SINTRAN III
Communication Guide

ND-60.134.02

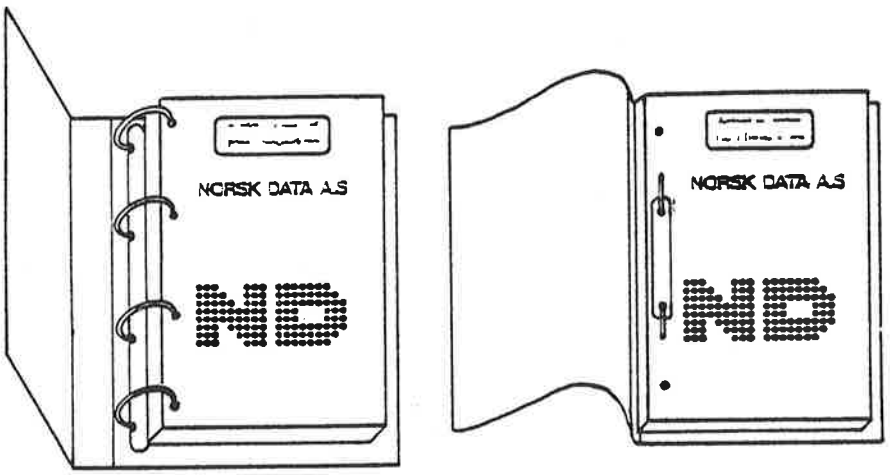
NOTICE

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

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

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

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



A Ring Binder B Plastic Cover

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

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

ORDER FORM

I would like to order

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

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

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

Name .....
Company .....
Address .....
.....
City .....

PRINTING RECORD

Printing	Notes
09/80	VERSION 01 (SINTRAN III Special I/O Guide)
11/81	Version 02

SINTRAN III Communication Guide

ND -60.134.02

NORSK DATA A.S
P.O. Box 4, Lindeberg gård
Oslo 10, Norway

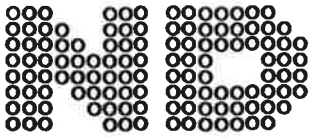
Manuals can be updated in two ways, new versions and revisions. New versions consist of a complete new manual which replaces the old manual. New versions incorporate all revisions since the previous version. Revisions consist of one or more single pages to be merged into the manual by the user, each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

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

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

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

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



Norsk Data A.S M A N U A L

SIII COMMUNICATION GUIDE

THE READER

This manual is intended for the time-sharing user who needs a users guide to the communication functions in SINTRAN III.

PREREQUISITE KNOWLEDGE

SINTRAN III TIME-SHARING-BATCH GUIDE (ND-60.132)

THE MANUAL

This manual describes commands and monitor calls used mostly by time-sharing-batch users. The functions are ordered by functional category as opposed to the SINTRAN III REFERENCE MANUAL where most of these functions are documented in alphabetical order. In the computer examples, user input is underlined.

"..." denotes a single control key. For ex. "rub-out" means pressing the "rub-out" key. Related manuals are:

SINTRAN III TIME-SHARING-BATCH GUIDE (ND-60.132), and
SINTRAN III REAL TIME GUIDE (ND-60.133)

Other related SINTRAN III manuals are:

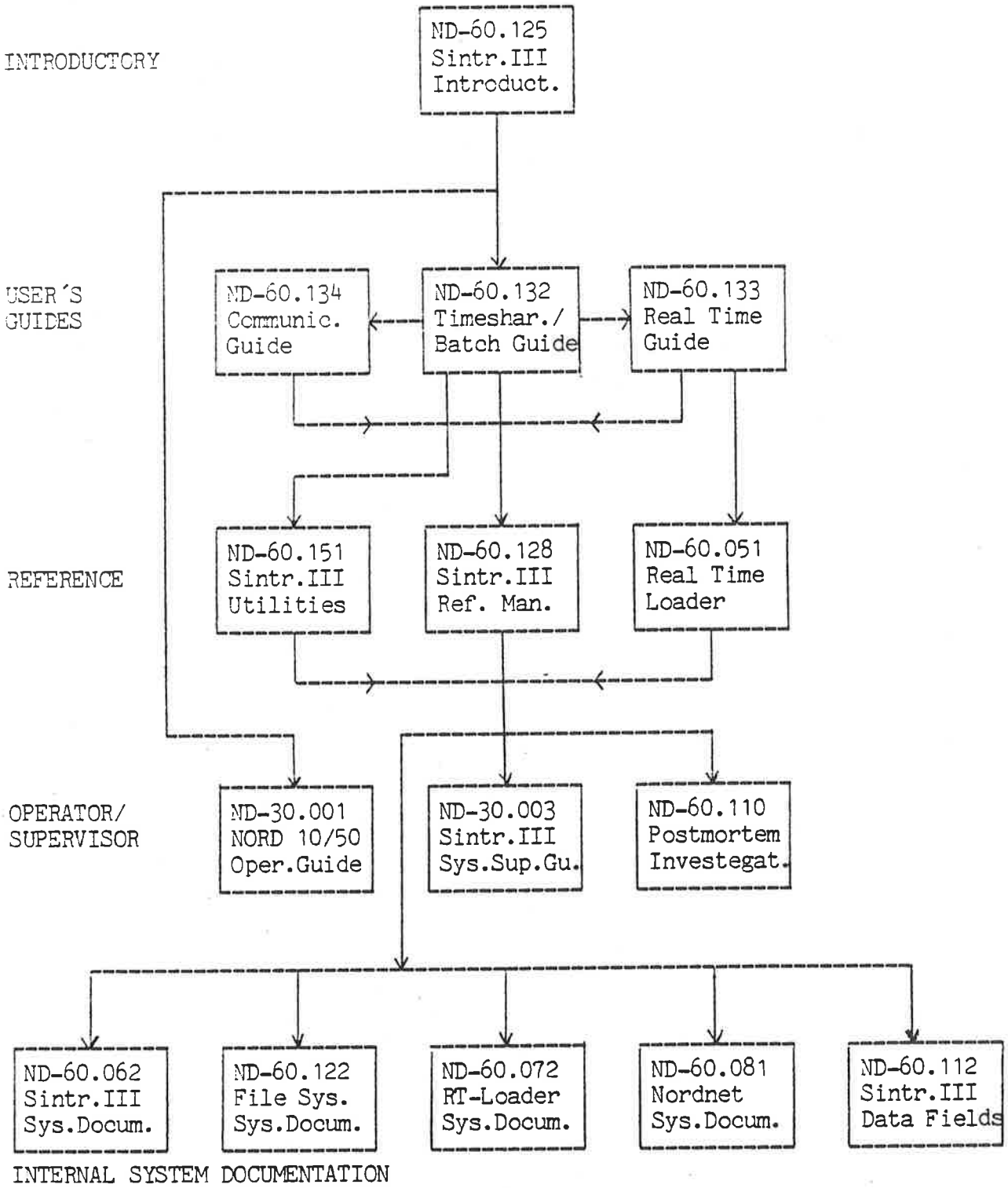
SINTRAN III REFERENCE MANUAL (ND-60.128),
SINTRAN III SYSTEM SUPERVISOR (ND-60.103), and
SINTRAN III RT LOADER (ND-60.051)

This manual partially obsoletes SINTRAN III Users Guide (ND-60.050) (see ND Bulletin no. 4, 1980).

THE PRODUCT

This manual documents the SINTRAN III VS version F.

SINTRAN III/VS



SINTRAN III/RT

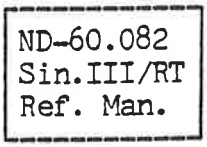


TABLE OF CONTENTS

Section Page

APPENDICES

1. INTRODUCTION

Most SINTRAN III users handle local input-output through the file system. This manual is a guide to the data communication functions which can be used on the local peripheral equipment. It contains documentation on the functions available in the SINTRAN III communication software.

All commands are available for ordinary time-sharing users unless otherwise noted. Similarly, all monitor calls are public unless otherwise stated.

Chapter 1 gives an introduction to the manual.

Chapter 2 is a guide to the NORD-NET communication system. NORD-NET enables a user in the local computer to communicate with other NORD computers through a distributed data network. Resource-sharing and inter-program communication are implemented, forming an extension to the SINTRAN III operating system.

The NORD-NET architecture is based on node-to-node connections. There is no master/slave relationship imposed in the architecture. This makes it possible to arrange different types of network structure to suit various user requirements. The network organization may be hierarchical, ring or star.

Chapter 3 is a guide to Remote Job Entry (RJE) commands in SINTRAN III. This is implemented as software packages for emulating RJE to CDC, Honeywell, IBM, SIEMENS and UNIVAC equipment.

The RJE packages together with packages for interactive terminal communication on IBM 3270, Honeywell VIP 7750 and Univac UTS-400 constitute NORD Intelligent Data Terminals (IDT software packages).

Chapter 4 is the complete documentation on XMSG task-task message system (Also called X-message.) A task can be a foreground or background program, a direct task or a peripheral equipment driver. The main features of X-message are:

Data Transfer The transfers are message oriented, i.e. data is transferred in units of variable length messages. Security is provided by making it impossible to overwrite the data while it is being transferred. Many tasks may be talking to one receiver (Fan in).

Addressing Ports may be named by a string of ASCII characters. The access is checked for validity. Abnormal situations, such as abort, escape, etc. will be handled properly.

Syncronization A process may wait for a message or it may be woken up upon arrival of the next message (MON RT).

General All functions are available from foreground and background programs, direct tasks and drivers.

XMSG assimilates the functions of internal devices.

Chapter 5 describes the monitor call for the HDLC driver. HDLC stands for High Level Data Link Control, a data transmission format defined in the ISO 3309 standard. The driver makes it possible for application programs to use the HDLC interface (ND-720 or ND-730).

Finally, chapter 6 describes the monitor call for the X.21 communication protocol. It makes use of the HDLC driver, and makes calling and searching on a line switched network easy for the user.

The X.25 procedure on packet level switching is described in a separate manual.

2. NORD-NET

2.1. Introduction

The NORD-NET communication system is an optional part of the SINTRAN III I/O system. Its purpose is to provide communication between two or more independent NORD computer systems. The communication can be divided into four categories.

1. Remote terminal communication. A user of a local terminal may use commands and run programs in the remote computer as if his terminal was connected directly to that computer.

2. Remote file access. Files on a remote computer may be accessed by commands or monitor calls as if they were local files. However, then only the functions for open, close, read and write are available

3. Data transfer. A remote and a local program may communicate directly through the channels in a fashion similar to using an internal device.

4. Remote load. The remote computer may be loaded from the local computer. Only main memory can be loaded.

Since all communication occurs on serial lines, the line transmission speed may be a limiting factor.

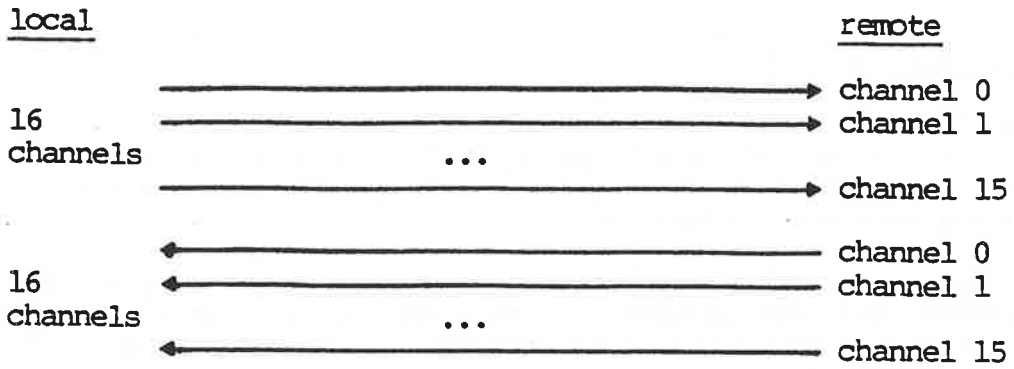


Figure 2-1 A Communication Line

This chapter describes points 1 to 3. Remote load is described in the SINTRAN III SYSTEM SUPERVISOR manual. Besides Remote Load, the system supervisor is responsible for starting and stopping the communication on a line (use @START-COMMUNICATION and @STOP-COMMUNICATION).

User RT can associate a password with remote file access (@REMOTE-PASSWORD). A further guide to this command can be found in the manual SINTRAN III REAL TIME GUIDE (ND-60.133).

2.2. The Communication Line

2.2.1. General

The communication line can be divided logically into a maximum of sixteen channels each way (figure 2-1).

They are numbered from zero to fifteen. If more channels are required, another communication line must be added.

Each channel is provided with a buffer on either side. A buffer is scheduled for transmission either when it is full or when a break character is written to the buffer.

The set of break characters may be choosen by the user.

Information is transmitted in units called communication frames. Acknowledgement for correctly received frames are transmitted together with the frames returned to the sender.

Up to four frames may be transmitted without receiving acknowledgement. This is done by dividing the buffers into four groups.

For each group, the buffer is not discarded until acknowledgement for this group is received. The buffers for sending are always directed to the four groups in a cyclic manner to ensure a correct sequence.

On the receiving side, they are distributed in the same cyclic manner.

When a buffer is transmitted, it is preceded by a buffer header and followed by a cyclic check sum.

A logical device number (LDN) is assigned to the channel on either side. It may be reserved, released, and accessed in a similar manner to any other device in SINTRAN III.

The LDN on either side may be of a different value (figure 2-2).

The various channels can be interrogated by the commands shown in the next section.

A channel with an associated background program can only be used for remote terminal communication.

Such channels are marked with "BACKGROUND" in the report made by these interrogation commands.

A channel without a background program is used for remote file access and data transfer.

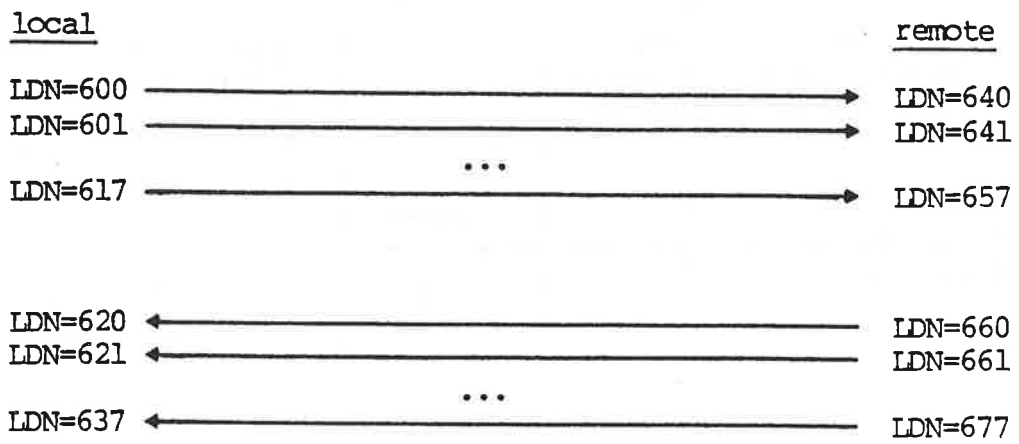


Figure 2-2 Example of Logical Device Numbers in NORD-NET

2.2.2. @COMMUNICATION-STATUS

@COMMUNICATION-STATUS <line number>,<output file>

Report the status of the <line number> on the <output file>. The report contains logical device numbers, background vs. data channels, error information and the current communication state.

2.2.3. @COMMUNICATION-LINE-STATUS

@COMMUNICATION-LINE-STATUS <line number>

This command produces an abbreviated report containing only error information and the current communication state.

2.3. Remote Terminals

2.3.1. General

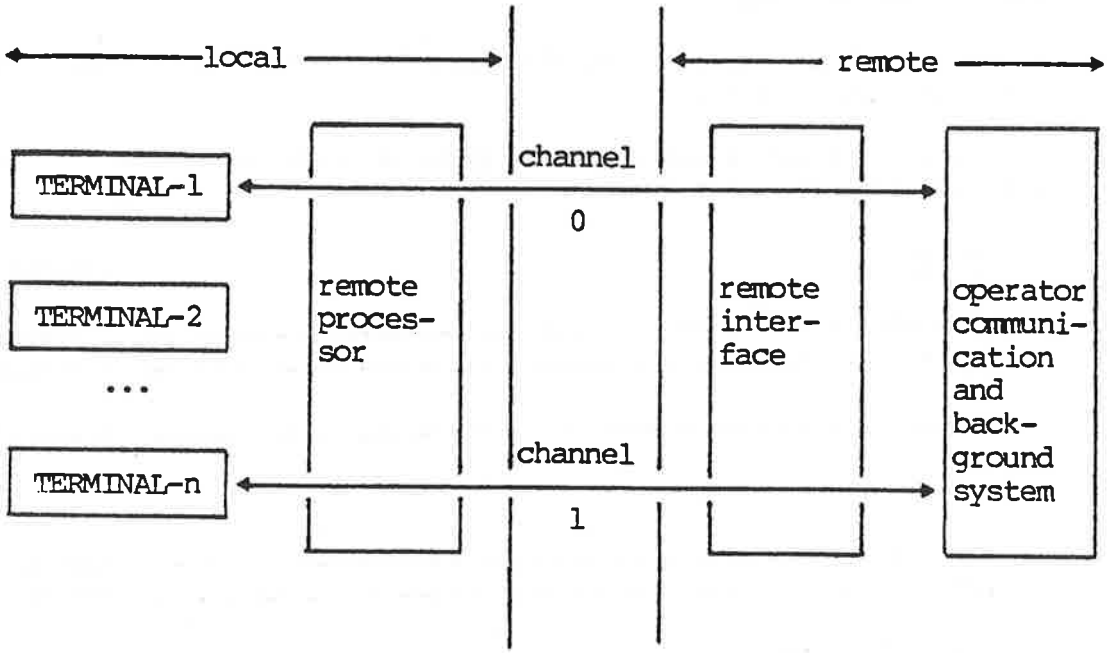


Figure 2-3 Remote Terminal Processing

The channels marked BACKGROUND in the report, mentioned above, can be connected to a remote processor. These channels are used for communication between a terminal user in a local SINTRAN III system and the operator's communication and background system in a remote SINTRAN III system. Figure 2-3 shows the main parts of the NORD-NET implementation.

A terminal user may connect to the remote processor by typing the command @REMOTE <line number> on his terminal. A free channel will be allocated, if available, to the terminal. The user may now LOG IN on the remote system. "rub-out" or "del" puts him temporarily back to the local command processor. The channel is still allocated to the terminal. Another @REMOTE with the same <line number> puts him back to the remote command processor. If he instead types @LOCAL the channel will be disconnected. (A more detailed description is found in section 3.3.5.

