
EM	014400	000031	25	9	034400	000071	57
SUB	015000	000032	26	:	035000	000072	58
ESC	015400	000033	27	;	035400	000073	59
FS	016000	000034	28	<	036000	000074	60
GS	016400	000035	29	=	036400	000075	61
RS	017000	000036	30	>	037000	000076	62
US	017400	000037	31	?	037400	000077	63

CHAR	Byte position: Left	Right		CHAR	Byte position: Left	Right	Decimal
@	040000	000100	64	`	060000	000140	96
A	040400	000101	65	a	060400	000141	97
B	041000	000102	66	b	061000	000142	98
C	041400	000103	67	c	061400	000143	99
D	042000	000104	68	d	062000	000144	100
E	042400	000105	69	e	062400	000145	101
F	043000	000106	70	f	063000	000146	102
G	043400	000107	71	g	063400	000147	103
H	044000	000110	72	h	064000	000150	104
I	044400	000111	73	i	064400	000151	105
J	045000	000112	74	j	065000	000152	106
K	045400	000113	75	k	065400	000153	107
L	046000	000114	76	l	066000	000154	108
M	046400	000115	77	m	066400	000155	109
N	047000	000116	78	n	067000	000156	110
O	047400	000117	79	o	067400	000157	111
P	050000	000120	80	p	070000	000160	112
Q	050400	000121	81	q	070400	000161	113
R	051000	000122	82	r	071000	000162	114
S	051400	000123	83	s	071400	000163	115
T	052000	000124	84	t	072000	000164	116
U	052400	000125	85	u	072400	000165	117
V	053000	000126	86	v	073000	000166	118
W	053400	000127	87	w	073400	000167	119
X	054000	000130	88	x	074000	000170	120
Y	054400	000131	89	y	074400	000171	121
Z	055000	000132	90	z	075000	000172	122
[	055400	000133	91	{	075400	000173	123
\	056000	000134	92	|	076000	000174	124
]	056400	000135	93	}	076400	000175	125
^	057000	000136	94	~	077000	000176	126
-	057400	000137	95	DEL	077400	000177	127

APPENDIX G: PERIPHERAL FILE NAMES

This is SINTRAN III's standard peripheral file names.

ND Number	Description	Peripheral File Name	Notes
202, 204, - 228	Terminals	TERMINAL	Refers to own terminal in background. Terminals can also be PRINTER.
252, 254	Intercomputer link	CHANNEL-0 CHANNEL-1 ... CHANNEL-15 L1-CH-0 ... L1-CH-15 L2-CH-0 ... L2-CH-15 ...	If only one link If two or more links Links with background programs are usually not included.
301, 302	Paper Tape Reader	TAPE-READER	Suffix "-1", "-2", etc. is used if more than one device.
303	Page Tape Punch	TAPE-PUNCH	See ND301.
305, etc.	Floppy Disk	FLOPPY-1 FLOPPY-2 ...	These names only work with one controller
400, etc.	Card Reader	CARD-READER	See ND301
430, 431, etc.	Line Printer	LINE-PRINTER	See ND301
414, 415, 417	Matrix Printer	PRINTER See	ND301
420	Card Punch	CARD-PUNCH	See ND301
515, etc.	Magnetic Tape	MAG-TAPE-0 MAG-TAPE-1 ...	These names only work with one controller
603, 604, 605, 606	Versatec Printer Plotter	LINE-PRINTER-1 LINE-PRINTER-2 ... VERSATEC-1 VERSATEC-2 ...	If no other line printer on the system If another line printer on the system

APPENDIX H: TERMINAL TYPES

SINTRAN III handles the following terminals. Other terminals are added on request. The column marked "No." is the model number. A mark in the VDU column means that the terminal has a Video Display Unit. BS means backspace. FF means form feed.