For example:

```
   ...
local processing
   ...
@REMOTE 1
CHANNEL NUMBERS: LOCAL -600 REMOTE -600
"escape"
15.54.20 18 APRIL 1980
ENTER OLE
PASSWORD:
OK
R@
   ...
remote processing
   ...
R@LOGOUT
16.11.34 18 APRIL 1980
-EXIT-
"rub-out"
@
   ...
   local processing
   ...
```

Remote command mode is indicated by R@ as prompt characters instead of only the @ alone.

A user can be connected to only one line at a time. Thus, if he is connected to remote line 1 and wants to change to remote line 2, it is done as follows,

1. Log out as remote user (R@LOGOUT).

2. Type "rub-out".

3. Type @LOCAL.

4. Type @REMOTE 2

5. Log in as remote user on line 2.

Typing "rub-out" to the remote command processor while in remote command execution mode or remote user mode causes a return to local mode, but the remote processing will continue. The terminal output will be saved and displayed when the user returns to remote command processing.

2.3.2. @REMOTE

@REMOTE <line number>
Connect terminal to remote command processor. If no remote connection exists for this terminal (no @REMOTE since last @LOCAL) a free channel is found and the terminal is connected to the background processor of the remote computer. If a remote connection already exists, the terminal is connected to this channel. In the latter case, @REMOTE has the reverse function of "rub-out".

2.3.3. @LOCAL

Disconnect remote connection. The communication channel used by the remote connection is released and may be used for other purposes.

2.3.4. Example of @REMOTE and @LOCAL

In this example, @REMOTE and RUB-OUT are used to connect to and disconnect from the remote system.

```
"escape"
15.25.56  5 SEPTEMBER 1980
VERSION 80.02.01A
ENTER TOM
PASSWORD:
OK
@DATCL
15.26.12  5 SEPTEMBER 1980
@WHO

   1  TOM
  38  GROUP-4
 670  SYSTEM
 672  SYSTEM
@REMOTE
CHANNEL NUMBERS: LOCAL -600, REMOTE -600
"escape"
15.25.25  5 SEPTEMBER 1980
ENTER SYS
PASSWORD:
OK
R@DATCL
15.25.39  5 SEPTEMBER 1980
R@WHO

   1  RT
 670  SYSTEM
 672  SYSTEM
 384  SYSTEM
R@"rub-out"
VERSION 80.02.01
@DATCL
15.27.03  5 SEPTEMBER 1980
@REMOTE
R@LOG
15.27.01  5 SEPTEMBER 1980
-- EXIT --
"rub-out"
VERSION 80.02.01
@LOCAL
@LOG
15.28.18  5 SEPTEMBER 1980
-- EXIT --
```

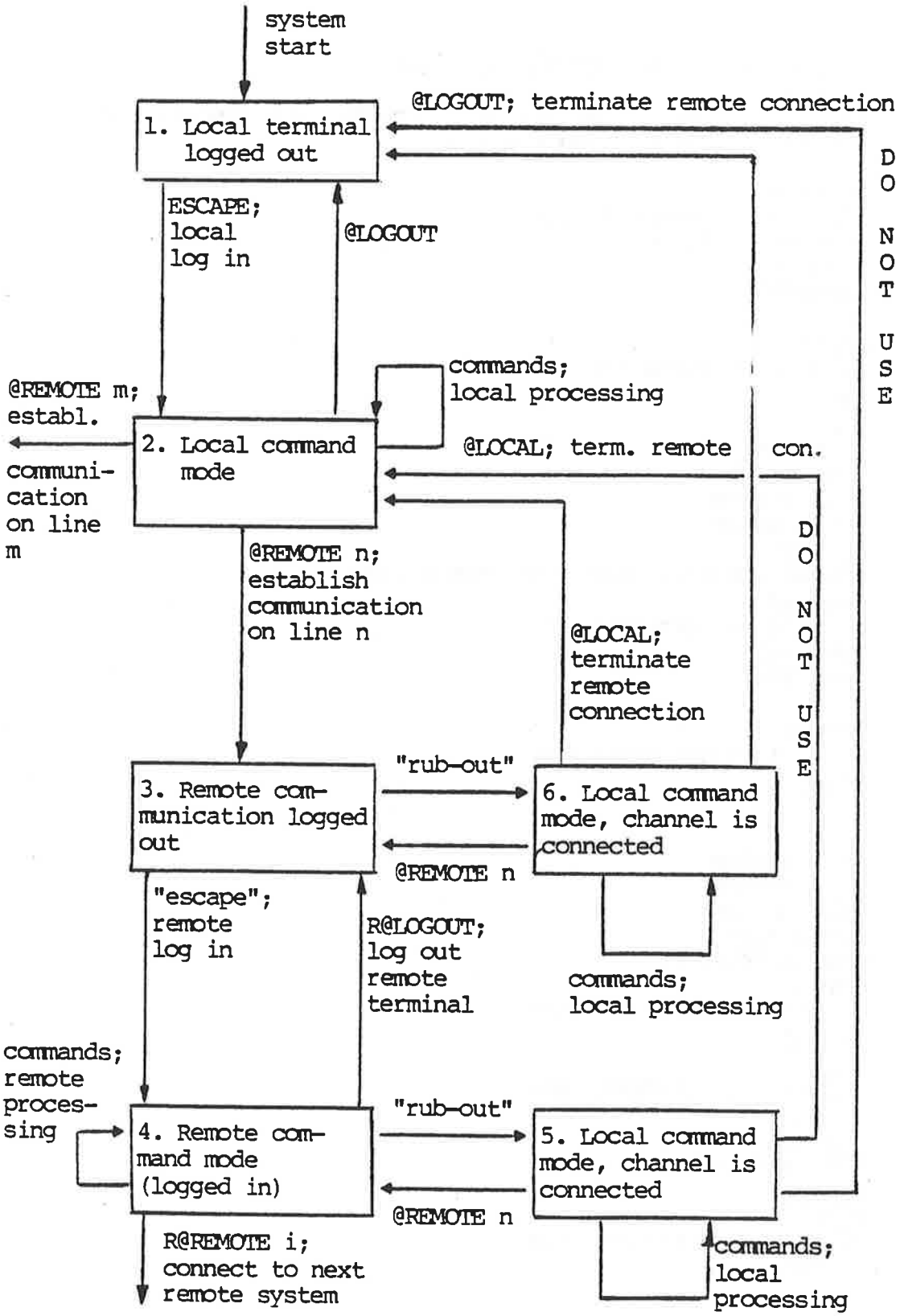


Figure 2-4 Remote Processing, State Diagram

2.3.5. Detailed Description of Remote Terminal Connection

This section is a thorough description of the states of remote terminal connection.

The state diagram of remote terminal connection is shown in figure 2-4. The diagram should be compared to figure 2-4 in the SINTRAN III TIME-SHARING/BATCH GUIDE.

State 1 of figure 2-4 is equivalent to state 1 of figure 2-4.

State 2 of figure 2-5 is equivalent to the other states of figure 2-4 except the response to @REMOTE.

This command establishes communication by reserving a free channel ,if any. The state is changed to 3.

Normally, the user proceeds to state 4 by logging on remotely.

All commands will now be processed on the remote system. Typing "rub-out" causes a transfer to local command mode, state 5.

Note that "rub-out" may be entered while in remote command processing mode or remote user mode.

This causes any remote processing to proceed as an independent process while the next commands are processed locally.

Any remote terminal output during state 5 will be collected and displayed when returning to state 4.

Another @REMOTE n command will cause a transfer to remote command mode, state 4, using the same channel as before.

In state 5 it is possible to terminate the remote connection (@LOCAL or @LOGOUT), but this should be avoided. The remote processing may be left in an indeterminate state. Instead, go back to state 3, type "rub-out" to change to local communication, and type @LOCAL to terminate the remote connection. (@LOGOUT will also terminate the connection.)

State 6 is equivalent to state 5 with respect to handling @REMOTE n. The state is changed to remote communication, state 3, using the same channel as before.

Nesting of remote connections is performed in state 4 by typing another @REMOTE m command. It could even be a remote connection back to the local system.

This is necessary for the type of processing shown in figure 2-5. Here, the user runs a remote program which uses one or more files in the local system.

The session starts with the first @REMOTE n command, establishing the interactive dialog on channel a. The user must then establish channel b by means of a @REMOTE m command back to the local system.

Then log onto the local system and finally type "rub-out" to get back to the remote system. The user may now start the remote program (program x) which can be run either in foreground or background.

The program will use a third channel for data transfer (channel c) while channel b will be used for administration.

Transfer of data can only occur directly between two systems which have a direct connection.

If system A connects to system B and B connects to system C, it is possible to connect to B, log in on B, connect to C and log on to C. However, any data transfer from A to C must first be made to B and then to C.

An intermediate file or program in B will take care of this problem.

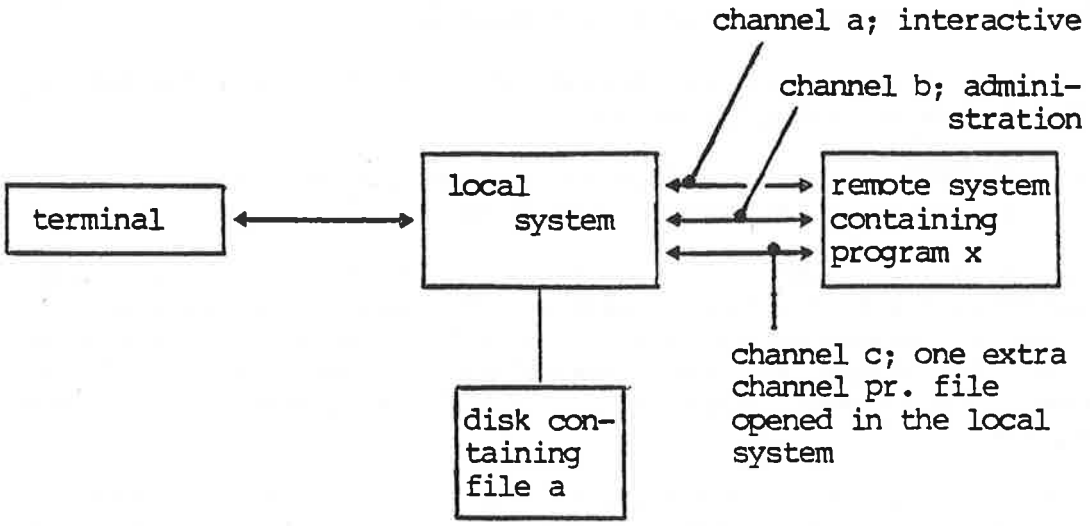


Figure 2-5 Remote Program Using a Local File.

2.4. Remote File Access

Files on a remote NORD system may be opened and closed from the local system. Only open, close, input and output functions are permitted on remote files. The communication channel to be used is specified as a prefix to the file name. To open a remote file from a background program, the user must be logged in on both systems. To open a remote file from a foreground program, the password of user RT on the remote system must be set by the @REMOTE-PASSWORD command. In the following example, the local file LFIL is read by QED and written to the remote file RFIL. The channel to be used is CHANNEL-1 and must have been defined as a peripheral file by user SYSTEM (@SET-PERIPHERAL-FILE). LFIL is owned by user PER and RFIL by user OLE.

The use of CHANNEL-1 as a prefix to the remote file name is explained at the end of this section.

```
"escape"
12.1.06  28 AUGUST  1980
ENTER PER
PASSWORD:
OK
@REMOTE
CHANNEL NUMBERS:  LOCAL - 600, REMOTE - 600
"escape"
12.10.12  28 AUGUST  1980
ENTER OLE
PASSWORD:
OK
R@"rub-out"
@QED
QED 3.8
*R LFIL
2480 WORDS READ
*W CHANNEL-1.RFIL
2480 WORDS WRITTEN
*EX
@REMOTE
R@LOG
12.12.04  28 AUGUST  1980
-- EXIT --
"rub-out"
@LOG
12.12.15  18 AUGUST  1980
-- EXIT --
```

Note that when the remote file is accessed, the own user name is the name used when logging in on the remote system. The file RFIL is expected to be among the files owned by user OLE or user SYSTEM on the remote system.

When processing the command *W CHANNEL-1.RFIL, two channels are used. The first is the one allocated to commands. The second is the channel corresponding to the peripheral file name CHANNEL-1.

The next example is a compilation in the local system. The FORTRAN source file LFIL is compiled. The compiler listing is output to the remote line-printer and the object code is output to the new remote file OBJ:BRF belonging to user OLE.

```
"escape"
12.25.18  28 AUGUST  1980
ENTER PER
PASSWORD:
OK
@REMOTE
CHANNEL NUMBERS:  LOCAL - 600, REMOTE - 600
"escape"
12.25.30  28 AUGUST  1980
ENTER OLE
PASSWORD:
OK
R@"rub-out"
@FTN
NORD-10 FTN COMPILE
$COM LFIL,CHANNEL-1.L-P,CHANNL-2."OBJ"
189 STATEMENTS COMILED, OCTAL SIZE = 3122
CPU-TIME USED IS 6.9 SEC.
$EX
@REMOTE
R@LOG
12.32.31  28 AUGUST  1980
-- EXIT --
"rub-out"
@LOG
12.32.40  28 AUGUST  1980
-- EXIT --
```

In general, the syntax of a remote file name is:

<channel name>.<local file name>

Only one level of remote connection can be specified. For ex.:

```
CHANNEL-1.(PACK-ONE:PER)FILA:SYMB;2
CH-2."FILE-2"
KANAL4.(SYS)FTNLIBR:BRF
```

The file number returned from a remote @OPEN-FILE, OPEN statement or OPEN monitor call will be the logical device number of the channel as defined in the local computer. After the file is opened, a remote file may be accessed by READ and WRITE statements, INBT and OUTBT monitor calls, etc.

2.5. Data Transfer

2.5.1. General

Any free channel not dedicated to backgrond programs may be used for data transfer. The channel will then have a function similar to an internal device. The only difference is that the sending and the receiving programs are in two different systems. The following rules apply:

1. The channel must be reserved by the user programs. The logical device number on either side is used for reservation.

2. The receiving program asks for input (by using a monitor call or statement) and is set in a waiting state until data is received through the channel.

3. The sending program outputs to the channel (by using a monitor call or statement) and is set in a waiting state under one of the following conditions:

- the receiving program has not asked for input,

- a break character is sent,

- the buffers available are almost full, or

- a wait acknowledge (WACK) is received from the channel.

The sending program is restarted when a request for input is received through the channel. The request is sent from the receiving program when it detects a break character.

A wait acknowledge is sent if the input queue for a channel exceeds a predefined number of buffers. (The number is defined at system generation time.) The wait will prevent one channel occupying the whole buffer pool if the receiver reads data at a lower rate than the sending program.

A wait acknowledge simulates a break character at the end of the last transmitted buffer on the channel.

4. The break strategy may be defined by the receiving program. The strategy is transmitted to the sending system as a special buffer. It is possible to specify no break is permitted occur.

A buffer will then be transmitted only if it is full or the sending program executes CLOSE-FILE or IOSET function on the channel. The break strategy should cause as few breaks as possible in order to reduce the system overhead.

The following standard monitor calls may be used on a communication channel:

RESRV - Reserve channel

RELES - Release channel

INBT - Read a byte

OUTBT - Write a byte

B8INB, M8INB, B4INW - Read 8 bytes

B8OUT, M8OUT - Write 8 bytes

CIBUF - Clear input buffer

COBUF - Clear output buffer

IOSET - On input: function = -1: clear input buffer
-2: set break strategy
On output: function= -1: send last buffer
-2: clear output buffer
>1000: the receiver will display an error message corresponding to (function - 1000).

BRKM - Set break strategy

ECHOM - Set echo strategy. The command will only have an effect if the program on the other side is a remote terminal processor.

MAGTP - Transfer a block of data to or from a communication channel.

RFILE - Read a block of data from a communication channel.

WFILE - Write a block of data to a communication channel.

In the following example, data is received from the communication channel having the LDN = 600:

```
        LDA  (REPAR       %
        MON  122          % RESERVE CHANNEL
        LDT  (600         %
        MON  13           % CLEAR INPUT BUFFER
        JMP  ERROR        % ERROR EXIT
        SAA  -1           %
        MON  4            % SET BREAK STRATEGY
LOOP,   LDT  (600         %
        MON  1            % INPUT A BYTE
        JMP  TEST         % TEST FOR ERROR
         .
         .
         .
        JMP LOOP          %
         .
         .
         .
% IF ERROR = 161, NO ANSWER FROM DEVICE,
% THEN TRY AGAIN AFTER 5 SEC. (CAN BE LIMITED
% TO FOR EX. 40 RETRIES.) THE DRIVER HAS NO
% RETRY FACILITY.
TEST,   SAT 161           % T = 161
        SKP EQL SA DT     % TEST FOR 161
        JMP ERROR         % NOT 161 - ERROR
        LDA (HPAR           WAIT 5 SEC.
        MON 104           % HOLD
        JMP LOOP          % TRY AGAIN
         .
         .
         .
REPAR,  (600              % CHANNEL NUMBER
        (0                % INPUT BUFFER
        (0                % WAIT FOR RESERVATION
HPAR,   (5                % 5 SEC.
        (2                % UNIT OF SECONDS
)FILL
```

The input buffer should always be cleared, since the previous program using the channel may have been terminated abnormally.

The following FORTRAN program will write a record to the channel:

```
       ...
      I=RESRV(600B,1,0)
      I=IOSET(600B,1,0,-1)
       ...
      WRITE(600B,10) ...
10    FORMAT( ...
       ...
```

The corresponding program to read is:

```
       ...
      I=RESRV(600B,0,0)
       ...
      READ(600B,10) ...
10    FORMAT( ...
       ...
```

2.5.2. WRQI (MON 163)

Place the calling program in a wait state until a request for input is received from the remote system. The call is useful in interactive communication programs when the local echoing should wait until the receiving program asks for input.

2.5.3. Example of a foreground data transfer program

In this example, two foreground programs will be seen. A sending program, FIRSTR will run in the remote system and a receiving program, FIRST, will run in the local system:

FIRSTR

- Read records of 5 characters from a file (RT)SEND and send them through channel 603.

- Terminate when reading EOF from the file.

FIRST

- Read records of 5 characters from channel 603 and write them on the terminal.

- Terminate when reading EOF.

```
"escape"
15.18.02  19 OCTOBER  1979
VERSION 78.10.18.B
ENTER RANDI
PASSWORD:
OK
@FTN
NORD 10 FORTRAN COMPILER FTN-2090F
$COM FIRST,1,FIRST
     1*          PROGRAM FIRST,45
     2*          CHARACTER IARR*6
     3*          CALL RESRV(603B,0,0)
     4*          I=IOSET(603B,0,0,-1)
     5*    1     READ(603B,*,END=10)IARR
     6*          CALL RESV(1,1,0)
     7*          WRITE(1,*)IARR
     8*          CALL RELES(1,1)
     9*    10    CALL RELES(603B,0)
    10*          STOP
    11*          END

11 STATEMENTS COMPILED , OCTAL SIZE=   200
CPU-TIME USED IS 0.5 SEC
$EX
```

```
@REMOTE
CHANNEL NUMBERS:  LOCAL -   600,  REMOTE -   600

"escape"
 15.15.10  19 OCTOBER  1979
VERSION 78.10.18.B
ENTER RANDI
PASSWORD:
OK
R@FTN
NORD 10 FORTRAN COMPILER FTN-2090F
$COM FIRSTR,1,FIRSTR
     1*         PROGRAM FIRSTR,45
     2*         CHARACTER IARR*5
     3*         IFILE=2
     4*         OPEN(IFILE,FILE='SEND',ACCESS='R')
     5*         CALL RESRV(603B,1,0)
     6*         I=IOSET(603B,1,0,-2)
     7*  1      READ(IFILE,*,END=10,ERR=20)IARR
     8*         WRITE(603B,*)IARR
     9*         GO TO 1
    10*  20     II=ERRCODE+1000B
    11*         I=IOSET(603B,1,0,II)
    12*         GO TO 100
    13*  10     I=IOSET(603B,1,0,1003B)
    14*  100    CALL RELES(603B,1)
    15*         STOP
    16*         END

16 STATEMENTS COMPILED , OCTAL SIZE=  172
CPU-TIME USED IS 1.0 SEC.
$EX
R@LOG
 15.15.54  19 OCTOBER  1979
--EXIT--
"rub-out"
VERSION 78.10.18.B
@LOCAL

@LOG
 15.19.52  19 OCTOBER  1979
--EXIT--
```

```
"escape"
 15.19.54  19 OCTOBER  1979
VERSION 78.10.18.B
ENTER RT
PASSWORD:
OK
@RT-LO

REAL-TIME LOADER 78.10.18B

*NREE (RANDI)FIRST,,
NEW SEGMENT NO:  65
*END-LOAD
*EX

@RT FIRST
@REMOTE
CHANNEL NUMBERS:  LOCAL -  600,  REMOTE -  600

"escape"
 15.16.45  19 OCTOBER  1979
VERSION 78.10.18.B
ENTER RT
PASSWORD:
OK
R@RT-LO

REAL-TIME LOADER 78.10.18B

*NREE (RANDI)FIRSTR,,
NEW SEGMENT NO:  111
*END-LOAD
*EX

R@RT FIRSTR

R@LOG
 15.17.49  19 OCTOBER  1979
--EXIT--
"rub-out"

VERSION 78.10.18.B
@LOCAL

@LOG
 15.21.51  19 OCTOBER  1979
--EXIT--

PER
PAAL
ESPEN ASKELADD
```

3. COMMANDS FOR REMOTE JOB ENTRY (RJE)

3.1. General

SINTRAN III can be delivered with software packages for emulating RJE terminals on several large mainframe computers. At present the available RJE emulators are;

ND-10026 CDC 200 User Emulator, manual ND-60.061

ND-10027 Honeywell GERTS 115, manual ND-60.041

ND-10028 IBM HASP Work Station, manual to be issued

ND-10029 UNIVAC NTR (for SINTRAN III/VS), manual ND-60.070

ND-10030 IBM 2780/3780, manual ND-60.067

ND-10031 UNIVAC DCT 2000, manual ND-60.060

ND-10056 UNIVAC NTR (for SINTRAN III/RT), manual ND-60.070

ND-10057 UNIVAC DCT 2000 (for SINTRAN III/RT), manual ND-60.060

ND-10063 IBM HASP Work Station DMA, manual to be issued

ND-10069 CDC 200 User Multidrop, ND-60.061

These emulators constitute the NORD Intelligent Data Terminals (IDT) together with the following interactive emulator packages,

ND-10016 IBM 3270, manual ND-60.114

ND-10059 Honeywell VIP 7750, manual ND-60.100

An emulator is loaded and started as a foreground program by user SYSTEM. Once the RJE emulator is running any user may append jobs to a batch queue (in the local computer) in a similar way to local batch. In general there are three ways in which a SINTRAN III user may run batch jobs,

1 , he may run local batch as explained in chapter 7 of the manual SINTRAN III TIME-SHARING/BATCH GUIDE. The jobs contain SINTRAN III commands.

2 , he may use NORDNET commands to run remote batch in another NORD computer. These jobs also contain SINTRAN III commands.

3 , he may use RJE commands to submit jobs to a host computer which is not a NORD computer. The jobs contain commands in the job control language of the host computer.

Batch jobs can be sent to the remote computer in two ways:

1) - when the emulator is started, a terminal is allocated as the remote batch console. This is normally terminal 2. The console is under control of the emulator and the user enters special emulator commands in order to send remote batch files. The files are not queued.

2) - a SINTRAN III command permits any time-sharing user to submit jobs to the remote computer. (User SYSTEM must have started the emulator from the remote batch console.) The jobs are queued in the local computer. The commands are shown below.

3.2. @APPEND-REMOTE

@APPEND-REMOTE <remote computer>,<input file>

Append a batch input file to the remote batch queue of a computer. For ex.:

@AP-REM UNIVAC,JOB-1

The batch input file JOB-1 is appended to the batch queue of the remote computer UNIVAC. The file must have read access for user RT.

3.3. @LIST-REMOTE-QUEUE

@LIST-REMOTE-QUEUE <host computer>

List the contents of a remote batch queue. For ex.:

```
@L-R-Q IBM
1 (SYSTEM)CARD-READER
2 (USER-NAME)IBMJOB
```

The queue contains two entries, one from the card reader and one from the file (USER-NAME)IBMJOB.

3.4. @DELETE-REMOTE-QUEUE-ENTRY

@DELETE-REMOTE-QUEUE-ENTRY <remote computer>,<queue entry>

Remove a remote batch input file from the queue for a remote computer. For ex.:

@D-R-Q-E UNIVAC,JOB-1

The file name JOB-1 is deleted from the queue for the remote computer UNIVAC.

4. XMSG - TASK-TASK MESSAGE SYSTEM (OPTION)

4.1. Introduction

Many applications require the division of a program system into separate, asynchronous processes or tasks, that communicate by sending messages.
This separation may be motivated by security considerations (separation of work-areas, definition of interface points), by hardware design (tasks may run in separate machines), by address space limitations, or simplicity of program development.

We will use the word task to mean a driver, direct task, or RT (foreground or background) program. The XMSG system allows tasks to send messages to each other, including handling of memory allocation, queueing, and task synchronisation.

A task can open ports through which it can send and receive information about messages. Data is normally transferred between tasks via message buffers within XMSG. The sending task first opens a port, then reserves an XMSG message buffer, transfers his data into that buffer and finally informs the receiving task's port that data is awaiting collection. Reservation and releasing of messages is done explicitly by the user.

XMSG facilities take two forms: Functions and Services:

XMSG functions are invoked via the XMSG monitor call (200) with parameters being passed in the registers. The T register indicates the particular function required with option bits set in its high order byte when required.

Completion status is returned in the T register, positive (precise meaning depends on the function) if successful, zero if the operation was not terminated and negative indicating an error.

The functions are divided into two groups: user functions (of general interest) and system functions (used mainly by XROUT and XMSG-COMMAND). The functions in each group are described in the corresponding sections below: 'User Function Specifications' and 'System Function Specifications'.

XMSG Services are invoked by sending messages (using functions) to a standard task called XROUT. The services and method for accessing them are described in the section 'XROUT Service Specifications' below.

Note that all function, service and error codes are referred to symbolically. Their values are defined in file XMSG-VALUES (see Appendix D), which should thus be used as an include file when compiling the task code. The routine XMERR in XMSG-LIBRARY converts an XMSG error code in the A-register to a pointer to an explanatory text returned in the A-register. The text is in ASCII characters terminated by a quote character (').

4.2. Single- and Multi-machine XMSG

XMSG can be configured in two ways:

Single-Machine XMSG only provides communication between tasks running in a single ND-100 CPU.

Multi-Machine XMSG (XMSGM) also allows communication (but not indirect buffer access) between tasks running in a group of ND-100s. The current XMSG version allows up to 64 machines per XMSGM network.

The following extra concepts are used in Multi-machine XMSG:

A machine is a Processing Unit that runs an independent XMSG kernel - (i.e. ND-100 CPU - not PIOC (Programmable I/O Controller) or ND-500 which are seen as part of a ND-100 since every PIOC or ND-500 task which uses XMSG has a 'shadow' task in the ND-100).

A link connects machines.

4.3. User Function Specifications

In the following descriptions these symbols will be used in the parameter lists (integer unless specified otherwise):

ISTAT - result status

XFxxx - function code (options are indicated in parentheses)

NBYTES - number of bytes

MESAD - message identifier (in fact an address on the XMSG segment)

UADD - user buffer address

ULEN - length of user data in BYTES

DISP - displacement within message in BYTES

PORTNO - local port identifier. If zero, the most recently opened port is assumed.

RPORT - remote machine no and port.

MAGNO - double word containing remote port identifier.

XFWTF - wait flag. Leads to IO-wait until the user specified function terminates.

XFWAK - wake flag. XMSG wakes up the RT program (RTENTRY) when the user specified function terminates.

QLEN - number of messages currently queued for a port.

DATA0 - first two bytes of user data.

The calls will be described by showing the NPL code required to use them. The user must remember that the T-register always contains the status on return (which should be checked!)

Some functions and services are privileged. Before calling these, a task must make itself privileged by invoking the XFPRV function described below in the section 'System Functions'. A short description of XMSG functions is given in appendix B.

4.3.1. Manipulating Ports

When a task opens ports they are identified locally with a port number (like a file number). A task identifies other tasks' ports using a 32 bit magic number (MAGNO) which comprises the port number, the machine number and a random part that guarantees that a port that is closed and then re-opened does not have the same identifier.

The current XMSG version allows addressing of up to 1020 ports per machine.

4.3.1.1. Opening Ports (XFOPN)

```
T:=XFOPN (BONE XFPRM); *MON 2XMSG
A=:PORTNO
```

A port is opened and its number returned to the calling task. If the XFPRM flag is set, the port is defined as permanently open and will only be closed by an explicit close of that port, or by a close (-2) - see XFCLS description below.

4.3.1.2. Closing Ports (XFCLS)

```
T:=XFCLS; A:=PORTNO; *MON 2XMSG
```

Closes the specified local port. If A=-1, all non-permanent ports will be closed. If A=-2, all ports, including permanently opened ones, will be closed.

When a port is closed, all 'non-secure' messages currently queued for that port are released, while all 'secure' messages (as well as the port current message, if any) are set 'non-secure' and returned to the sender.

A close (A=-1) is automatically executed whenever a background processor returns to the command input mode (@..) A close (A=-2) is automatically executed when a background user logs out or a foreground program terminates or aborts.

4.3.1.3. Port Status (XFPST)

```
T:=XFPST (BONE XFWTF/XFWAK); A:=PORTNO; *MON 2XMSG
A=:RPORT; A:=D=:MESAD; X=:QLEN
```

On return T indicates the message type of the first message in the queue (or 0, if there are none). If a message is waiting, D contains its address and A the machine number in the lefthand byte and the port number in the righthand byte of the port from which the message has been sent. The X register always contains the queue length. The message types and wait options are as for the receive function (XFRCV) described later.

4.3.1.4. General Status (XFGST)

A task may have many open ports, and not be sure to which one the next message is going to come. XFGST allows him to check all ports:

```
T:=XFGST (BONE XFWTF/XFWAK); A:=PORTNO; *MON 2XMSG
A=:PORTNO;
```

The call parameter PORTNO indicates where the message system should begin the search (next port after that specified). If we have, for example, just handled a request received on port 4, we can then call XFGST with A=4 to find out whether any requests have been received on any port. Port 4 will then be the last to be looked at by XMSG. This is called 'round-robin' scheduling of requests. The result parameter PORTNO contains the port number where the message is waiting.

If the XFWTF flag is set, then the task will go into IO-wait if no messages are waiting, otherwise a zero status will be returned, and if XFWAK is set, then the task will be 'woken up' when the next message arrives.

4.3.1.5. Disconnect (XFDCT)

```
T:=XFDCT; *MON 2XMSG
```

Closes all ports. All XMSG space belonging to the current caller is released. Special action is taken in the case of current messages, and messages waiting on the input queue (see XFSND, XFRCV and XFCLS). There is no return from driver calls to XFDCT (as the driver context is released by the call).

Note that RT-program abort and logout from background lead to automatic disconnect.

4.3.2. Manipulating Message Buffers

Message buffers are simply variable length areas which can be reserved within XMSG's address space. When assigned to a task they remain reserved for that task until it decides to release them or 'send' them to another task, at which point ownership is transferred to the receiving task so that it is able to read the data. Having read the data, the receiving task may then either release the buffers back to the pool or use them itself for storing a message to send back to the first or any other task.

Note that in many of the functions which follow, there is no parameter required to specify the message identifier (MESAD), for the reason that a current (default) message buffer is assumed, namely the last message received on the appropriate port, or, if none, the last operated on by the task. Sending or releasing a message leads to is currency being lost. The task may also change the value of the Current Message with the XFSCM function. A MESAD value of -1 implies the current message.

Messages cannot be released, read from or written to by tasks other than the current owner or whilst queued to a port. In the latter case the message must be received first.

4.3.2.1. Reserving Message Buffers (XFGET)

```
T:=XFGET (BONE XFWTF/XFWAK); A:=NBYTES; *MON 2XMSG
A=:MESAD
```

MESAD is returned to the caller for possible use in subsequent functions. The message buffer consists of a descriptor of the current owner, sender, size, length etc., and a buffer for user data. The buffer size has a maximum, system dependent size (X5MMX) defined when the XMSG system is generated.

At any particular time, the total space owned by a task cannot exceed another limit (X5MTS), which is also defined when the XMSG system is generated.

Only the current owner of a message is allowed to read or write in it, give it to someone else or release it.

Specifying a buffer length of 0 bytes implies that only a message descriptor will be reserved. Privileged tasks can then associate a physical memory area with that message descriptor by using the Define User Buffer (XFDUB) function described below.

4.3.2.2. Defining a User Buffer (XFDUB)

This is a privileged function (cf XFPRV) that allows a user to associate a physical memory buffer with a message descriptor previously obtained by XFGET with NBYTES=0. All XMSG functions then operate on that message as though the buffer space was part of the general XMSG buffer, except that XFREL (see below) only releases the message descriptor and not the buffer area.

This allows special systems or drivers to have full control over their memory allocation procedures, and to allocate, for example, messages whose buffer areas lie in a PIOC.

```
T:=XFDUB; AD:=PHYSAD; X:=NBYTES; *MON 2XMSG
```

The function acts on the current message. PHYSAD is the physical (24 bit) address of the start of the buffer, and NBYTES is its length in bytes.

4.3.2.3. Releasing Message Buffer (XFREL)

A buffer is released thus:

```
T:=XFREL; A:=MESAD; *MON 2XMSG
```

4.3.2.4. Writing into Message Buffers (XFWRI)

After building up a data buffer is its own space, a task transfers the data buffer into the current message buffer as follows:

```
T:=XFWRI; NBYTES=:D; A:=UADD; X:=DISP; *MON 2XMSG
A:=D=:NBYTES
```

If the 'whole-message-read' flag has been set (see XFREA) it is cleared, and the current message length (not the same as size) is set to 0. If DISP is -1, a value for DISP equal to the current message length is assumed instead, thus providing an append function. If the displacement is odd, 1 is added to it, and a zero byte inserted in the message. If DISP+NBYTES is greater than the message size, an error return occurs. Otherwise NBYTES bytes are copied from UADD into the message buffer. If this resulted in the message being longer than before, the current message length is set to DISP+NBYTES (rounded up if odd). NBYTES is returned to indicate the actual number of bytes transferred.

If the user has access to the buffer area directly (either because it was defined using the XFDUB function or because he has access to physical memory), he can of course do the read and writes himself. However, he must then be aware that the 'current displacement' and 'current length' information in the message descriptor will not be updated.

4.3.2.5. Writing only the Header of a Message Buffer (XFWHD)

```
T:=XFWHD; AD:=BOto3; X:=B4to5; *MON 2XMSG
```

This function inserts the A, D and X registers as the first six bytes of the current message, and increments the length parameter if necessary.

4.3.2.6. Reading from a Message Buffer (XFREA)

```
T:=XFREA; NBYTES=:D; A:=UADD; X:=DISP; *MON 2XMSG
A:=D=:NBYTES
```

The data is read from the current message starting with displacement DISP (rounded up to the next word boundary) into the user buffer specified by UADD (length NBYTES) and NBYTES is set to the actual number of bytes read. If DISP is -1, the reading of the message is resumed from the point it had reached on the previous read. If the last byte in the message is read, the 'whole-message-read' flag is set, so that the next XFWRI will reset the current message length.

NB Note that the displacement is always rounded up to the next word boundary.

4.3.2.7. Reading only the Header of a Message Buffer (XFRHD)

The first six user bytes of a message can be read using:

```
T:=XFRHD; A:=MESAD; *MON 2XMSG
```

With the six bytes being returned in the A, D and X registers (in that order!). If MESAD is not -1, the specified message becomes the current task message.

4.3.2.8. Sending Message (XFSND)

When a task wants to 'send' a message to another task, it naturally must know the 'address' or MAGNO of a port of the task. Since port numbers (and hence MAGNOs) are allocated by XMSG when the port is opened, the destination MAGNO must be obtained by an executing task via XMSG.

Initial contact is in fact made by sending a message to a dedicated task named XROUT (see services below), to first name one's port(s). Subsequently a second task may send a 'letter' via one of his ports also to XROUT, specifying a destination port by name (see XROUT Letter Service XSLET). If this name has been previous declared, XROUT will forward the message to the named port.

The first task can then use the XFMST function to extract the MAGNO of the second task and hence a direct dialogue can begin. (Note that only ports expecting letters need to have names. These will usually be ports providing services - 'server ports'.) XROUT and XFMST are described later.

In this section it is assumed the sender now knows the destination MAGNO.

A Message Buffer is 'transferred' from one task to another, thus:

```
T:=XFSND (BONE option); AD:=MAGNO; X:=PORTNO; *MON 2XMSG
```

The options are:

XFWTF - Wait flag. This is only significant when using multimachine XMSG and XFSEC (see below). If set, it implies that the caller will only be restarted (with proper status) when the message has been put into the receiver's input queue.
If not set, secure messages that cannot be delivered, will be returned as if they had been put into the destination queue and then the port been closed before the message was received by the destination task.

XFSEC - Secure message. The message will be returned to the sending port if it cannot be delivered or if the handling program terminates while the message is 'current'. Non-secure messages are discarded and released by XMSG if they cannot be delivered.

XFHIP - High priority message. It will be chained to the head of the receiver's queue instead of the tail, following any other high priority messages already queued.

XFFWD - Forwarding. The sender information in the message will not be updated, so that to the receiver it will appear that the message was sent directly from the previous sending port.