Model Name	No.	VDU	BS	FF	Comment
DUMMY	0				No terminal type set
VISTAR-OLD	1	x		x	
TELETYPE-ASR-33	2				
TANDBERG-TDV2115	3	x	x		
INFOTON-200-1	4	x	x	x	
INFOTON-400	5	x	x	x	
DEC-VT100	6	x	x		80-col. mode
TANDBERG-TDV2000	7	x	x		
BEEHIVE-100	8	x			
ND-NCT	9	x	x	x	
HAZELTINE-1520	10	x	x		
DEC-LA36	11		x		Decwriter-II
VISTAR-GTX	12	x	x		
DEC-VT52	29	x	x		
TEC-501/502	30	x	x		
DACOLL-242	31	x	x		
NEWBURY-7000/3	32	x	x		
TELEVIDEO-912/920	33	x	x		
VISUAL-200	34	x	x		
LEAR-SIEGLER-ADM-3A	35	x	x		
TANDBERG-TDV2215-EXTENDED	36	x	x		
VOLKER-CRAIG-VC404	37	x	x		
VOLKER-CRAIG-VC410	38	x	x	x	
VOLKER-CRAIG-VC414	39	x	x		
HEWLETT-PACKARD-2621A	40	x	x		
DATA-MEDIA-ELITE-3045	41	x	x		
BEEHIVE-MINIBEE	42	x			
PERICOM-6800	43	x	x	x	80-col.mode
LEAR-SIEGLER-ADM-31	44	x	x		
BEEHIVE-DM5A	45	x	x		
FACIT-4420	46	x	x	x	VT52-mode
ADDS-VIEWPOINT	47	x	x	x	
HAZELTINE-EXECUTIVE-80	48	x	x		
AMPEX-DIALOGUE-80	49	x	x		
VOLKER-CRAIG-VC4404	50	x	x		ADM-3A
DATA-MEDIA-ELITE-1520/1521	51	x	x	x	
TANDBERG-TDV2215-SDS-V2	52	x	x		
TANDBERG-TDV2200/9-ND NOTIS	53	x	x		
TANDBERG-TDV2220	54	x	x		
TANDBERG-TDV2200/9-ND-NET	55	x	x		
FACIT-4420-ND NOTIS	57	x	x	x	

Continued

Model Name	No.		VDU	BS	FF	Comment
NOKIA-VDU210	58		x	x		
LEAR-SIEGLER-ADM-42	66		x	x		
LEAR-SIEGLER-ADM-32	70		x	x		
GENERAL-TERMINAL-CORP.-100/10	73	1	x	x	x	
TEKTRONIX-4105	78		x	x		
TANDBERG-TDV2200/9-V2-ND-NOTI	83	S	x	x		

The terminal type number is a 16-bit integer. Bit 0 to 7 contains the model number above. The other bits are used as follows:

Bit	Meaning
15	Reserved.
14	Set to one if the terminal is a VDU, ie., not a hard copy terminal.
13	Set to one if the terminal handles the ASCII backspace character properly.
12	Set to one if the ASCII form feed character gives new page or clears the screen.
11	Set to one if the VDU has cursor positioning, either directly or by use of cursor arrows.
10	Set if the terminal uses the ASCII escape character in input sequences.
7-0	Terminal model number.

You may use either the model number or the complete terminal type number to set the terminal type.

APPENDIX I: HARDWARE STATUS VALUES

This is the status values returned by the monitor call DeviceFunction with function codes 20 and 24. The specified condition is true if the bit is set in the 16-bit status word.

Hardware status values for the following devices are described:

- Status word for Tandberg, Pertec and STC magnetic tape units
- Status word for Hewlett-Packard magnetic tape units
- Status word for Philips cassette
- Status word for Versatec line printer/plotter
- Status word for old (PIO) floppy disk
- Status word for new (DMA) floppy disk
- Status word for ECC disk controllers
- Status word for big disks 33/66 MB
- Status word for small disk 10 MB
- Status word for 45 MB Micropolis and 21 MB Finch disks

The command @DEVICE-FUNCTION is similar to the monitor call.

Status word for Tandberg, Pertec, and STC magnetic tape units	
Bit 0	Tape on line.
1	Write enable ring present.
2	Tape standing onload point.
3	CRC error/fatal error.
4	Set if any of bits 5, 6, 7, 8, 9, 11 or 12 are set.
5	Control or modus word error. Trying to write on protected tape, trying to reverse tape at load point, tape unit not on-line etc. Action is inhibited.
6	Bad data block. An error is detected.
7	End of file is detected.
8	The search character is detected.
9	End of tape is detected. Resetting this bit depends on the model. Tandberg, STC: the bit remains set if carrying out a function after EOT. Pertec: the bit is cleared if carrying out a function after EOT.
10	Word counter is not zero.
11	DMA error.
12	Overflow in read.
13	Tape busy or formatter busy.
14	LRC error/software error.
15	Interrupt when formatter is ready.