XFROU - Ignore the MAGNO parameter and send the message to the local routing task (XROUT). The message contents should be parameters to XROUT. (See section on XROUT services.)

XFBNC - Bounce message. When the receiver issues XFRCV which would have led to this message being received, it will instead be returned to the sender.

A magic number parameter of -1 (in both A and D) will direct the message back to the port from which it was last sent.

4.3.2.9. Returning a Message (XFRTN)

The user often needs to write a return status into a message and send it back to the port from which it came (e.g. replying to a transaction):

```
T:=XFRTN; DATA0=:D; A:=MESAD; X:=PORTNO; *MON 2XMSG
```

leads to DATA0 being written into the first two bytes of the message buffer, and the message buffer being returned to the port from which it was last sent. The function options are as for XFSND. (In fact the function is as XFSND, except that the D register contains two bytes of data and the A register the message address.)

4.3.2.10. Receiving Next Message (XFRCV)

When a task is ready to handle the next request, it calls XFRCV:

```
T:=XFRCV (BONE XFWTF/WFWAK); A:=PORTNO; *MON 2XMSG
A=:RPORT; A:=D=:MESAD; X=:NBYTES
```

If a message is waiting on the specified port, it is received (unchained from the message queue) and the A register contains the sending machine (high-order byte) and port number. The D-register indicates the message address, X the message length in bytes, and T the message type (see XMSG-VALUES for values):

XMTNO - Normal message.

XMROU - Message last sent by a routing program (XROUT - see below).

XMTHI - High priority (sent with XFHIP option).

XMTRE - Returned message (sent as secure but could not be delivered).

XMTPS - Pseudo message (not used at present!)

If no message is waiting, then if XFWTF is set, the task is suspended until the next message arrives, otherwise a zero status is returned and, if XFWAK is set, the next transmission to that port will lead to a 'wake up' (RT - MON 100) of the receiver task. This allows timed-out waits to be executed.

When the 'wake up' is done, the message is not received, and so the receive must be repeated. This 'wake up' option can be enabled on more than one port at a time.
A successful XFRCV leads to the returned message becoming the current message for that task (and port). If that message is 'secure', and, if the task aborts before the current message is cleared, the message will be returned to the sender with 'return' status.
The current task message is cleared by releasing/sending it to someone else, or receiving another secure message.

4.3.2.11. Receive and Read (XFRRH)

As an alternative to receive, the user can call the XFRRH function, which receives the next message in the queue (as XFRCV), and also reads the first two bytes of the message:

```
T:=XFRRH (BONE XFWTF/XFWAK); PORTNO; *MON 2XMSG
A=:RPORT:=D=:MESAD; X=:DATA0
```

Note that this is identical to the receive function, except that it returns the first two bytes of user data instead of the message length.

4.3.2.12. Message Status (XFMST)

XFMST allows one to extract the sender's magic number, and get the length and type of a received message:

```
T:=XFMST; A:=MESAD; *MON 2XMSG
AD=:MAGNO; X=:NBYTES
```

The message type is returned in the T-register. (See Receive - XFRCV above.) It might be expected that this requires an extra call, but:

a) - one often just sends a message back to its sender (XFSND with MAGNO=-1 or XFRTN) and

b) - one can read the magic number once, and after that use the machine and port information returned by XFRCV to identify the sender, whose MAGNO you now have.

4.3.2.13. Set Current Message (XFSCM)

Since many functions implicitly operate on the current message, it is useful to be able to set the latter:

```
T:=XFSCM; PORTNO=:D; A:=MESAD; *MON 2XMSG
```

Sets the specified message as task current. If the port number is >=0, the message is also set as current for the specified port.

4.3.3. Indirect Data Transfer

The amount of space available for message buffers is limited, but the user may wish to transfer large quantities of data within a machine.

He may then, instead of transfering the data into a message buffer before sending it, just send a buffer description which allows the receiver to read or write to this buffer when required.

The data area of the message can be used to include usual information (describing what the indirect buffer contains, fo example).

These functions can only be used between foreground and/or background programs running on the same machine. The transfer speed is about 3 milliseconds per kbyte (ND-100)

4.3.3.1. Define Indirect Buffer (XFDIB)

Appends to the message descriptor the information required to allow the receiver to do indirect read and write operations:

```
T:=XFDIB (BONE XFWOK); ULEN=:D; A:=UADD; X:=MESAD; *MON 2XMSG
```

The XFWOK (Write OK) flag determines whether the task to which the message is going to be sent, is allowed to write in the buffer area described by UADD and ULEN.

This definition of the indirect buffer becomes usable only after the next XFSND (without forwarding option). It then becomes invalid when the message is sent further, unless forward option is used. This prevents a user receiving a message and then defining a buffer area in the virtual space of the last sender. Closing the sending port also stops access to the indirect buffer.

4.3.3.2. Read/Write Indirect Buffer (XFRIB/XFWIB)

Allows someone who has received a message containing an indirect buffer address to read or write to that buffer. (Write only if XFWOK was specified).

```
T:=XFRIB/XFWIB; NBYTES=:D; A:=UADD; X:=DISP; *MON 2XMSG
A:=D=:NBYTES
```

The data is transferred between the local buffer area specified by the AD registers and the remote indirect buffer specified by the XFDIB function, starting with the displacement specified in X, which must be an even number. The transfer terminates, when either the count is zero, or the remote buffer is exhausted. NBYTES returned is the number of bytes not transferred.

4.4. XROUT Service Specifications

As mentioned above, XROUT is a special task that allows tasks to find each other initially by providing a port naming scheme.

It can be considered to be equivalent to the 'directory enquiries' service provided by a public telephone company, but with the following restriction

XROUT will never give you somebody else's telephone number, but will give him a message sent by you, together with your magic number. The destination task can then look at your message and ring you back, if he wants to, and thereby give you his magic number.

XROUT is implemented as a standard foreground-program that runs on a demand segment, and so is relatively independent of the XMSG communication system as such.

Tasks communicate with XROUT by sending it messages using the XMSG function XFSND with option XFROU (instead of a MAGNO).

4.4.1. XROUT Message Formats

The messages that users send to XROUT have a standard format:

Byte 0 - a serial number returned unchanged by XROUT in order to allow users who may have many requests outstanding, to recognise particular replies. Note that messages sent from XROUT also return a special message type (XMROU) in the T-register as a result from a receive call (XFRCV or XFRRH), so that they can be distinguished from messages originating from other tasks.

In order to comply with the ND standard message format, the high order bit of byte 0 should be zero.

Byte 1 - the service number (symbol XSxxx) of the service being requested. XROUT overwrites this with the return status from the request: 0 is OK, whilst other values are errors as defined by the XR... symbols (cf. XMSG-VALUES - e.g. XRISN=1 - illegal service number). Note: XROUT service values and result/error codes are always in the range 0..127, so that the user may set the high-order bit (bit 7) to indicate user services and/or result statuses. The routine XMRER in XMSG-LIBRARY takes as input a routing error code in the A-register and returns a pointer in the A-register, to an error message containing ASCII characters (terminated by a quote character).

Byte 2-3 - length of remainder of message in bytes. Followed by a sequence of parameter blocks.

Each parameter block has the form:

Byte 0 - Parameter number and type (0 means skip this byte to allow for fill). Integers have positive values, strings negative (two's complement of parameter number).

Byte 1 - Length of parameter in bytes.

Byte 2 ... Parameter data.

The number and type of parameters is dependent on the particular service. All parameter blocks must start on even byte boundaries in the message (use zero fill). NPL routines are provided to allow the building of service messages in a user buffer, which can then be written to a message buffer (XFWRI) and sent (XFSND). The message sent to XROUT must be big enough for the reply, if the latter is longer than the request.

The parameters in sections 4.4.1.1 and 4.4.1.2 are used.

4.4.1.1. Integer

Since messages will be sent between machines with different word lengths, integers will be treated as signed, so that the sign bit will be extended if necessary. This allows the sender to decide how much space he wishes to use in the message for an integer, and the user to take appropriate action when receiving.

4.4.1.2. ASCII Strings

The length is defined by the parameter length. If a fixed length record is required, the record will be filled up with blanks.

4.4.2. Services in Detail

The following list (sections 4.4.2.1 through 4.4.2.18) of services is tentative, and will most certainly be extended later. The symbolic names XS... are defined in the XMSG-VALUES file.

4.4.2.1. Name a Port (XSNAM)

In order for a port be named, this name must be declared to XROUT. This is done by the XSNAM service with the following parameters:

1 - Identifier (type string)

If an open port already has the specified name, an error status is returned.
The sending port is then given the specified name. If it previously already had a name, the port is renamed with the new name.
The maximum name length accepted can be defined when the XMSG system is generated (symbol X5NLW in XMSG-SYS-DEF, default=32 bytes).

4.4.2.2. Create Connection Port (XSCRS)

Parameters:

1 - Identifier (type string)
2 - Max no of connections accepted (type integer)
3 - Uniqueness flag (type integer)

This service is very similar to XSNAM, but allows XROUT to control the number of users that a port can handle simultaneously, and even distribute users among servers.
XROUT first handles the message like an XSNAM service request, except that unless the uniqueness parameter is specified and is non-zero, connect ports are allowed to have identical names. It then sets a counter (the free connection counter) associated with that port to the value specified in parameter 2. For remainder of specification, see Letter service (XSLET) below.

4.4.2.3. Increment Free Connection Count (XSNSP)

Parameter:

1 - No of new free connections (type integer)

After opening a connection port (see XSCRS above), a task can later increment the free connection count (when connections become available) by using the XSNSP service.

4.4.2.4. Send Letter (XSLET)

This service is used to contact a remote port. The parameters used by XROUT is:

1 - Port or Connection Name (type string)
2 - Machine name (type string)

If parameter 2 is specified, XROUT looks this up in the name table, and if this has been defined as a remote name (see define remote name below), forwards the letter to the XROUT in the specified machine.

Otherwise XROUT extracts the identifier and looks up the string in its name table. If a match is found, XROUT looks at the port type. If this is a normal named port (named using the XSNAM service), the whole message is forwarded (function XFFWD) to the matching magic number.

If it is a connect port (named using XSCRS), XROUT looks at the free connect count and if greater than zero, it decrements it and forwards the letter. If not it tries to find another port with the same name. If no match is found, the function code is set to an error value, and the message returned to the sender.

The remainder of the message, can contain data for the receiving task (user name, password, ...) to allow the server to check that the sender is entitled to use that service, before replying to him and thereby giving the caller his magic number. If the server wants to reply to the requester without giving away his own magic number, he should reply with the forward option (XFFWD).

4.4.2.5. Return a null status message (XSNUL)

XROUT returns a message of two bytes containing the reference number and 0 (used for testing/benchmarking).

4.4.2.6. Get Name from Magic Number (XSGNM)

Any XMSG user can obtain the name of a given port by sending a message containing the magic number (integer) as parameter 1. The return message will contain the port name appended as parameter 2 (type string), if there was space in the message (make sure there is enough!)

4.4.2.7. Get Name of Port from Port Number (XSGNI)

Privileged callers may obtain the name of a given port by sending a message containing the machine number and the port number as parameter 1. XROUT will return the name of the port with the least machine number/port number greater than or equal to the input parameter. The port's machine number and port number are returned as parameter 1 and the name as parameter 2. If no port was found satisfying the above conditions, the first parameter is zero.

4.4.2.8. Clear name (XSCNM)

When a name's validity has expired, the clear name service can be used to remove the specified name from the name table.

Name clearing is also done automatically by XROUT when it notices that a port has been closed.

Other machines that have fetched the current port's magic number (see XSREM below) are also informed when the port is closed.

4.4.2.9. Find Remote Name (XSREM)

In multi-machine XMSG configurations, a local XROUT must be told explicitly to go and find a remote name. A user process does this by sending an XSREM request to his local XROUT, who then contacts the remote XROUT to ask for the target port's magic number, which it then enters in its own tables. Note that the magic number is not returned to the user.

This 'Find Remote Name' request only needs to be done once - at system initialisation, but it must be done after the remote process has opened its port and named it locally using the XSNAM service described above.

Parameters:
1 - Local name (string)
2 - Remote name (string)
3 - Remote machine number (integer)

The local name is the name by which the port can be referred to (after completion of the XSREM request) locally, whilst the remote name is that which will be sent to the remote XROUT in order to fetch the magic number, and so should be the name used by the remote task when naming its port. A user may normally set the local name the same as the remote name, but cannot, where this would cause a name conflict.

4.4.2.10. Get Magic Number (XSGMG)

This is a privileged service, which is used between XROUTs in order to implement the XSREM service described above. It returns the magic number for a particular name, and implies that the remote XROUT should inform the calling XROUT when the port is closed (using the XSCMG service described below). Each XROUT remembers which remote XROUTs have got each port's magic number using a bit map of 256 bits which is part of each name record.

Parameters:
1 - Name (string) Result:
1 - Magic Number (integer)

4.4.2.11. Clear Magic Number (XSCMG)

This is another privileged service used for XROUT to XROUT communication and implies that the specified magic number is no longer valid. It is sent by an XROUT when it becomes aware that a local port has been closed to all XROUTs who have obtained that port's magic number by using the XSGMG service described above.

Parameters:
1 - Magic number (integer)

4.4.2.12. Define Remote Name (XSDRN)

XSDRN is used for defining the names of machines (specified as parameter 2 in letters - XSLET service). XSDRN is normally accessed via the Define-Remote-Name command of the XMSG-COMMAND background program. XSDRN is privileged.

Two parameters are specified:

1 - Machine name (type integer)
2 - Machine no

The specified name is put into the name table (must be unique) and all letters that are addressed to that machine (parameter 2 in XSLET) will be forwarded to the specified machine. Note that a machine can have many names, so names should be used to identify functional machines rather than physical machines whenever possible (e.g. SIBAS-BACKEND or MAIL-HANDLER rather than ND-100-377.)

If the second parameter is not specified, the name is cleared.

4.4.2.13. Define Machine Routing (XSDMC)

Whereas Define Remote Name (XSDRN) defines a mapping of a machine name to an XMSG machine number, Define Machine Routing (XSDMC) specifies how to get to that machine. This is not necessary if the machine is directly connected, since XROUT will find out when the link to that machine starts up, but is necessary (at present) for both the local machine and machines connected via neighbours. The XSDCL service (which is privileged) takes two parameters:

1 - Machine Number
2 - To be routed via this machine (or 0)

It leads to the routing tables being updated such that the specified machine is marked as being available via the machine defined in parameter 2. If the second parameter is zero, the cluster is marked as 'not available.'

If no parameter 2 is specified, the machine number is defined as the number of the local machine. This operation must be done before any access to multimachine XMSG can be made.

4.4.2.14. Get Routing Information for a Machine (XSGMC)

XMSG-COMMAND allows one to list the routing information held by any accessible XROUT in an XMSG network. This is done by sending XSGMC messages with one integer parameter, the object machine number. XROUT replies with a message containing one double word parameter which should be interpreted as 4 bytes.

The most significant byte is the machine number as requested by the user, or the next higher known machine number if that machine number is unknown. If there are no machine numbers greater than or equal to the input parameter, the byte is zero.

The next byte contains zero.

The next two bytes contains the connection type and address:

Byte 2 - Connection Type	Byte 3 contains:
0 - Unavailable	
1 - Neighbour	Link Index
2 - Via	Machine number
4 - Local	

4.4.2.15. Starting up / Stopping a Multi-Machine Link - XSLKI

This privileged service is used by the START-LINK and STOP-LINK commands in XMSG-COMMAND. It is used when one wants to use an HDLC link (which must have been declared in generation of SINTRAN) as a multi-machine link. The XSLKI request requires three parameters:

1 - HDLC link logical unit number
2 - Timeout value in Basic Time Units (BTU).
3 - Number of frames to allocate (window+1)

XROUT will reserve the HDLC link (both input and output data fields), check that there are enough free frame buffers and then initialise the interface. The link will then go into the 'calling' state, which means that it tries to make contact with the adjacent machine, by sending a predefined maximum number of SABM frames. (SABM= Set Asynchronous Balance Mode.)

If the number of frames to allocate (parameter 3) is zero, a 'close' link operation is performed instead; the link is disabled, released, and the routing information updated.

4.4.2.16. Trace Initialise - XSTIN

The service is used by the OPEN-TRACE command in XMSG-COMMAND to initialise the trace system (see description under OPEN-TRACE). It takes as a parameter, the file name of the trace file. XROUT then opens and initialises the file, resets the trace infomation and starts up the trace dump foreground program (XTRACE). Systems 0 and 1 (Clock and Trace Management) are automatically enabled.

4.4.2.17. Trace Close - XSTCL

Tis is the opposite to XSTIN (above) and leads to the last block(s) being written, the file closed and XTRACE aborted.

4.4.2.18. Define Trace Conditions - XSDTC

This takes as a parameter, an integer. If it is positive, the system with that number is enabled for tracing, if it is negative then the system is disabled.

4.4.3. NPL Routines for Message Formatting

The file XMSG-LIBRARY:BRF contains a set of NPL routines which can be useful for formatting messages to XROUT. They all act on a local user buffer which is always pointed at by the X-register. The B-register is always preserved over a call from any of these NPL routines. They all give a skip return if OK, non-skip if buffer full.

4.4.3.1. XBINI - Initialise Buffer

This initialises the specified buffer:

X=Buffer Address
A=Buffer length in bytes (4 bytes used by header)

4.4.3.2. XBAST XBARC - Append String Append Record

Appends the specified string (terminated by quote character) or record (of specified length) to the current message:

X=Buffer Address
A=String address (terminated by ' if XBAST)
D=Number of bytes (for XBARC)
T=Parameter number (complemented by XBAST)

4.4.3.3. XBAIN, XBADB - Append Integer

These routines append an integer as next parameter in the message. XBAIN always appends a 16-bit value, whereas XBADB takes as input a 32-bit value, which it puts into the message as 16-bits if (and only if) this is valid:

X=Buffer Address
A/AD=Value (XBAIN/XBADB)
T=Parameter number

4.4.3.4. XBLOC - Locate Parameter

Since the parameters can be put into a message in a random order, it is useful to have a routine that can find a specified parameter:

X=Buffer Address
T=Parameter Number (always positive)

Result: A=Start of parameter block

4.5. System Function Specifications

All system functions are privileged.

These system functions are mainly used by XMSG command program (background) to enable the user to find out what the message system is doing. They should not normally be called by users but are included here for reasons of completeness.

4.5.1. XFPRV - Make Calling Task Privileged

Most system functions, as well as some user functions (e.g. Define User Buffer - XFDUB) can only be executed by privileged tasks. In order to become privileged (for XMSG), a task must successfully execute the XFPRV function.

In order to do this the caller must be either a driver, direct task, foreground program on ring 2,3 or background program logged in as user system. Besides this, the program must also specify the current XMSG password (XPASW) which is defined in the XMSG-POFTABS file in the A-register:

```
T:=XFPRV; XPASW; *MON 2XMSG
```

When the task wants to stop being privileged, the same call should be used, but with the A-register equal to zero.