Status word for Hewlett-Packard magnetic tape units	
Bit 0	Ready interrupt enabled. It is cleared by the interrupt.
1	Error interrupt enabled. It is cleared by the interrupt.
2	Device active.
3	Device ready for transfer.
4	Set if any of bits 6, 9, 10, 11 or 12 are set or if a reverse command is given with tape at load point.
5	Write enable ring present.
6	LRC error.
7	EOF detected.
8	Load point. The unit remains in this state also after the first forward command after load point is detected.
9	EOT detected.
10	Parity error.
11	DMA error.
12	Overflow in read.
13	Density select: 1 = 800 BPI.
14	Magnetic tape unit ready; selected, online and not rewinding
15	Bit 15 is loaded by the previous control word.

Status word for Philips cassette	
Bit 0	Ready for transfer, interrupt is enabled.
1	Error interrupt enabled.
2	Device is active.
3	Device is ready for transfer.
4	Set if any of bits 0, 1 or 5 are set.
5	Write enable.
6	Cassette side indicator (A = 1, B = 0).
7	Bit clock.
8	Read failure
9	Synchronisation failure
10	Not used.
11	Not used.
12	Drive failure
13	Write protection violation.
14	Beginning or end of tape.
15	Not used.

Status word for Versatec line printer/plotter	
Bit 0	Ready for transfer, interrupt enabled.
1	Error interrupt enabled.
2	Device active.
3	Device ready for transfer.
4	Set if bit 6 or 7 is set.
5	Not used.
6	No paper.
7	Plotter not on-line.
8	Not used.
9	Not used.
10	Not used.
11	Not used.
12	Not used.
13	Plotter ready.
14	Not used.
15	Not used.

Status word for old (PIO) floppy disk	
Bit 0	Interrupt enabled.
1	Not used.
2	Device busy.
3	Device ready for transfer.
4	Set if any of bits 5, 8, 11, 12 or 14 are set.
5	Deleted record detected.
6	Read/write completed.
7	Seek completed.
8	Drive not ready.
9	Write protected.
10	Not used.
11	Address mismatch.
12	CRC error.
13	Not used.
14	Data overrun.
15	Must be 0 for this type of floppy disk.

Status word for new (DMA) floppy disk	
Bit 0	Rft-interrupt enabled.
1	Not used.
2	Device active.
3	Device ready for transfer.
4	OR of errors.
5	Deleted record.
6	Retry on controller.
7-8	Not used.
9-14	Error code from controller, see below.
15	Should be 1 for this type of floppy disk.

Number	Description of the octal error number in bit 9-14
0	Ok.
5	CRC-error.
6	Sector not found.
7	Track not found.
10	Format not found
11	Diskette defect, impossible to format.
12	Format mismatch.
13	Illegal format.
14	Single-sided diskette inserted.
15	Double-sided diskette inserted.
16	Write-protected diskette.
17	Seleted record.
20	Drive not ready.
21	Controller busy on start.
22	Lost data, over- or underrun.
23	Track zero not detected.
24	Vco-frequence out of range.
25	Microprogram out of range.
26	Timeout.
27	Undefined error.
30	Track out of range.
31	Ram error.
32	Compare error.
33	Internal DMA-error.
40	ND-100 bus error during command fetch.
41	ND-100 bus error during status transfer.
42	ND-100 bus error during data transfer.
43	Illegal command.
44	Wordcount not zero.
50	No bootstrap found on diskette.
51	Wrong bootstrap, too old version of floppy-monitor.
70	Prom checksum error. (selftest error)
71	Ram error. (selftest error)
72	Ctc error. (selftest error)
73	Dmactrl error. (selftest error)
74	Vco error. (selftest error)
75	Floppy control error. (selftest error)

Status word for ECC disk controllers 37/75/288 Mb Phoenix disks	
Bit 0	Controller not active, interrupt enabled.
1	Error interrupt enabled.
2	Controller active.
3	Controller finished with a device operation.
4	Inclusive OR of errors (Bit 5 - 13).
5	Illegal load, ie., load while status bit 2 is true, or load of block address while the unit is not on cylinder.
6	Timeout.
7	Hardware error, disk fault, missing read clocks, and missing servo clocks.
8	Address mismatch.
9	Parity error.
10	Compare error.
11	DMA channel error.
12	Abnormal completion.
13	Disk unit not ready.
14	On cylinder.
15	Extended cylinder address.