The reason for specifying the XMSG password is to ensure that privileged programs that base themselves on accessing XMSG table structures have been updated to the current XMSG table definitions (in XMSG-POFTABS).

4.5.2. XFSIN - Initialise for System Functions

This returns the basefield address of the message system in POF, which is needed in order to be able to access XMSG tables using the system functions.

```
T:=XFSIN; *MON 2XMSG
A=:BASEADD
```

4.5.3. XFABR - Absolute Read from POF

This function allows a program to read a block of data from POF into his user area. This function can be executed even when the message system is locked (see XFMLK, XFMUL).

```
T:=XFABR; ULEN=:D; A:=UADD; X:=ABADD; *MON 2XMSG
```

4.5.4. XFABW - Absolute Write to POF

This function is simited when the message system is locked.

T:=XFABW; ULEN=:D; A:=UADD; X:=ABADted when the message system is locked.

```
T:=XFABW; ULEN=:D; A:=UADD; X:=ABADD; *MON 2XMSG
```

4.5.5. XFMLK - Message System Lock

This function locks the message system, so that all requests will be refused until an unlock is done. This allows consistent modification of tables to be done using XFABR and XFABW.

```
T:=XFMLK; *MON 2XMSG
```

4.5.6. XFMUL - Message System UnLock

Inverse function to XFMLK:

```
T:=XFMUL; *MON 2XMSG
```

4.5.7. XFM2P - Convert Magic Number to Port and Machine Number

```
T:=XFM2P; AD:=MAGNO; *MON 2XMSG
A=:PORT; A:=D=:MCNO
```

4.5.8. XFP2M - Convert Port Number to Magic Number

```
T:=XFP2M; A:=PORT; *MON 2XMSG
AD=:MAGNO
```

4.5.9. XFCRD - Create Driver

This function is used to create a driver with a given context (see section 4.7 for details on calling XMSG from drivers):

```
T:=XFCRD (BONE XFPON); UADD=:D; A:=ILEV; *MON 2XMSG
A=:TASKADD;
```

The ILEV parameter contains the interrupt level that the driver should run on. XFPON should be set if paging should be on when the driver is started (PIT3 assumed), and UADD points to an 8 word buffer.

The buffer contains the register block that the driver will be started with, in the order required for the Load Register Block (LRB) hardware instruction (cf. NORD-100 Reference Manual - ND-06.014.01).

XFCRD allocates an XT-block to the driver and returns its address in the A-register.

4.5.10. XFSTD - Start Driver

Starts an already created driver:

```
T:=XFSTD; TASKADD; *MON 2XMSG
```

XFSTD overwrites the driver's L-register with his XT-block address before starting the driver.

In this way a driver started will have the L-register containing its XT-block address. The driver must be set back again before calling XMSG - see section 4.7 on calling XMSG from drivers.

XFSTD does not set the appropriate bit in the PIE, or load or fix any segments.

This should be done using FIXC and ENTSG - see SINTRAN III Reference Manual - ND-60.128.01.

4.6. The XMSG-COMMAND Background Program

XMSG-COMMAND is a background program that is used to control and supervise the XMSG system.

It can also be used for XMSG testing and benchmarking.

XMSG-COMMAND accepts commands in the usual SINTRAN way, with abbreviations being allowed, and prompts for parameters that are not specified on the command line.

A MODE command allows a file of commands to be executed instead of having to type them in.

Any command line preceded with the @ sign will be handed over to the SINTRAN III background command processor for execution.

Many of the commands in the background program use privileged XMSG functions.

XMSG-COMMAND will automatically use the XFPRV function to make itself privileged when some of these functions are invoked, but will be refused if the user is not logged in as 'SYSTEM.'

4.6.1. Output Formatting

Since the command program is mainly a formatting program for XMSG tables and trace elements, a generalised formatting facility has been implemented (QFORM). This is similar to FORTRAN formatted output, but extended to be able to handle record, table and list structures. It also allows arithmetic operations to be executed by the format strings and for formats to call each other - like subroutines.

This is not the place for a complete description of QFORM, but a brief overview of the commands in XMSG-COMMAND that are associated with these formats follows:

4.6.1.1. LIST-FORMATS

This lists the formats matching the specified parameter on the terminal.

4.6.1.2. FETCH-FORMAT

Fetches the specified format into the format edit buffer.

4.6.1.3. EDIT-FORMAT

Allows editing of the format currently in the edit buffer. All normal S-III edit control characters are allowed. Note that the format is NOT implicitly saved after editing, so this must be done explicitly using the following command:

4.6.1.4. SAVE-FORMAT

Saves the current edit buffer contents as a format with the specified name. If no name is specified, the last name specified will be used.

4.6.1.5. DUMP-FORMATS

Allows all formats matching the specified parameter to be dumped onto the specified file. The file written is such that it can be executed using the MODE command (see below) and to have all the formats reloaded. This makes it easy for users to build up their own format library.

4.6.2. Commands that List XMSG Tables:

The following commands list XMSG tables. They all use the privileged function XFABR in order to read the tables from the Paging Off area.

4.6.2.1. List-Tasks

This command gives a list of tasks using the local XMSG system.

It prompts for a 'Task Address?'. If an XT-block address is specified, only that task is listed. If 0 is specified, all tasks are listed. If nothing is specified, the command repeats what was done last time.

Information given for each task is:

ADDRESS	- Address of XT-block (like RT-block)
RT/DR	- RT name if RT-program, else *DRIVER*
PROCESS	- RT address or interrupt level and paging status
IOW	- Y if task is in I/O wait.
GWT	- Y if task is in a global wait on all ports
PRV	- Y if task is privileged
MEM USED	- Number of bytes currently owned by this task
LIMIT	- Max number of bytes that can be owned by this task
PORT HD	- Port number of last port opened (start of chain)
CURMES	- Address of current task message

4.6.2.2. List-Ports

This command can list either one port, or all ports, or all ports owned by a task. The first prompt asks whether a task port chain is to be followed, and if so, the second parameter asks for the task address. If not, a port number can be given, or all ports requested.

The information listed for each port is:

NO	- The port number (index in port table)
ADDRESS	- Address of port descriptor in physical memory
OWNER TASK	- Task address of owner
QHEAD	- Address of first message in input queue
QLEN	- Number of messages in input queue
CHAIN	- Port number of next port belonging to same task
WAK	- Task will be woken if a message arrives on this port
PRM	- 1 if port was opened with permanent option
IOW	- 1 if the task is in I/O wait on this port
CURMES	- Address of current port message

4.6.2.3. List-Messages

List information from the message descriptors. As for ports, list either a single message, or all messages, or all messages on the input queue to a port, depending on the reply to the 'Record Address? ' prompt.

The information listed for each message is:

ADDRESS	- Address of message descriptor in physical memory
OWNER	- Task address of owner and name if RT-program
FROM-PORT	- Machine number and port index of last sender
LENGTH	- Current used message length in bytes
BUFFER	- Physical address of start of buffer area
SIZE	- Buffer size in bytes
R	- 1 - Implies message is being returned.
S	- 1 - Message is a secure message
C	- 1 - Message is chained to a queue

4.6.2.4. List-Names

This command asks an XROUT to dump out its name table, listing the machine and port numbers for each name in XROUT's tables.

In a multi-machine configuration, the command allows access to any XROUT, so the first prompt asks for the machine number where the XROUT program is to be found. (Default is local.)

4.6.2.5. List-Routing-Info

As with List-Names, this command accesses an XROUT lying in the machine specified by the reply to the first prompt. The command lists out the information that the target XROUT has about how to access each machine known to it. The information specified for each machine is (see section 3.6.3, modifying the routing tables, for details):

MCNO	- the machine's number
CONNECTION	- Connection type:
UNAVAILABLE	
VIA MACHINE	- Access is via machine ....
VIA LINK	- Access is via (running) link ....
PIOC NUMBER	- Machine is local PIOC ....
MACHINE/LINK	- Specifies the .... field according to connection type above

4.6.2.6. List-Links

This command is used to list the current status of all HDLC links being used by XMSG (i.e. those that have been started by the START-LINK command. The information listed for each link is:

NO - link index in XMSG
ADDRESS - in POF of link block (XL-block)
STATE - of connection to adjacent machine. Values are:
DEAD - crashed (fatal timeout or hardware error)
INIT - being initialised (purely internal)
CALL - trying to make contact with neighbour
CONN - contact made.
RUN - data phase.
MCNO - of neighbour (CONN and RUN states only!)
ANY-AC - last HDLC A/C bytes received at all
-RUN - A/C bytes from last RR or INFO frame
LUN - SINTRAN Logical Unit No (Octal)
HDLCST - Transmitter Driver status, if DEAD (octal)
HDLCHW - HDLC Hardware status
TIMEOUT - Timeout value in BTU´s

4.6.2.7. List-Frames

This command always list all the frames in the Multi-machine XMSG frame pool. Information listed for each frame is:

NO - Frame index in pool
ADDRESS - in POF of frame´s XD-block
OWNER - address of link that owns it
HANDLER - address of link currently handling it
HAC - HDLC A/C bytes (HEX)
NETAD - Destination/Source mc numbers (HEX)
TRTYP - Transport type
CNT1 - Number of bytes in the first data part
BUFFER1 - Address of the first data part
CNT2 - Number of bytes in the second data part
BUFFER2 - Address of the second data part

4.6.2.8. List-Command-Prog-Variables

Lists the current values of variables that are internal to the command -program - e.g. buffer addresses, lengths, sizes, as well as the values for the current task, port and message.

4.6.2.9. Dump-Memory

Is an unstructured dump of physical memory in bank 0. It takes as parameters the first and last address of the area to be dumped (in octal).

4.6.3. Modifying the Routing Tables and Controlling Links

In a multi-machine XMSG configuration, we need to be able to build tables that specify for each local XMSG where it should send messages destined for a particular machine. This information lies in the routing tables. See section 4.2. for a definition of the terms 'machine' and 'link'.

Each machine contains an XROUT program, which in turn contains three routing tables:

The Machine Location Table (MLT) is almost the same for every machine. It contains a word for each machine (indexed by machine number) which indicates either the machine via which this machine can be accessed or the link index if directly connected. The 'via' operation can be repeated.

When a link is established, the machines at each end of the link exchange identifiers and update their routing tables accordingly. Otherwise all modification of routing tables must be done manually using the commands described below. (Note that the commands allow one to execute the command in any machine to which one has access - 'XROUT machine?' prompt.)

4.6.3.1. Define-Local-Machine

This command must be used before multi-machine XMSG is available and cannot be repeated. It simply informs XMSG of the local XMSG machine no.

4.6.3.2. Define-Machine-Route

Updates the machine location table in the specified machine (parameter 1 can be accessed via parameter 2).

4.6.3.3. Start-Link/Stop Link

These commands provide a direct interface to the XROUT Start Link Service: see section 4.4 for further details.

4.6.4. Commands for Debugging Systems that use XMSG

When debugging complex systems made up of many communicating programs, one may want to take dumps of the XMSG tables every now and then on a file, and then look at the file afterwards to see how the situation evolved. XMSG provides two ways of doing this:

1 - Snapshots - Implies writing a copy of XMSG tables to disk, either at the beginning of a file, or appended to the current contents, so that snapshots can follow each other. Another command allows the snapshots to be fetched, and then one can use all the usual commands on the snapshot instead of on the actual situation. Since all RT-names are also included as part of the snapshot, this allows the analysis to be done on another machine. If you have problems using XMSG, it is strongly recommended that snapshots are taken in a file which can be sent to Norsk Data on a diskette.

2 - Tracing. A trace facility has been incorporated into XMSG. Routines in XMSG place trace elements (of variable length) in buffers, which when full, are written out on to a contiguous (wrap-around) disk file by an RT-program called XTRACE. This file can then be inspected using the trace manipulation routines described below. Tracing can only be done on ND-100´s (not NORD-10´s.)

4.6.4.1. SAVE-POF and FETCH-POF Commands

SAVE-POF takes a snap-shot of the current POF tables and writes them out on to a file, together with all RT-program names. (The latter takes a little time - so do not panic!) The snap-shot can either be put at the beginning of the output file, or following previous snap-shots. Default file type is :XPOF.

FETCH-POF reads a snap-shot as specified by the user from a file previously written using the SAVE-POF command. This copy of POF is then locked in XMSG-COMMAND´s logical space, either until another snap-shot is fetched, or until the UNLOCK-POF command is invoked. Note that the DUMP-MEMORY command does not access the snap-shot, but tries to access the real physical memory.

4.6.4.2. TRACE Generation Commands

The following commands control the way in which the trace is made. They should therefore be used whilst the test system is being run. Note that tracing does take a certain amount of time (partly writing the information to disk) - preliminary estimates - about 50 microsecs/word traced, on a slow ND-100.

In order to include the trace facilities, XMSG must be generated with the trace option 8TRAC. This leads to the required calls being assembled in, as well as inclusion of the handling routines and buffer space. The RT-program XTRACE is always included.

The format of the trace file is simple. The file comprises blocks of 512 words. Block 0 is the header block and contains the following information:

Word (oct)	Contains
0	0
1	123456 octal
2-3	MTIME (basic clock time) when was file opened (OPEN-TRACE)
4-12	CLOCK (date, time) when file was opened (OPEN-TRACE)
13	XMSG password to check version information
400-777	Copy of XMSG basefield

The remaining blocks contain the trace information itself. Each block has a two word header containing the logical block number (incremented by 1 for each block written - skipping 0 - indicates that the block has not yet been used) and the number of trace calls missed since the last block was written (due to all buffers being full). The remainder of the block comprises trace elements following immediately after each other.

Each trace element has a header - which is of interest to the trace system, and a body, which of interest to the user of the trace system. The header comprises one word: the left hand byte is the system number (see ENABLE-TRACE command) and the right hand byte contains the number of words in the body.

A header that is zero terminates the block.

The following system numbers have been allocated:

System (dec.)

0 : Clock. Only output when necessary. Body (2 words) contains ATIME (in basic time units.)
Trace management. First word of body contains the function
1: open
2: close
3: enable/disable (next word contains the system number - negative means disable)

8 : XMSG Calls. The 5-word body contains the T-,A- and D-regs, the XT-block address and the X-reg from the caller.

9 : XMSG return to user. Body is as for system 8, but with result registers instead.

10 : XMSG kernel context switch - traces queue and element address.

11 : XMSG Link Layer frame received.

12 : XMSG Link layer - bad frame received and ignored.

13 : XMSG Link Layer send frame. Trace body: AC bytes, length, etc.

14 : Network Layer - Complete datagram queued to receiver queue.

15 : Network Layer - Datagram fragment received.

16 : Network layer - Any frame received (inc. route thru')

4.6.4.2.1. OPEN-TRACE

Open the trace file and initialise the trace system. The header block is written and the first word in all other blocks is set to zero to indicate unused. Systems 0 and 1 are automatically enabled (clock and trace management).

Note that since XROUT actually initialises the trace system it is possible to start a trace in any machine that one may access. XMSG-COMMAND only sends a start-trace message to the specified XROUT and waits for the reply.

N.B. The trace file should lie on user RT, since some Sintrans have a bug preventing direct mode access to files lying under other users. It must also have been previously created as a contiguous file (cf. SINTRAN REFERENCE MANUAL ND 60.128.02).

4.6.4.2.2. ENABLE-TRACE

Specifies which systems are to be traced. Only one system can be enabled by each call of the ENABLE-TRACE command, but many systems can of course be enabled at the same time.

4.6.4.2.3. DISABLE-TRACE

Has the opposite effect to ENABLE-TRACE.

4.6.4.2.4. CLOSE-TRACE

Leads to all traces being disabled, the remaining blocks written to the trace file, and the trace file is closed.

4.6.4.3. Commands for Dumping a Trace File

The following commands allow 'post-mortem' analysis of a trace file created using the above commands. Together with the SAVE and FETCH-POF commands they allow trace analysis to be performed on a machine, other than the one on which the trace was made. (Send trace files to Software Service department Norsk Data, if problems are encountered.

4.6.4.3.1. DUMP-TRACE-OPEN/DUMP-TRACE-CLOSE

Opens (or closes) the trace file for dumping. This is done by the command program itself, so (RT) should be specified if necessary. The command opens the file and checks its format.

4.6.4.3.2. NEXT-TRACE and PREVIOUS-TRACE

These take as parameters the number of trace elements to be dumped and lead to the trace elements being read and formatted. If the parameter zero is specified the command will skip either to the first element (PREVIOUS 0) or last element (NEXT 0) in the current trace. The trace elements are (at present) always output on the terminal. The time that is specified is the time in seconds and hundredths of a second between the trace file being opened and the generation of the trace element. Resolution is in Basic Time Units (20 msecs).

Each element is formatted by calling the QFORM format whose name is FXZEnnn where nnn is the decimal representation of the system number for the system that caused the trace element to be generated. For example, XMSG-call trace elements are formatted by FXZE008. Try the command 'LIST-FORMAT FXZ' and see!

4.6.5. Commands that act like normal XMSG Function Calls

The following commands provide an interactive way of executing XMSG calls without having to write a program to do it. XMSG-COMMAND just collects the parameters and executes the appropriate monitor call(s).

An asterisk (*) indicates that the command also asks for a count of how many times it should repeat the operation. Default is once, and if any other value is specified, XMSG-COMMAND will take the start and stop (real) times and calculate the time taken per loop in order to facilitate benchmarking.

XMSG Function	XMSG-COMMAND command
XFDCT	Disconnect
XFOPN	Open-port
XFCLS	Close-port
XFGET	Get-message-space
XFREL	Release-message-space
XFREA	* Read-direct
XFWRI	* Write-direct
XFSND	Send-message
XFSND (XFROU)	Route-message
XFRCV	Receive-message
XFMST	Message-status
XFSCM	Set-current-message
XFGST	Wait-general
XFDIB	Define-indirect-buffer
XFWIB	* Write-indirect
XFRIB	* Read-indirect
XFDUM	* Dummy-loop
XFRCV & XFSND	* Loop-receive-reply (echo)
XFPRV (XPASW)	Set-privileged
XFPRV (0)	Clear-privileged
XFCRD	Create-driver
XFSTD	Start-driver
XFMLK	Lock-message-system
XFMUL	Unlock-message-system
XFSIN	Initialise-service-system (automatically)

XMSG-COMMAND allows default values to be specified for many of these parameters.

A list of the current defaults held by XMSG-COMMAND can be obtained using the 'list-command-prog-variables' command described in section 3.6.2.8. Some commands are more complex, providing a sequence of XMSG calls:

Ask-Route expects the user to have set up data for a message in the output buffer using the commands 'clear-buffer', 'append-integer' and 'append-string' described in sections 3.6.6.3 - 3.6.6.6.

'Ask-route' then does the 'buffer-ready' action, writes the buffer into a message (which must previously have been got), opens a port if none is open and sends the message to the XROUT task in the specified machine.
It then waits for the reply, reads it into the input buffer and decodes it, (see 'decode-buffer' section 3.6.6.7).

Remote-Loop is more complex, and expects two other special programs to be running (echo-slave and echo-master). It is used for more generalised benchmarking and testing.

4.6.6. Commands affecting Buffers in XMSG-COMMAND

XMSG-COMMAND has two local buffers - one for input and one for output which can be used to build up data, which can then be written into a message and sent using the above commands (see 'ask-route', section 3.6.5, in particular), so that one can interactively build and send messages to programs that are being tested. The following commands have been implemented to access these buffers:

4.6.6.1. List-buffer

Lists the addresses, sizes and current lengths of the two buffers, as well as their contents as ASCII strings. (See also 'decode-buffer'.)

4.6.6.2. Fill-output-buffer

This is the simplest command for putting data in the output buffer - either a sequence generate by XMSG-COMMAND or typed in as response to a prompt.

The SINTRAN command LOOK-AT MEMORY can also be used to put information directly into the buffer, but the length should be preset using the fill-output-buffer command, as this length is used as default when writing the data into the XMSG message (write-direct command).

4.6.6.3. Clear-buffer

This command presets the output buffer for repeated use of the next three commands, which allow the building up of a message conforming to the XMSG standard, expected by XROUT (see beginning of section 3.4 on XROUT). Obviously, if this format is not being used, then these commands (in fact most of the remainder of this section) will not be relevant.

4.6.6.4. Append-integer

Appends an integer parameter to the current output buffer in the standard format. Note that this only has an effect in the local XMSG-COMMAND output buffer.

4.6.6.5. Append-string

As above, but appends a parameter of type string.

4.6.6.6. Buffer-ready

When all the parameters have been appended, the serial number and service number need to be put in (first two bytes in message). This is done by the Buffer-ready command, which does not copy the local buffer into the message.

A typical example of the building up of a buffer for sending to XROUT (or any other task understanding the message standard) would be:

```
*clear-buffer
*append-string
  Parameter no? 1
  String? KLODDY
*buffer-ready
  Service no? 3
  Reference no? 1
*
```

Note that this just prepares the buffer locally. It must then be written into a message and sent, using the commands 'write-direct' and 'send-message', or the combined command 'ask-route' if the message is to be sent to a routing program.

4.6.6.7. Decode-buffer

Having put things into these buffers, either using the above commands, or by reading messages that have been received, they can be dumped in an intelligible format (if they are in the standard format) by using the 'decode-buffer' command. This can decode either the output or input buffer.

4.6.6.8. Generate-, Check-Pattern

One can also use the remainder of XMSG-COMMAND's logical memory space (above 100000 octal) to generate and check patterns which have been written to, and read from, other areas using the read/write-direct/indirect commands. Generate and check for linear sequences of bytes of lengths up to 32 k, with user definable start value and increment.

4.6.7. Miscellaneous Commands

4.6.7.1. Mode

This command asks for the name of a file from which it will read all further input until it meets an end of file. Mode commands can be nested to a maximum depth of 8.

The mode file allows the system configuration for a system to be kept as a source file, ('define-machine-location' and 'define-cluster-routing-info' commands) which is run when the system is started up.

4.6.7.2. Set-port

Allows the default port number to be set.

4.6.7.3. Get-error-message

Takes as input an XMSG error code (<0) or crash code (>0) and returns an explanatory text (uses the XMERR routine in XMSG-LIBRARY).

4.6.7.4. Debugprint-on/-off

These commands control the debugprint flag. When this is on, XMSG-COMMAND writes out the register contents before and after every XMSG monitor call it makes.

This can be useful for learning how XMSG works. For example by switching debug print on and using the 'list-names' command, one finds out how one communicates with XROUT.

4.6.7.5. Monitorcall-on/-off

These commands control the monitorcall flag. When this is off, XMSG-COMMAND skips all XMSG monitor calls. (For debugging).

4.6.7.6. Help

List commands matching the parameter specified.

4.6.7.7. Disconnect

Executes the disconnect function (XFDCT) followed by an exit (MON 0).

4.6.7.8. Exit

Does an exit (MON 0) without disconnecting.

4.7. Calls from Drivers/Direct Task

Since there is no RT-description for drivers and direct tasks, the message system uses an XT-block (which is a data area it reserves for each task) for saving the current context. When a driver/direct task calls the message system, the L-register must contain the address of this block. If none has so far been allocated, then L must be zero, leading to a new block being automatically allocated, and its address returned in L.

When XMSG restarts a driver after executing an XMSG call, it starts it with a skip return. This allows the driver to have a jump to a wait routine directly following the XMSG call. (Since XMSG runs on interrupt level 5, it will not be able to execute the function before the driver/direct task has done a WAIT.) If the driver is restarted by XMSG, it will continue at the skip address (with the corresponding context), but if triggered by an interrupt, it will continue after the WAIT, or call to WTxx (but must not call XMSG until a return has been made from the previous function call). No restart is done by XMSG after a call to the disconnect function (XFDCT).

If the XMSG 'create-driver' function (XFCRD) is used, it allocates an XT-block before starting the driver and hands this over to the driver in its L-register.
Note that this makes it extremely easy to write drivers handling identical devices with each driver having its own full context, since it is only necessary to have a word in the datafield where the XT-block address is saved. Each driver can then wait for messages as a separate task owning its own port(s).
The XMSG monitor call is special in that it can be called directly from a driver or direct task in almost the same way as from an RT-program (see example in appendix C).

4.8. Error Handling

XMSG spends a lot of its time checking itself, since inconsistencies in the POF tables could easily lead to the destruction of the operating system. If one of these checks fails (which could be due to somebody else overwriting XMSG tables or XMSG itself going wrong!), XMSG will close itself down, return a bad status (XECRA) to the user, and print an error message on the SINTRAN error device (error number 46 or 47). If this occurs, a dump of the XMSG basefield should be taken to find out what has happened.
In the normal course of events, the user will probably have made a mistake, and so will get an error status returned in his T-register. The file XMSG-VALUES provides a symbolic name and description for each of these error messages, which are also listed at the end of this manual.
The routine XMERR in XMSG-LIBRARY converts an XMSG error code in the A-register to a pointer to an explanatory text returned in the A-register. The text is ASCII terminated by a quote character(').

4.9. Loading Instructions

This section describes how to load XMSG into a SINTRAN III VS(E) system:

If a new system has just been installed start at section 4.9.1
If a HENT/LOAD operation has been done, start at section 4.9.3
If a normal system restart (Master Clear/Load) has been done, start at section 4.9.4

4.9.1. Assumptions prior to loading

1) That user UTILITY exists and has enough space (300 pages, say).

2) That (UTILITY)SYMBOL-1-LIST:SYMB and SYMBOL-2-LIST contain the (MAC readable) symbol lists for parts 1 and 2 of SINTRAN. These are delivered together with SINTRAN.

3) That the SINTRAN system you have ordered has XMSG resident routines included (library mark 8XMSG). If not, order a new SINTRAN!

4) That the following subsystems have been installed: MAC, FMAC, NPL, NRL (with those names!) If other names are used, the XMSG-:MODE files should be edited accordingly.

4.9.2. Generating XMSG

a) Log in as UTILITY.

b) Copy all XMSG files to UTILITY using the BACKUP-SYSTEM.

c) Define the XMSG Configuration:

If a non-standard XMSG configuration is required, XMSG-SYS-DEF must be edited. See Appendix D for a list of symbols defined in this file, together with their meaning and default value. Some care must be taken here, in particular if the system is to be run on NORD-10, in which case the 8X100 library mark must be reset and the buffer space moved from physical memory to POF (X5SBS must be zero, and X5BUF equal to the number of words of buffer space required). This will then imply that 5XFPP (first page to use in POF) must be decremented accordingly.

d) Generate XMSG:

@MODE XMSG-GENERATE:MODE LINE-PRINTER

This does not affect the running system, but results in the creation of the files XMSG-POF:BPUN (POF code), XMSG-XROUT:BPUN (Routing program) and XMSG-SEGMENT:BPUN and their listings.

4.9.3. Loading XMSG

This needs to be done after all)HENT and)LOAD operations:

a) Log in as SYSTEM.

b) Load XMSG

@MODE (UTILITY)XMSG-LOAD:MODE,,

1) loads XMSG-POF:BPUN onto segment 33 (non-demand).

2) loads XMSG-SEGMENT:BPUN onto segment 35 (will be FIXC'd in physical memory to provide buffer space for messages, frames and indirect transfer).

3) loads XMSG-XROUT:BPUN onto segment 34.

4) creates the foreground programs XROUT and XTRACE.

5) patches XMSGU+4 in resident to -1 to indicate that XMSG is loaded.

c) Restart the System:

@RESTART-SYSTEM

Leads to the POF space needed by XMSG being reserved as swapping pages.

If the system does not restart now, but instead stops at ERRFATAL the chances are that there was not enough space in the Paging Off area. One can then either do a)HENT or use MACM to patch back location 165 in image to 0 (clears XMSG) and then restart by typing 22! Contact support for further help in finding space in POF.

4.9.4. Starting XMSG

After a normal MASTER CLEAR, LOAD sequence (or RESTART-SYSTEM) the START-XMSG command in the SINTRAN service program should be executed:

```
@SINTR
*START-XMSG
OK: XMSG STARTING UP
*EX
@
```

which leads to the XMSG-POF and buffer area segments being fixed (FIXC) into physical memory, the monitor call being enabled, and XROUT started. This operation should be included in the normal start up sequence and executed before starting NORDNET or SPOOLING, since these can 'steal' the POF space reserved for XMSG when they fix their segments.

Since XROUT now has to find space in which to fix segment 35, it may take a short time between the OK: ... message being printed (usually <2secs) and XMSG being fully active. This may imply that it is worthwhile putting a short HOLD in startup Mode files if the next operation requires XMSG to be running.

4.9.5. Stopping XMSG

If at any time XMSG needs to be stopped (it can later be restarted), the STOP-XMSG command in the SINTRAN service program can be used. This disables the XMSG monitor call and releases the physical memory space.

4.10. Overview of files on ND-10130

XMSG is loaded in 4 parts: 1) POF - loaded into physical memory in the 33-64k part of bank 0 (known as the Paging Off Area - POF) and is on segment 33, 2) Buffer area in physical memory on segment 35, 3) XROUT - an RT program running on (demand) segment 34 and 4) XMSG-COMMAND - a normal background program.

The loading address for the POF part of XMSG is system dependent, as are some of the internal configuration parameters for XMSG, so some preparation needs to be done and values set by editing a file, unless a default configuration is required. A short description of the files distributed on the XMSG floppy disk follows.

The only file that needs to be edited is XMSG-SYS-DEF.

4.10.1. System Definition Files

These files describe the tables used by XMSG and define the configuration. They are all of type :SYMB.

4.10.1.1. XMSG-SYS-DEF - XMSG System Definition file

This file contains a sequence of NPL code that defines the symbols needed for generating XMSG. The values of these symbols can therefore be redefined by editing this file, noting that the values are OCTAL.

4.10.1.2. XMSG-VALUES - Function and Error Symbols

This file defines values for symbols used to represent the XMSG functions and services provided by XROUT, and associated error codes. Explicit constant values should never be used in programs using XMSG, but the symbolic names used instead, after including the file (e.g. MON 2XMSG). This file was previously known as XMSG-FUNC-VALUES

4.10.1.3. XMSG-SYSTABS - XMSG Internal Table Descriptions

NPL file defining the structure of XMSG internal tables shared between all parts of XMSG. It is not usually relevant to users.

4.10.1.4. XMSG-POFTABS - XMSG Internal Table Descriptions

As XMSG-SYSTABS, but defining tables shared between the POF part and XMSG-COMMAND.

4.10.1.5. XMSG-SIN-DATA - SINTRAN Table Descriptions

As XMSG-SYSTABS, but defines the structure of SINTRAN tables accessed by XMSG. As far as possible the same symbolic names have been used as in SINTRAN.

4.10.2. XMSG-XROUT:SYMB - The Routing Program

4.10.3. XMSG-POFCODE:SYMB - The POF Kernel Code

4.10.4. XMSG-MULTI-MC - The Multi-Machine XMSG Code

4.10.5. XMSG-COMMAND:PROG - The Command Program

Needs to be copied in, and dumped as a reentrant subsystem, if required. Start and restart addresses are 0 and 1 respectively.

4.10.6. XMSG-LIBRARY:BRF - Library Routines

This file contains the BRF form of the routines which can be used to build up a message to XROUT, as well as an error routine.
This file was previously called XMSG-MESS-FORMAT:BRF

4.10.7. Mode Files

The following mode files (of type :MODE) are used for generating and loading XMSG.

4.10.7.1. XMSG-GENERATE:MODE

When XMSG is first obtained, or a new version of SINTRAN is ordered, the XMSG-GENERATE:MODE file should be run using the MODE command after editing the XMSG-SYS-DEF file to define the XMSG configuration required, although the values provided are probably OK for a first attempt!
XMSG-GENERATE creates the files required by XMSG-LOAD:

4.10.7.2. XMSG-LOAD:MODE

One of these files should be run (depending on which SINTRAN III version is being used) using the MODE command after loading a new system, or restarting after)HENT or)LOAD. After executing it, the system should be restarted using the RESTART-SYSTEM command (or MASTER-CLEAR, LOAD).
After restarting, the SINTRAN-SERVICE command START-XMSG, starts XMSG.

4.10.8. XMSG Generation Definition Symbols (XMSG-SYS-DEF)

The following is a list of symbols describing the XMSG configuration together with their meaning and default values. The symbols are defined (in OCTAL) in the file XMSG-SYS-DEF. The information at the end of each comment defines the approximate space required in bytes (decimal) per element and where (POF=Paging Off, PHYS=Physical Memory outside bank 0, SEG=XROUT Demand Segment).

Symbol	Default Value	Meaning of symbol
8X100	Set	Library mark set if this generation is only to be run on ND-100 CPUs.
XMSGM	Reset	Library mark set to include the multi-machine XMSG code.
8TRAC	Reset	Library mark set to include trace calls and handling code.
X5MMX	2000	Maximum message size in bytes.
X5MTS	4000	Maximum number of bytes of message space that can be owned by a task at one time (see XFGET function - Get message space.)
X5VBT	240	is a "tuning parameter." It determines the minimum transfer size in bytes for which it is worthwhile using the window copying mechanism to transfer data between a user logical space and physical memory. This can be different for different machines, but a non-optimal setting will only lead to a slight loss of speed. If in doubt (!) leave this to the original default.
X5RTP	400	Number of RT programs in this system. (*2 POF.)
X5NAM	100	Maximum number of names that can be remembered in XROUT. (XROUT demand segment)
X5NLW	20	Maximum length of a name in words (SEG).
X5MLT	20	Maximum machine number in network (*2 POF.)
X5MXH	10	Maximum number of hops a frame can make before it gets thrown away.
X5LTO	100	Link Layer Receiver timeout in basic time units.
X5IRM	100	Default number of attempts when opening a link.
X5RPM	5	Maximum number of unsuccessful repeats before closing a link.

X5LNK	4	Maximum number of links (*64 POF).
X5PIO	4	Maximum number of PIOCs (*? POF).
X5NBF	3	Default number of XF-blocks/links (see X5FRM).
X5TSK	20	Number of task descriptor blocks. This is equal to the maximum number of tasks that can use XMSG simultaneously. (*60 POF).
X5PRT	20	Number of port descriptor blocks. This is equal to the maximum number of ports that can be opened simultaneously. (*14 POF).
X5MES	40	Number of message descriptors (*32 POF).
X5COM	4	Number of frames that can be under transmission simultaneously from local messages (*48 POF).
X5FRM	10	Number of receive or forward frame buffers(*48 POF, *X5FSZ PHYS).
X5FSZ	400	Maximum frame size in bytes.
X5BUF	0	Buffer area in POF in words for NORD-10 only. (*2 POF).
X5SBS	20000	Buffer area in physical memory in words (*2 PHYS).
5XSG3	35	XMSG segment 3. If SINTRAN III F or later, use 35.
5XFPP	71	defines the first page that will be used by XMSG in the POF. (Paging Off Area.)
5XPPS	140	first page to use in physical memory for message buffers and frame buffers and indirect transfer buffer.
X5TRB	2	number of trace buffers if 8TRAC (*4 POF+*1024 PHYS).

The other symbols set in XMSG-SYS-DEF are dependent on the symbols defined above.

5. HIGH LEVEL DATA LINK CONTROL (HDLC) DMA (OPTION)

5.1. Introduction

The HDLC Monitor Call (MON 201) is used to control a High Level Data Link Control (HDLC) Interface. This is a synchronous modem interface which can also be used as an intercomputer link interface. The Monitor Call MON 201 is currently used by both HDLC DMA Controller (ND-720 or ND-730) and X.21 (next chapter).

The HDLC DMA option is included in the user's SINTRAN III configuration when he orders his operating system from Norsk Data A.S.

The HDLC-driver and the user program communicate by means of Driver Control Blocks (DCB). The first 3 words of the DCB contain control and status information, while the rest of the block may contain data frames or additional information. Section 6.4 describes the DCB format in detail.

For the purpose of transferring DCB's between user programs and the HDLC-driver, the monitor call HDLC is used as explained in the next section.

5.2. The Monitor Call HDLC (MON 201)

MON HDLC is used for transferring DCB's back and forth between user programs and the HDLC-driver.

DCB's are sent from a user program to the HDLC-driver using SEND. When a DCB is sent to the driver, it is copied from the user program to a driver buffer-area. Here it is inserted in a queue of DCB's to the driver. The driver processes the DCB's one by one and, after processing, puts them in another queue of DCB's back to the user program. The user program can then receive them using RECEIVE. This is done by copying the DCB from the buffer area to the user program. Since the receiving of DCB's may be done asynchronously with respect to the sending, each DCB is given an identifier.

Figure 5-1 shows how the data transfer is organized. The DCB's are filled with frame data or emptied of frame data by the driver.