Status word for big disks 33/66 MB	
Bit 0	Controller not active.
1	Error interrupt enabled.
2	Controller active.
3	Finished with device operation.
4	Inclusive OR of errors (5-13).
5	Write protect violation.
6	Time out.
7	Hardware error.
8	Address mismatch.
9	Parity error.
10	Compare error.
11	Dma channel error.
12	Abnormal completion.
13	Disk unit not ready.
14	On cylinder.
15	Extended cylinder-address.

Status word for small disk 10 MB	
Bit 0	Ready for transfer, interrupt enabled.
1	Error interrupt enabled.
2	Device active.
3	Device ready for transfer.
4	Inclusive OR of errors (bit 5-11).
5	Write protect violation.
6	Timeout.
7	Hardware error.
8	Address mismatch.
9	Parity error.
10	Compare error.
11	Dma channel error.
12	Transfer complete.
13	Transfer on.
14	On cylinder.
15	Loaded by previous control-word.

Status word for 45 MB Micropolis and 21 MB Finch disks	
Bit 0	Controller not active interrupt enabled.
1	Error interrupt enabled.
2	Controller active.
3	Controller finished with a device operation.
4	Inclusive OR of errors (bits 5-11).
5	Finch: 0 (not used), Micropolis: trying to read or write while performing rtz.
6	Timeout.
7	Disk fault or missing clocks.
8	Address mismatch.
9	Crc error.
10	Compare error.
11	Fifo over/under-run or DMA channel error.
12	Finch: Serious error (or of status bits 6, 7 and 8). Micropolis: Track 0.
13	Finch: Read or write gate active. Micropolis: Always 1.
14	On cylinder.
15	0, used to distinguish from 10 Mb controller.

APPENDIX J: GLOSSARY

This glossary explains the SINTRAN III terms in the manual. Some general terms are included. The explanations are intended for programmers.

Access code. A number describing how a file is accessed. For example, 0 means it is written to sequentially.

ADA. A high level programming language. Available on ND-500.

Address area. The maximum amount of memory a program may use. The address area on ND-100 is 128 Kbytes for one-bank programs and 256 Kbytes for two-bank programs. ND-500 has no practical restrictions on the address area.

Allocated file. A contiguous file created at a particular page address on the disk. See CreateFile.

ASCII. A code where each character is assigned a number. For example, "A" is represented by 65, "B" is represented by 66, etc. The ASCII code include control characters. Appendix F contains an ASCII table.

ASSEMBLY-500. The ND-500 assembly language.

B. Number followed by a "B", eg., 400B, means octal numbers.

Background program. A program that is loaded with the BRF-LINKER, NRL, or the LOAD command in a compiler. Programs are either background programs or RT programs.

BACKUP-SYSTEM. A SINTRAN III system used to copy several files efficiently, eg., to make security copies. See the manual SINTRAN III UTILITIES (ND-60.151).

BAK01. RT programs called BAK01, BAK02, BAK77, BK100, etc., control each terminal. The RT programs are a part of SINTRAN III.

Basic time unit. One 1/50 of a second. The length can be changed by the system supervisor.

Batch. A way of executing command and background programs independently of any terminal. Input is read from a mass storage file instead of a teminal.

BK100. RT programs called BAK01, BAK02, BAK77, BK100, etc., control each terminal. The RT programs are a part of SINTRAN III.

Block. A file is logically divided into blocks. A block is normally 512 bytes. Some monitor calls operate on blocks instead of bytes. The first block of a file is number 0.

BPI. Bits stored per inch.

BRF-LINKER. A SINTRAN III subsystem to load and link programs.

Buffer. A buffer is a temporary storage area.

Byte. A byte is always 8 bit in ND computer systems.

Camac. A common interface to scientific instruments.

Capability. A domain keeps a description of each logical segment in use. The description is called a capability. It contains information about access rights, location in physical segments, and sharing with other processes. See the manual ND-500 LOADER/MONITOR (ND-60.136).

Character device. A device which receives and sends one byte at a time, eg., a terminal, a printer, or a magnetic tape station.

COBOL. A high level programming language. Available on ND-100 and ND-500.

Compiler. SINTRAN III subsystems to translate high level language programs to code that can be loaded and then executed. The compilers for ND-100 have names like FORTRAN-100, PASCAL-100, etc. The ND-500 compilers have names like FORTRAN-500 and COBOL-500.

Console. A terminal which prints on paper instead of a video display unit. Most computers have a console used for system supervising. The console is normally also the error device.

Contiguous file. A file with fixed length. Its pages are placed contiguously on the disk. The access to contiguous files is faster than to ordinary files. See CreateFile.

Control character. A control character, eg., CTRL A, is given by holding down the CTRL key on the terminal keyboard while pressing A. Control characters are mostly used to perform certain functions, eg., delete a character. Control characters have ASCII codes in the range 0:37B.

COSMOS. ND's family of computer network products.

Database. A system used to store and retrieve information in an efficient way. The SINTRAN III subsystem SIBAS is the most common database system on ND computers.

Default. A value that is assumed if nothing is specified. For example, the default file type :DATA means that this file type is assumed if you do not specify the file type in a file name parameter.

Demand segment. The pages of a segment are normally moved from the disk to physical memory one by one when they are accessed. Demand segments allow this. Non-demand segments are moved as a whole. This gives faster access. The RT LOADER defines segments as demand or non-demand. See the SINTRAN III REAL TIME GUIDE (ND-60.133).

Device. In most cases equipment connected to the computer, eg., terminals, printers, links to other computers, and disks. Various types of internal devices exist in addition, eg., semaphores and buffers. Opened files are treated as devices. Most monitor calls identify devices by their logical device number. See appendix B.

Direct task. Programs run as an extension to the operating system. Direct tasks normally handle devices which require fast response.

Directory. For most practical purposes you can regard directories as groups of files. There is normally one directory for each disk or floppy disk. A file, for example, (PACK-ONE:JOHN)TEST:TEXT1, belongs to the directory called PACK-ONE.;

DMA. Direct memory access. Data is transferred between a device, eg., a disk, and memory one page at a time.

Domain. ND-500 programs are called domains. The address space of a domain is divided into segments. For example, one segment may contain the main program, and another a routine library. The routine library can be shared with another domain.

Echo. When you press a key on the terminal, a character is normally displayed. This is called echo. You may turn echo on and off. It is also possible to specify that only some characters should give echo.

End-of-file. An ASCII character which marks the end of a file. Abbreviated EOF.

ErrCode. A system defined variable in some programming languages. You can test the ErrCode to see if errors have occurred in a monitor call. If it is greater than 0 something is wrong. Appendix A describes the various ErrCodes.

Error code. Some monitor calls return an error code. If it is greater than 0, an error has occurred. In some programming languages, the error code is available in the system variable ErrCode. In MAC it is returned in the A register. ASSEMBLY-500 returns the error code in the W1 register. Appendix A shows the meaning of error codes.

Error device. A terminal used for error messages from RT programs and SINTRAN III itself. The error device is normally the console. This can be changed by the system supervisor.

Escape function. The ESCAPE key on the terminal normally terminates a program. This is called user break. You can turn this escape function off with DisableEscape. EnableEscape turns it on again.

Execution queue. The RT programs waiting for the CPU. See the SINTRAN III REAL TIME GUIDE (ND-60.133). The command @LIST-EXECUTION-QUEUE lists the contents of the queue.

Exception. Some programming languages have user defined exception handling. A program may execute a set of statements, called an exception handler, when special errors conditions occur. For example, monitor calls with illegal parameters may cause an exception.