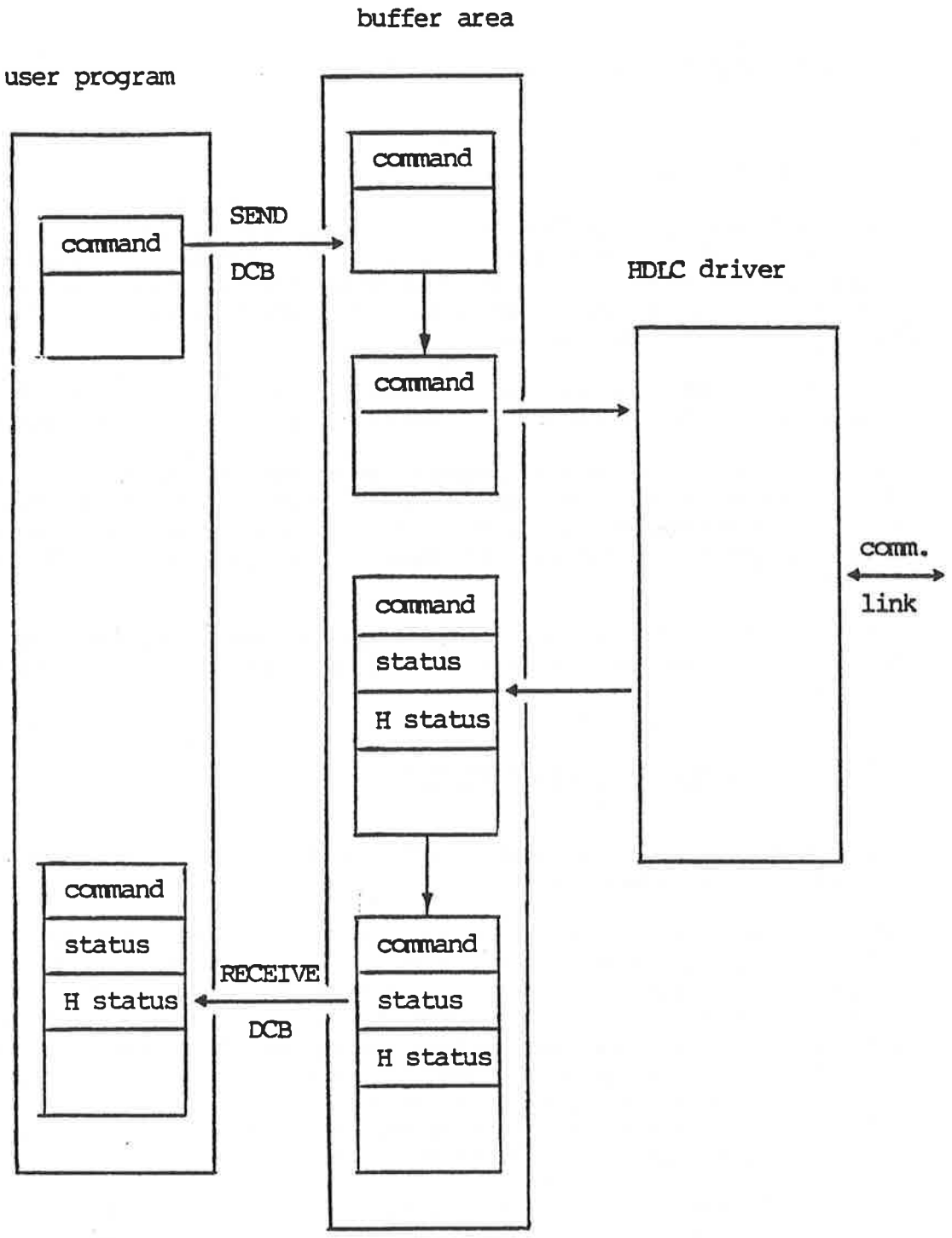


Figure 5-1 Data Organization

5.2.1. HDLC Monitor Call Format

5.2.1.1. Calling HDLC in NPL

When calling HDLC in NPL, the A-reg points to a list of parameter addresses.

```
        LDA       (PLIST
        MON       201
        JMP       ERROR
        JMP       OK
        -------
PLIST,  PARAM1 ADDRESS
        PARAM2 ADDRESS
        PARAM3 ADDRESS
        PARAM4 ADDRESS
        PARAM5 ADDRESS
```

Parameters 1 to 5 are described in the next section.

There are two basic return sequences.

- skip return: DCB successfully transferred
  A-reg contains DCB identifier.

- Non skip return: Error in DCB transfer
  A-reg contains error code. (Negative)
  See Appendix E

5.2.1.2. Calling HDLC from FORTRAN

ISTAT = HDLC (PARAM1,PARAM2,....,PARAM5)

If ISTAT is positive, the transfer of the DCB was sucessful, and ISTAT is set to the DCB-identifier. Otherwise, if ISTAT is negative, an error has occured. For further decription, see the list of error codes in Appendix E.

In the remaining part of this section, we will simply refer to the argument list as (PARAM1,PARAM2, ...,PARAM5). As described below, PARAM1 gives the function (0 for SEND, 1 for RECEIVE), PARAM2 is the logical device number, PARAM3 is the DCB, PARAM4 the used DCB size, PARAM5 the max DCB size or a wait flag.

The size of the DCB is given in two parameters, used size and maximum size. The maximum size is the size of the DCB in the driver buffer area. The used size is the size of the DCB in the user program, when it is sent or when it is received. The sent size and the received size may differ, for instance when the sent DCB contains frame data and the received DCB only the resulting status. The maximum size will be the larger of the two used sizes.

The symbols below will be used in the parameter list.

SDCB	- Equals 0. Send DCB to driver.
RDCB	- Equals 1. Receive DCB from driver.
LDN	- Logical Device Number. One for input, one for output.
DCB	- The DCB to be sent or received.
DCB-usize	- Used size of the DCB in number of bytes.
DCB-msize	- Maximum size of the DCB in number of bytes.

5.2.1.3. The use of Device Numbers in Mon HDLC

One HDLC-Interface requires use of two logical device numbers (LDN). One LDN will cover the input part, while the other will cover the output part of the interface.

For obtaining exclusive access to a LDN, the monitor calls RESERVE and RELEASE should be used.

Note that there is one SEND queue and one RECEIVE queue for each LDN, and that one LDN controls only input from the communication link or output to the communication link. The HDLC driver can handle many LDN´s and has two queues per LDN. Some queues may be empty (LDN3 SEND and LDN4 RECEIVE).

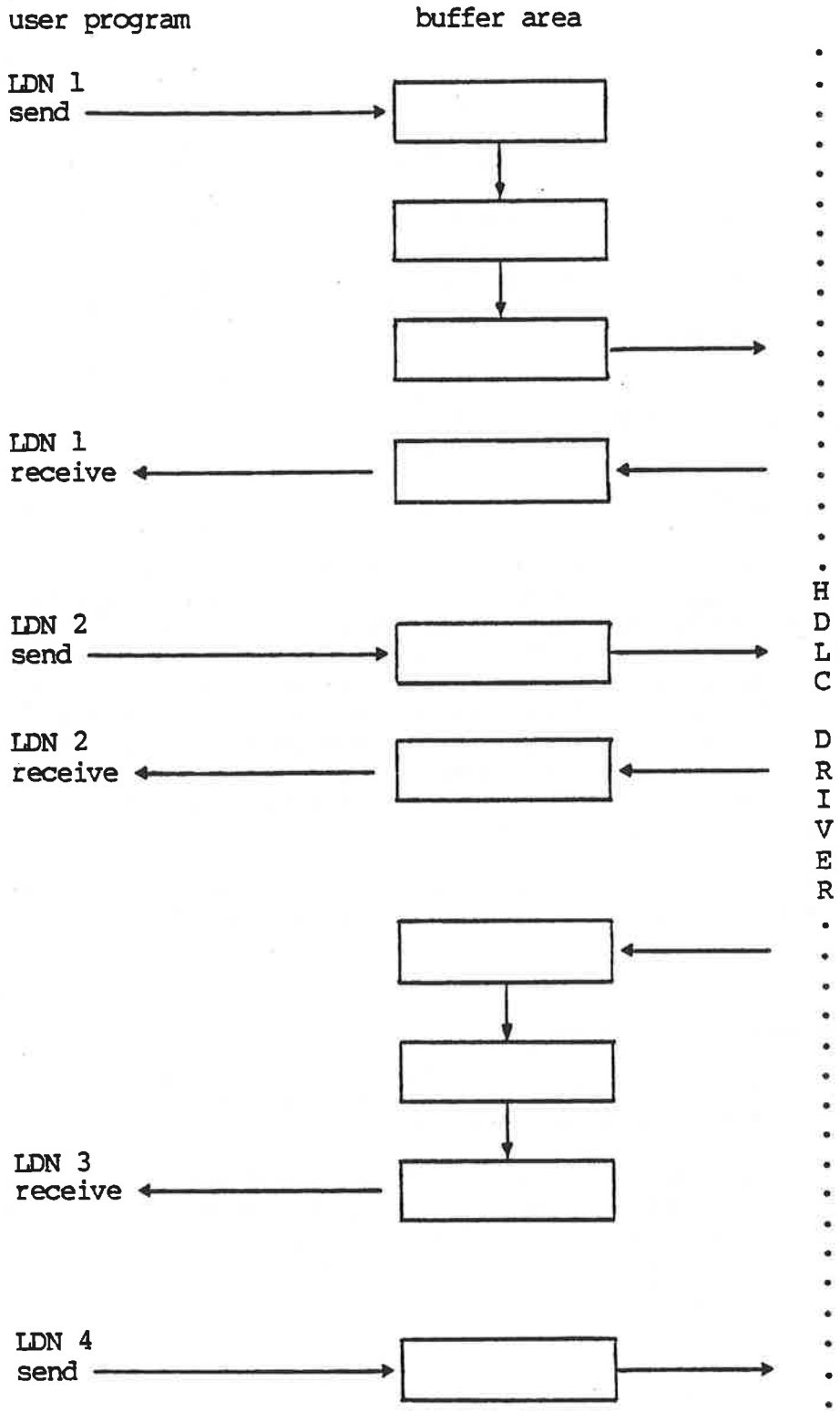


Figure 5-2 Queues of Driver Control Blocks (DCB)

5.2.2. Send DCB (SDCB)

Istat = HDLC (SDCB,LDN,DCB-address,DCB-usize,DCB-msize)

SDCB is used when the user program wants to send a DCB to the HDLC-driver. The DCB is chained to the tail of the driver queue. Only as many bytes as specified in DCB-usize is sent to the driver. However, upon later reception of the same DCB, after driver treatment, it might have grown to the size specified in DCB-msize.

5.2.3. Receive DCB (RDCB)

Istat = HDLC (RDCB,LDN,DCB,DCB-usize,WAITFL)

By means of RDCB, the user program may get DCB´s back from the HDLC-driver. If no completed DCB exists, the system response depends upon the use of the WAITFL-parameter. If WAITFL is 1, the calling program is set in I/O-WAIT until a DCB arrives from the driver. If WAITFL is 0, the user program will always continue whether there exists a DCB for it or not. However, if RTWT is called after an unsucessful RDCB, the program will be activated at the instruction following MON RTWAIT, when a DCB is sent to it. This property may be useful if one program controls many LDN´s, and is not sure which LDN the next DCB is going to come from and when it will come.

The DCB-usize is set by the driver to the size of the DCB after driver treatment. DCB size may never exceed the DCB-msize used when the DCB was sent to the driver.

5.3. The Driver Control Block

A user program makes a request for service to the HDLC-driver by sending it a block of data called the Driver Control Block (DCB).

The first 3 words in the DCB contain command and status information, while the rest may contain frame data or additional information. When the request is granted by the HDLC-driver, the STATUS parameter is updated, and the DCB is sent back to the user program.

5.3.1. The Driver Control Block Format

The DCB has the following general format used when transferring data.

NOTE: Other commands such as device initialization and device status, have slightly different formats.

```
----------------------
- COMMAND            -
- STATUS             -
- HARDWARE STATUS    -
- FRAME DATA         -
- ....               -
- ....               -
----------------------
```

The various commands are decribed in the next section. Status codes are found in Appendix E. The hardware status (HSTAT) varies depending on the transfer direction. It is not updated if the transfer is successful. If an output LDN is used, hardware status is a copy of the transmitter transfer status register. This is described under programming specifications (IOX GP + 12) in the manual "HDLC - High Level Data Link Control Interface" (ND-12.018). If an input LDN is used, HARDWARE status is a copy of the receiver list status word found in the section on receiver lists in the same manual.

In the remainding of this section we will simply refer to the DCB as

(PARAM1,PARAM2,....)

5.3.2. HDLC-Driver Commands

The commands used in the DCB are,

DEVCL	(=3)	Device Clear
DEVINI	(=4)	Device Initialization
RESET	(=2)	Reset Logical Device
TRANS	(=1)	Transfer Frame Data
DEVSTA	(=5)	Get Device Status

In this chapter these symbols will be used in the DCB argument list,

STAT	- Status. result of operation.
NA	- Argument not applicable.
HSTAT	- Hardware status. For details see the HDLC manual ND-12.018.
FRDAT	- Frame of data, DCB-usize is the number of bytes in the frame.

5.3.2.1. Device Clear (DEVCL)

(DEVCL,NA,NA)

DEVCL will completely clear the HDLC-interface. Both the input and the output side of the interface will be cleared. All data transfer to and from the interface will stop. DCBs currently being processed by the driver will be returned, with the value of STAT = 110. Further use of the interface must include the DEVINI command.

Input parameters (set by user)

DEVCL equals 3

5.3.2.2. Device Initialization (DEVINI)

(DEVINI,STAT,HASTAT,MODUS,FRSIZE,MAXERR,DISP)

DEVINI should always be used after DEVICE CLEAR. The command will give the interface necessary information related to the operation mode.

Input parameters (set by user)

DEVINI equals 4

MODUS=0: full duplex operation, MODUS=1: half duplex, MODUS=2: maintenance mode, looping transmitted data back to received data.

FRSIZ specifies the maximum size of the dataframes to be transferred.

MAXERR is number of retries in case of errors. It only applies to the output side of the interface.

DISP or displacement is the number of free bytes reserved at the beginning of each dataframe in the DCB. Only the FRAME DATA part of the DCB is affected (displaced).

Output parameters (set by driver)

STAT is the resulting status of the operation, see Appendix F.

HSTAT is set to the checksum given by the interface as a response to the initialize command. See the section on initialization in the HDLC manual, ND-12.018.

5.3.2.3. Device Reset (RESET)

(RESET,NA)

The command is used to reset either the input or the output side of the interface depending on the LDN. It is not necessary to do DEVINI after this command. DCB's currently being processed by the driver will be returned, with the value of STAT = 110.

Input parameters (set by user)

RESET equals 2

5.3.2.4. Transfer Frame Data (TRANS)

(TRANS,STATUS,HSTAT,FR-DATA ...)

The command is used for transferring frames of data to and from the computer. If the DCB is sent to an output LDN, the frame data will be transferred to the communication link. If the DCB is sent to an input LDN, the frame data-part of the DCB will be filled with data from the communication link.

Input parameters (set by user)

TRANS equals 1

Output parameters (set by driver)

STAT is the result of the operation, see Appendix F.

HSTAT, see general description in section 5.3.1

FR-DAT is the array of frame data bytes received or transmitted. The array length is equal to DCB-usize.

5.3.2.5. Device Status (DEVSTAT)

(DEVSTAT,NA,NA,ERRNO,ORERR,LHAST,RSTOP,MAXR)

This command will give status information about the LDN used. The internal parameters inspected will be cleared.

Input parameters (set by user)

DEVSTAT equals 5

Output parameters (set by driver)

ERRNO is set to the total number of errors detected. The internal parameter ERRNO is set to zero after this call.

ORERR is an OR function of all hardware errors at the LDN. The hardware status register used, is the transmitter status register for an output LDN, and the receiver list status word for an input LDN. The internal parameter ERRNO is set to zero after this call. See the HDLC manual, ND-12.018.

LHAST is set equal to the last hardware status detected by the driver. (Transmitter Transfer Status or Receiver List Status word according to LDN, see the HDLC manual.)

RSTOP is the number of receiver stops due to lack of buffer space. Note that the user is responsible for providing the input LDN with sufficient buffer space. He must send enough DCBs with command TRANS to the LDN. The internal parameter RSTOP is set to zero after this call.

MAXR is set to the maximum number of DCB´s which can possibly be held by the driver. The argument only applies to an input LDN.

5.4. How to Program the HDLC-Driver

The two first DCB´s sent to the driver must be DEVICE CLEAR and DEVICE INITIALIZATION. These DCBs may be sent to both logical devices, as they will affect both the input and the output side. To check the driver reaction, the status information from the driver is obtained when the DCBs are returned by doing two RECEIVE DCB on the same LDN. After successful completion of DEVICE CLEAR and DEVICE INITIALIZATION, the actual data transfer may start. The methods of controlling the two LDNs are somewhat different, as explained below.

5.4.1. The Input LDN

As it is difficult to predict the arrival of data, the driver must have some amount of buffer space for storing the frame data when it arrives. This should be done initially by sending empty (command TRANS) DCBs to the driver. However, the driver is only capable of holding a limited number of empty DCB´s. This number can be obtained from the RMAX parameter of the DEVSTAT command. When this limit is reached, the driver will send the empty DCB back to the source with status 110.

The user program may then get a DCB back by doing a RECEIVE DCB. Normally the DCB will now contain data. To maintain the driver´s bufferspace, the user program should, upon receiving one DCB, send the driver a new empty one.

Note that the receiving part of the interface is activated when the driver receives the first empty DCB.

5.4.2. The Output LDN

The driver is activated by a "SEND DCB". The command in the DCB should be TRANSFER. The driver will then always give status information which the user program can receive on a RECEIVE DCB to the same LDN.

5.4. How to Program the HDLC-Driver

The two first DCB´s sent to the driver must be DEVICE CLEAR and DEVICE INITIALIZATION. These DCBs may be sent to both logical devices, as they will affect both the input and the output side. To check the driver reaction, the status information from the driver is obtained when the DCBs are returned by doing two RECEIVE DCB on the same LDN. After successful completion of DEVICE CLEAR and DEVICE INITIALIZATION, the actual data transfer may start. The methods of controlling the two LDNs are somewhat different, as explained below.

5.4.1. The Input LDN

As it is difficult to predict the arrival of data, the driver must have some amount of buffer space for storing the frame data when it arrives. This should be done initially by sending empty (command TRANS) DCBs to the driver. However, the driver is only capable of holding a limited number of empty DCB´s. This number can be obtained from the RMAX parameter of the DEVSTAT command. When this limit is reached, the driver will send the empty DCB back to the source with status 110.

The user program may then get a DCB back by doing a RECEIVE DCB. Normally the DCB will now contain data. To maintain the driver´s bufferspace, the user program should, upon receiving one DCB, send the driver a new empty one.

Note that the receiving part of the interface is activated when the driver receives the first empty DCB.

5.4.2. The Output LDN

The driver is activated by a "SEND DCB". The command in the DCB should be TRANSFER. The driver will then always give status information which the user program can receive on a RECEIVE DCB to the same LDN.

6. X.21 COMMUNICATION PROTOCOL

6.1. Introduction

The CCITT X.21 recommendation defines the physical characteristics and the call control procedures between the DTE (Data Terminal Equipment, The Subscriber) and the DCE (Data Circuit Equipment, The Network). The Nordic Public Data Network (NPDN) operates in accordance with this recommendation.