File name. (PACK-ONE:FLOPPY-USER)EXAMPLE-FILE:TEXT1 is an example of a complete file name in SINTRAN III. PACK-ONE is the directory name. FLOPPY-USER is the owner of the file. The file name is EXAMPLE-FILE. The file type is TEXT. The file version is 1. There are default values for most parts of the complete file name.
You may access files in remote computers when the COSMOS network is installed. The complete file name must then be preceeded by an identification of the remote system. A user on the remote system must be specified, eg., SNURRE(PACK-ONE:OLE(ABC:XYZ)).<complete file name>. SNURRE is the name of the computer. User OLE on the main directory PACK-ONE has password ABC and project password XYZ. You get the access rigths of user OLE. There are default values for most parts of the remote system identification.;

File number. A number given to a file when it is opened. The file number is used to identify the file in read and write operations. File numbers are a special kind of logical device numbers.

Fix. The pages of a program's address area are normally swapped between physical memory and the disk during execution. You can fix pages in physical memory. See FixScattered, FixContiguous, and similar monitor calls.

Flag. A parameter which may have two different values, eg., on or off. In a few cases, flags have been allowed to have 3 values.

FORTRAN. A high level programming language. Available on ND-100 and ND-500.

Friend. A user with special access privileges to your files. You may create up to 8 friends. Use the command @CREATE-FRIEND.

HDLC. A device which connects two computers in a network. One HDLC is installed in each computer. A cable connects them.

Indexed file. SINTRAN III's most common type of file. The size of the file adapts to what is written to it. Its pages are scattered around on the disk. The first page contains pointers to these pages. See CreateFile.

Internal device. Normally a character buffer or a semaphore.

Internal time. The time since SINTRAN III was started. It is specified in basic time units. There are 50 basic time units in a second.

Interrupt. A device sends an interrupt when it needs attention from the CPU. For example, an interrupt is sent when you press a key on the terminal. The ND-100 has 16 levels of interrupt. Each device is connected to one of these levels. The higher levels are handled first.

I/O wait. A waiting state where a program waits for input or output.

LAMU. Logically addressed memory unit. SINTRAN III's LAMU system is an extension to the segments. It makes it possible for background and RT programs to access a larger address area than available by the 3 segments. The address space of a LAMU may be shared by several CPUs.

Library. A set of compiled routines. Your program may call these routines if you load the library together with your program. The high level language interface to the monitor calls is provided as a library.

Local function. You can log in on remote computers if the COSMOS network is installed. Normally, a key on your terminal terminates this connection. DisableLocal turns this local function off. EnableLocal turns it on again.

Local system. The computer you are physically connected to in a data network. Opposed to remote systems.

Log in. You must log in before you can start working on a terminal. Press the ESCAPE key and enter your user name and your password. A project password is sometimes needed in addition.

Logical device number. A number that identifies a device. Opened files are given logical device numbers from 100-177. You may use logical device number 1 to access your own terminal from background programs. See appendix B.

Logical segment number. A domain in ND-500 numbers its segments from 0:31. These logical segment numbers are different from the physical segment numbers.

MAC. The ND-100 assembly language.

Mode job. Commands are read from a mass storage file instead of the terminal. Output is to a terminal. Mode jobs are used to execute commonly used sequences of commands in an efficient manner.

Monitor call. Programs request services from the operating system through monitor calls. Monitor calls look like routine calls. For example, a program may read the current time or write to a file by using monitor calls.

Monitor call number. Each monitor call has a number. This number is written in the headings of the reference chapter. The trailing B, eg., 256B, means that it is an octal number.

ND-100. ND's 16-bits computer. ND Sattelites and ND Compacts are ND-100 computers. ND-500 computers has a build-in ND-100 computer.

ND-500. ND's 32-bit computer. An ND-500 has both an ND-100 and an ND-500 CPU. Various models like ND-530, ND-570, etc., exist.

ND-500-MONITOR. An extension of SINTRAN III for ND-500 computers. It is run as a subsystem under SINTRAN III. ND-500-MONITOR provides almost the same set of monitor calls on the ND-500 as available on the ND-100.

No wait. A program normally waits until input and output are completed. The monitor call NoWaitSwitch will cause the program to continue in parallel with the input or output operation.

Object entry. An object entry describes each file. The command @FILE-STATISTICS outputs parts of this information. See the file system description in the SINTRAN III SYSTEM SUPERVISOR (ND-30.003)

Object index. Identification of an object entry.

Octal number. In the octal number system you count 1, 2, 3 ... 6, 7, 10, 11, 12 ... 17, 20, 21 ... 27, 30 ... 77, 100 ...177, 200 etc. The digits 8 and 9 do not exist. Octal numbers are written with a trailing B, eg., 64B. The V command in PED and NOTIS-WP converts to and from octal numbers. Our ordinary number system is called decimal.

Page. A page is 2048 bytes of physical memory or disk space.

Page table. SINTRAN III has 4 page tables. They are numbered 0-3. Each page table allows you to access 128 Kbyte memory. You may use more than one page table. Use AltPageTable to switch.

Parity. SINTRAN III uses even parity on ASCII characters. That is, the ASCII characters use bit 6-0 in a byte. The 7th bit is the parity bit. It is set if there is an odd number of bits set in bit 6-0.

For example, carriage return has the ASCII value 13. When input from a terminal, your program receives 141 because the parity bit is set. Subtract 128 to get the ASCII value, ie., 141 - 128 = 13. Note that the bit pattern for 13 is 00001101, ie., 3 bits are set. This causes the parity bit to be set. The value 10001101 is returned.

Pascal. A high level programming language. Available on both ND-100 and ND-500.

Project password. A additional password used by the accounting system. It is requested when you log in. Normally, all users working on a common project have the same project password.

Patch. Alter or extend the machine instructions of the operating system or possibly a subsystem. This is normally done with the command @LOOK-AT or the DMAC subsystem.

Periodic program. An RT program that is started automatically at regular intervals. See the monitor call StartupInterval.

Peripheral device. Equipment connected to the computer, eg., printers, floppy disk drives, and terminals.

Peripheral files. Peripheral equipment, eg., terminals, printers and floppy disk drives, are handled as files for input and output. See SetPeripheralFile.

Permanently opened file. Files may be set permanently open by the monitor call SetPermantOpen. CloseFile with -1 as parameter will not close such files.

PIOC. A programmable input and output processor. Commonly used in data communication. See the manual PIOC SOFTWARE GUIDE (ND-60.161).

PLANC. A high level language particularly for ND computers. Available on ND-100 and ND-500.

Physical device number. Each device is identified by a physical device number. It is used in the IOX instruction when accessing the device.

Physical memory. The main memory of a computer. A memory management system allows ND-100 programs to address 128 pages of virtual memory. The virtual memory pages are swapped into physical memory when accessed.

Process. A program in the ND-500 is called a process.

Protection ring. The ring determines which machine instructions a program may execute. Each segment belongs to one of four rings. The rings are numbered from 0 to 3. Background programs use ring 0. See the SINTRAN III REAL TIME GUIDE (ND-60.133).

Random access. When you read or write to specified addresses in a file. The opposite is sequential access where you access the bytes one by one from the start.

Reentrant program. A program that may be used by several users at a time. Reentrant programs are placed on a segement, not on a file. This saves time and space.

Reentrant segment. An ND-100 segment with a reentrant program. Use the command @DUMP-REENTRANT or the RT-LOADER to place a program on a reentrant segment.

Reference. Documentation intended to look up particular details. Prerequisite knowledge of the matter is needed.

Remote system. A computer installation accesssed via a data network. Opposed to your local system.

Resource. In this documentation, a resource is either a device or an opened file.

Restart flag. Each RT description has a restart flag. If set, the program restarts immediately when it terminates.

Resident memory. An area in physical memory used by the most important parts of SINTRAN III. The pages in this area are never swapped out to the disk.

Ring. The ring determines which machine instructions a program may execute. Each segment belongs to one of four rings. The rings are numbered from 0 to 3. Background programs use ring 0. See the SINTRAN III REAL TIME GUIDE (ND-60.133).

RT Description. A block of information about each RT program. Each RT description is identified by an address in a table of all RT programs. GetRTAddress or @LIST-RT-PROGRAMS gives the address. Further information is found in the SINTRAN III REAL TIME GUIDE (ND-60.133).

RT LOADER. A loader needed to load RT programs on the ND-100. See the manual SINTRAN III RT LOADER (ND-60.051).

RT program. Real time program. RT programs may use monitor calls not available to background programs. All RT programs must be loaded with the RT-LOADER, not the BRF-LINKER. User RT owns the RT programs. See the SINTRAN III REAL TIME GUIDE (ND-60.133).

Scratch file. A scratch file is connected to each terminal. It is opened automatically when you log in. Some subsystems, eg., PED, use the scratch file. You may also use it for temporary storage of information. It has file number 100. User SCRATCH owns the scratch files. The scratch file is normally reduced to 32 pages when you log out.

Segment. An address area where programs are placed before execution. Segments are areas on the disk. During execution pages are swapped in and out of main memory. The SINTRAN III REAL TIME GUIDE (ND-60.133) describes segments on the ND-100. Segments on the ND-500 differ from the ND-100. See the ND-500 LOADER

Semaphore. A device used to synchronize RT programs. For example, the semaphore allows you to control that only one program accesses a device at a time.

Sequential access. When you read from or write to a file byte by byte from the beginning to the end. The opposite is random access.

Short name. Each monitor call has a short name. For example, the short name of ExactInterval is DIntv. This is an old name. It is kept for backward compatibility.

SIBAS. The database system on ND computers.

SINTRAN III. The operating system on ND computers. It provides services through commands and monitor calls.

SINTRAN-SERVICE-PROGRAM. A program the system supervisor uses to make changes in SINTRAN III. See the manual SINTRAN III SYSTEM SUPERVISOR (ND-30.003).

Skip return. Skip return from a monitor call means that the first instruction following the monitor call is ignored. MAC uses skip return if no standard error code is returned.

Spooling. A system to queue files you want to print when the printer is occupied.

Standard error code. Some monitor calls return an error code. If it is greater than 0, an error has occurred. In some programming languages, the error code is available in the system variable ErrCode. In MAC it is returned in the A register. ASSEMBLY-500 returns error codes in the W1 register. Appendix A shows the meaning of error codes.

Swap file. The pages of a segment in the ND-500 are swapped between physical memory and the disk. The segment is copied to a swap file if you do not want to modify the original contents of the segment. There is one or more swap files in each ND-500 system.

TAD. Terminal access device. Used by COSMOS when you log in on a remote computer.

Temporary file. The contents of a temporary file is deleted the first time the file is read. Use the monitor call SetTemporaryFile to make a file temporary.

Terminal. In this manual, a terminal means a video display unit and a keyboard unless otherwise specified.

Terminal type. A number identifying each type of terminal. See appendix H.

Timeout. A timeout may be specified in various waiting situations. This will be the maximum waiting time.

Time queue. The RT programs which are set to start at a particular time. @LIST-TIME-QUEUE lists the contents of the queue. Further information in the SINTRAN III REAL TIME GUIDE (ND-60.133).

Time slicer. The time slicer allows programs to share the CPU. Each program executes for a short interval. Uncompleted programs return to the execution queue.

Trap handling. Detected error conditions, such as division by 0, or protect violation, cause a hardware trap in the ND-500. Either a standard trap handler or a user defined set of statements may be executed.

User name. In SINTRAN III, every user has a user name, eg., P-HANSEN. Up to 256 users may be defined in each directory. They have user indexes from 0 to 255. The monitor call ExecutionInfo returns the user index if needed in a program.

User break. A background program is terminated when the user presses the ESCAPE key. This is called user break. Use DisableEscape to inhibit user break.

User Environment. A subsystem which makes the operating system more user friendly. It is mainly intended for office automation users.

User RT. A privileged user with access to real time facilities. User RT owns all RT programs. See the SINTRAN III REAL TIME GUIDE (ND-60.133).

User SYSTEM. The user name of the system supervisor. User SYSTEM may use all commands and monitor calls.

Virtual memory. The address space available to programs. The memory management system automatically swaps pages into physical memory when your program accesses them.

Volume. A set of files on a magnetic tape or floppy disk. A volume has a function similar to a directory. Files in a volume can only be accessed sequencially. See the BACKUP-SYSTEM description in the SINTRAN III UTILITIES MANUAL (ND-60.151).

Index

********** 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 Monitor Calls Manual number: ND-60.228.1 EN

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