The X.21 Monitor Call is used to control a High Level Data Link (HDLC) Interface in accordance with the CCITT X.21 recommendation. The Monitor Call number is the same as for MON HDLC. The system uses the logical device number (LDN) to distinguish between HDLC and X.21.

The X.21-driver and the user program communicate by means of Driver Control Blocks (DCB).

For the purpose of transferring DCB´s between user programs and the X.21-driver, the monitor call X.21 is used. For a description of MON X.21 see section 6.2 below

The format of the DCB and the different X.21 commands can be found in section 6.3.

6.2. The Monitor Call X.21 (MON 201)

MON X.21 is used to transferring DCB´s between the user programs and the X.21-driver.

DCB´s are sent from a user program to the X.21-driver using SEND. When a DCB is sent to the driver, it is copied from the user program to a driver buffer-area. Here it is inserted in a queue of DCB´s to the driver. The driver processes the DCB´s one by one and, after processing, puts them in another queue of DCB´s back to the user program. The user program can then receive them using RECEIVE. This is done by copying the DCB from the buffer area to the user program. Since the receiving of DCB´s may be done asynchronously with respect to the sending, each DCB is given an identifier.

Figure 6-1 shows the data organization of the X.21 software. The parameters STATUS and CP-SIGNAL are filled in by the X.21-driver.

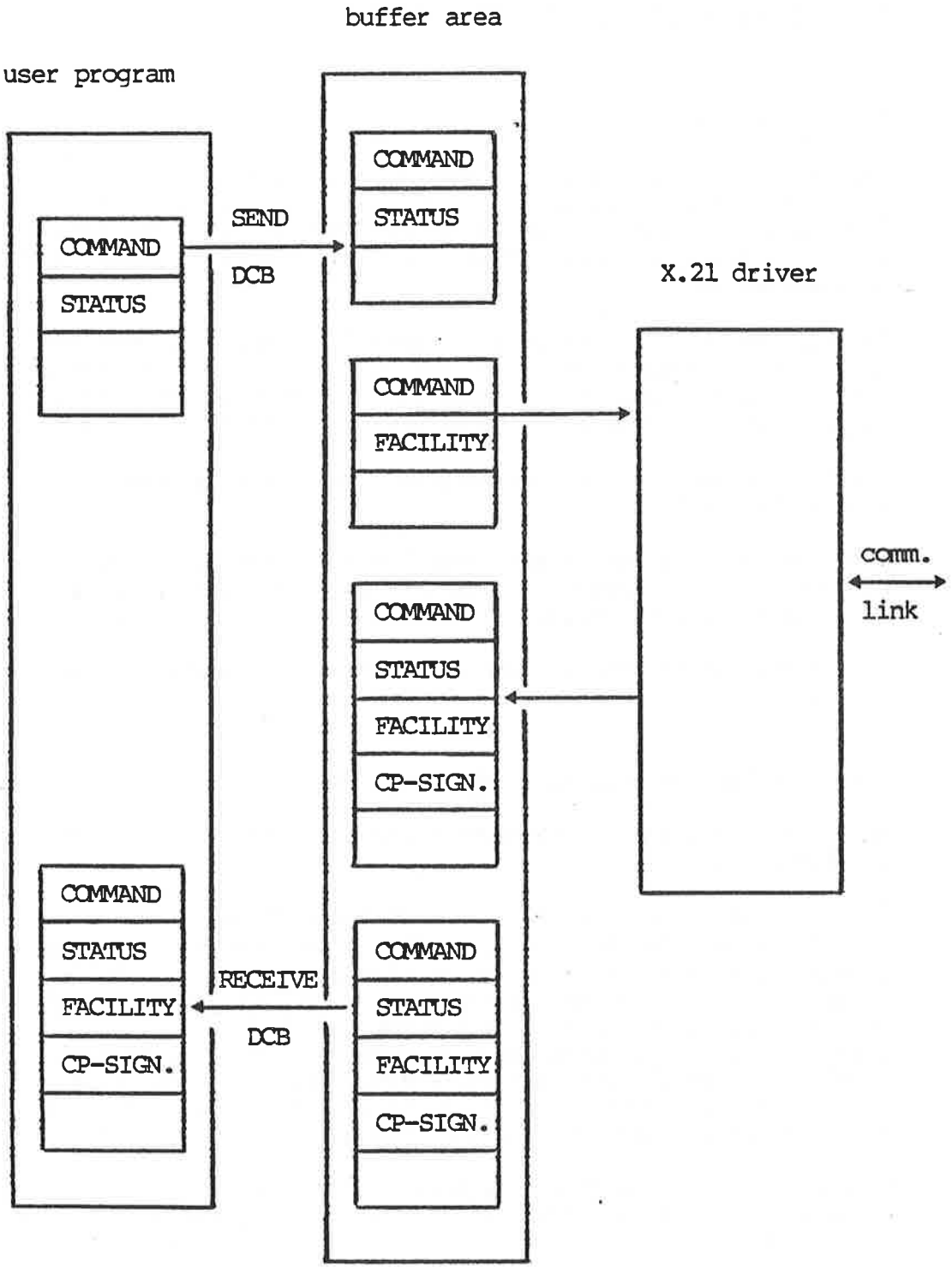


Figure 6-1 Data Organization in X.21

6.2.1. X.21 Monitor Call Format

6.2.1.1. Calling X.21 in NPL

When calling X.21 in NPL, the A-register points to a list of parameter addresses.

```
         LDA       (PLIST
         MON       201
         JMP       ERROR
         -------
PLIST,   PARAM1 ADDRESS
         PARAM2 ADDRESS
         PARAM3 ADDRESS
         PARAM4 ADDRESS
         PARAM5 ADDRESS
```

There are two basic return sequences.

- Skip returnø — DCB successfully transferred
  A-register contains DCB identifier.

- Non skip returnø — Error in DCB transfer
  A-register contains error code.
  (Negative)

6.2.1.2. Calling X.21 from FORTRAN

```
ISTAT = X.21 (PARAM1,PARAM2,....,PARAM5)
```

If ISTAT is positive, the transfer of the DCB was sucessful, and ISTAT is set to the DCB-identifier. Otherwise, if ISTAT is negative, an error has occurred. For further decription, see the list of error codes in Appendix J.

6.2.1.3. The Arguments of MON X.21

In the remaining part of this section, we will simply refer to the argument list as (PARAM1,PARAM2, ...,PARAM5). As decribed below, PARAM1 gives the function (0 for SEND, 1 for RECEIVE), PARAM2 is the logical device number, PARAM3 is the DCB address, PARAM4 the used DCB size, PARAM5 the maximum DCB size or a wait flag.

The size of the DCB is given in two parameters, used size and maximum size. The maximum size is the size of the DCB in the driver buffer area. The used size is the size of the DCB in the user program when it is sent or when it is received. The sent size and the received size may be quite different. The maximum size will be the larger of the two used sizes.

The symbols below will be used in the parameter list.

SDCB	- Equals 0. Send DCB to driver.
RDCB	- Equals 1. Receive DCB from driver.
LDN	- Logical Device Number.
DCB-address	- Address of the DCB
DCB-usize	- Used size of the DCB in number of bytes.
DCB-msize	- Maximum size of the DCB in number of bytes.

6.2.1.4. The use of Device Numbers in Mon X.21

There is one logical device number (LDN) used for each Data Network Connection.

To obtain the exclusive access to an LDN, the monitor calls RESERVE and RELEASE should be used.

Note that there is one SEND queue and one RECEIVE queue for each LDN.

6.2.2. Send DCB (SDCB)

ISTAT = X.21 (SDCB,LDN,DCB-address,DCB-usize,DCB-msize)

SDCB is used when the user program wants to send a DCB to the X.21-driver. The DCB is put at the end of the driver queue. Only as many bytes as specified in DCB-usize are sent to the driver. However, when receiving the same DCB at a later time (after driver treatment) it might have grown to the size specified in DCB-msize.

6.2.3. Receive DCB (RDCB)

ISTAT = X.21 (RDCB,LDN,DCB-address,DCB-usize,WAITFL)

By means of RDCB, the user program may get DCB´s back from the X.21-driver. If no complete DCB exists, the system response depends upon the use of the WAITFL-parameter. If WAITFL is 1, the calling program is set in I/O-WAIT until a DCB arrives from the driver. If WAITFL is 0, the user program will always continue whether there is a DCB for it or not. A subsequent call to RTWT (MON 135) should be used to wait for a DCB. This feature may be useful if one program controls many LDN´s, and it is not known which LDN the next DCB is going to come from, or when it will come.

The DCB-usize is set by the driver to the size of the DCB after driver treatment. The DCB size may never exceed the DCB-msize specified, when the DCB was sent to the driver

6.3. The X.21 Driver Control Block

A user program makes a request for service to the X.21-driver by sending it a block of data called the Driver Control Block (DCB).

The first 4 words in the DCB contain command and status information, while the rest of it may contain additional information. When the request is fulfilled by the X.21-driver, the STATUS parameter is updated, and the DCB is sent back to the user program.

6.3.1. The X.21 DCB Format

The X.21 DCB has the following general format.

```
   -------------------------------
P1 - COMMAND                     -   Integer (1 word)
P2 - STATUS                      -   Integer (1 word)
P3 - FACILITY                    -   Bits    (1 word)
P4 - CALL PROGRESS SIGNALS       -   IA5     (1 word)
P5 - DTE/DCE PROVIDED INFORMATION -  IA5     (n words)
   - ---------- " ----------     -    "
   - ---------- " ----------     -    "
   - ---------- " ----------     -    "
   - ......                      -    "
   -------------------------------
```

COMMAND is given by the user. A description are found in next section.

FACILITY is given by the user. By using this parameter, the user may make requests for optional service provided by the network. The coding is decribed in Appendix H.

STATUS is provided by X.21, and indicates the result of operation (Appendix H).

CALL PROGRESS SIGNALS is status information provided by the DCE. The characters used are selected from the International Alphabet No. 5 (Appendix I).

DTE/DCE PROVIDED INFORMATION field will hold information either given by the user (DTE PROVIDED) or the DCE (DCE PROVIDED) or both. All characters in this parameter should be according to the International Alphabet No. 5.

The only DTE PROVIDED information currently applicable is the subcriber number used in the CALL command.

When it comes to the DCE PROVIDED information, two different types may occur;

Called/Calling Line Identification
Charging Information

The Called/Calling Line Identification will have as a prefix the IA5 character "*" when the call is national, and "**" when the call is international. The Charging Information will have as a prefix the IA5 character "/". For a detailed decription of the Charching Information, see the specific command.

In the remainding part of this section we will simply refer to the DCB as

(PARAM1,PARAM2....)

6.3.2. The X.21 Commands

The following values are used for PARAM1.

CONNECT	(=-5)	Connect LDN´s to X.21
DISCONNECT	(=-6)	Disconnect LDN´s from X.21
CALL	(=-1)	Call request
READY	(=-2)	Ready for incoming call
CLEAR	(=-3)	Clear
GCHAR	(=-4)	Get charging information
RDIRC	(=-7)	Redirection of calls
GSTAT	(=-8)	Get status
TERM	(=-9)	Returned when call terminated

6.3.2.1. Connect (CONNECT)

(CONNECT,STATUS,NA,NA,ILDN,OLDN,RTUSER)

This command will connect the current X.21 LDN with ILDN (Input Logical Device Number) and OLDN (Output Logical Device Number), and by doing so enabling ILDN and OLDN to be used for transfering data on a X.21 network. All X.21 commands must be sent to the X.21 LDN, while data is transferred through ILDN and OLDN. The ILDN and OLDN must be reserved by an RT-program prior to using this command. The RTUSER parameter holds the address of the RT-program having reserved the ILDN and the OLDN. RTUSER set to 0 indicates that the current RT-program has reserved ILDN and OLDN. ILDN, OLDN and RTUSER are all integers.

6.3.2.2. Disconnect (DISCONNECT)

(DISCONNECT,STATUS,NA,)

The DISCONNECT command is used to cancel the last CONNECT command. The current X.21 LDN is disconnected from ILDN and OLDN.

6.3.2.3. Call (CALL)

(CALL,STATUS,FACILITY,CP,DTE/DCE PROVIDED INFORMATION)

CALL will try to establish a connection with the DTE having the number specified by the user in the DTE PROVIDED INFORMATION field. The parameter must be terminated by "+".

The FACILITY PARAMETER may contain combinations of the following bits; CHARGING INFORMATION, CALLED LINE INFORMATION, and CONNECT WHEN FREE. (See appendix H.) In the case of CHARGING INFORMATION, the network will send charging information when the call is terminated. The information is available through the GCHAR command. The CALLED LINE IDENTIFICATION will be returned within the current message in the DTE/DCE PROVIDED INFORMATION field.

When the CONNECT WHEN FREE facility is requested, the X.21-driver will wait until the connection is established. If however, a new message is sent to the X.21-driver the call will be terminated, and the message will be returned with status 21.

6.3.2.4. Ready (READY)

(READY,STATUS,NA,CP,DTE/DCE PROVIDED INFORMATION)

By sending this command, the user indicates that he is ready to accept incoming calls. The message will be returned to its originator when an incoming call arrives, or if some error occurs. The message will also be returned, and the Ready state terminated if a new message is sent to the same LDN. If the option for calling line identification is available for this subcriber, the information will be found in the DCE PROVIDED INFORMATION field.

The FACILITY parameter is not applicable in this command.

6.3.2.5. Clear (CLEAR)

(CLEAR,STATUS,NA,CP,NA)

This command will break any existing connection with the network, and set this subscriber in a "not ready" state. By doing so all communication with the network is disabled.

Only the STATUS and the Call Progress signals parameters are used in this command.

6.3.2.6. Get Charging Information (GCHAR)

(GCHAR,STATUS,NA,CP,DCE PROVIDED INFORMATION)

GCHAR is used to retrieve the charging information of the last call established with the CHARGING bit set in the FACILITY parameter. When receiving the DCB, the charging information is found in the DCB PROVIDED INFORMATION part of the DCB. This part informs the subscriber of either the monetary charges for a call, the duration of a call, or the number of units used during the call.

The syntax of the part is described below using Backus Normal Form (BNF) as shown in the CCITT document "DRAFT RECOMMENDATION X.21 - DRAFT REVISION A" with Addenda T119 and T123.

When charging information is given in monetary charges for the call, the prefix of the information is 1 and the information consists of x integer digits optionally followed by a colon and two digits representing the fraction. In general, the format is:

</><1></><X.....><+>

or

</><1></><X.....:YY><+>

When the charging information is presented as the duraction of a call, the prefix is 2. The information consists of x integer digits representing seconds. In general, the format is:

</><2></><X.....><+>

When the charging information is presented as the number of units used, the prefix is 3. The information consists of x integer digits representing the units. In general, the format is:

</><3></><X.....><+>

6.3.2.7. Redirection of Calls (RDIRC)

(RDIRC,STATUS,NA,CP,NA)

The purpose of this command is to redirect all incomming calls to another subscriber. The address of the new subscriber is predefined within the network.

6.3.2.8. Get Status (GSTAT)

(GSTAT,NA,NA,NA,STATUS)

The command will return the current status of the X.21 LDN. STATUS may take any of the following values:

=0 The X.21 LDN is not used.

=1 The connect command is used, but no link through the network is established.

=3 A connection through the network is established and a data transfer is currently going on.

6.3.2.9. Return when call terminated

(TERM,STATUS,NA)

The DCB will be returned if the current X.21-line is in or enters a non data phase state. The status parameter in the DCB will be set to 20(octal). See appendix K.

6.4. Writing HDLC Driver for X.21 Network

The Monitor Call MON X.21 is used for an explicit request to the network. This is done through the various commands decribed in previous sections.

When a connection is established, the DTE´s will be responsible for establishing their own alignment. In this phase, the data phase, a software driver will control the HDLC interface. Since the network at any time may initiate a disconnection, the following constraints will apply to the software driver.

When an input transfer is finished, the receiver transfer status register will be modified, and be of no use to the driver. But a copy with some additional information (bit 14 and 13) will be found in the A-register.

BIT 14 set to 1 means that the connection is broken due to DCE clearing.

BIT 13 set to 1 means that the transfer just completed is in error.

A P P E N D I X A

MAGTP Functions

Notes

1) See the section "Call Formats" following this section

2) Applies to the whole device, i.e. all units

3) Included in the 80B and later versions

4) Function depends on hardware configuration

5) Select parity and density as follows

<density/parity> = 0: 800 BPI, odd parity
= 1: 556 BPI, odd parity
= 2: 200 BPI, odd parity
= 3: 800 BPI, even parity
= 4: 556 BPI, even parity
= 5: 200 BPI, even parity

Default value is zero.

In some hardware conwigurations, the value is selected by setting a switch on the front panel.

6) Select density as follows

<density/parity> = 0: 1600 BPI
= 1: 6250 BPI

7) Clear unit only as determined by the logical device number.

8) Read density and parity (a)

9) Read parity and density (b)

10) Read format (c). (Floppy disk formats are shown in note 21 below, while a, b and c are explained in the section "Call Formats" on page 99 and 100.)

11) The format of ISTAT for Tandberg, Pertec or STC is (condition is set if bit is set)

bit 0: Tape on line
1: Write enable ring present
2: Tape standing on load point
3: CRC error/fatal error
4: Set if any of bits 5, 6, 7, 8, 9, 11, or 12 is set
5: Control or modus word error. Trying to write on unprotected tape, reserving tape at load point, tape unit not on-line, etc. Action is inhibited.
6: Bad data block. An error has been detected
7: End of file has been detected
8: The search character has been detected
9: End of tape has been detected. Resetting the bit depends on the model.
Tandberg, STC: The bit remains on if carrying out a function after EOT (end-of-tape).

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

12) For Hewlett-Packard magnetic tape the format of ISTAT is (condition is set if bit is set)

bit 0: Ready interrupt enabled (cleared by the interrupt)
1: Error interrupt enabled (cleared by the interrupt)
2: Device active
3: Device ready for transfer
4: Set if any of bits 6, 9, 10, 11 or 12 is set or if a reverse command is 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, on-line and not rewinding)
15: Bit 15 is loaded by the previous control word

13) Only available as @DEVICE-FUNCTION

14) For Philips cassette the format of ISTAT is (condition is set if bit is set):

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 0, 1, 4 or 5 is set
5: Write enable
6: Cassette side indicator (A = 1, B = 0)
7: Bit clock
8: Read fail
9: Sync fail
10: Not used
11: Not used
12: Drive fail
13: Write protect violation
14: Beginning or end of tape
15: Not used

15) For Versatec line-printer the format of ISTAT is (condition is set if bit is set):

bit 0: Ready for transfer, interrupt enabled
1: Error interrupt enabled
2: Device active
3: Device ready for transfer
4: Set if any of bits 6 or 7 is set
5: Not used
6: No paper
7: Plotter not on-line
8 - 12: not used, bits set at random
13: Plotter ready
14 - 15: Not used, bits set at random

16) For floppy disk the format of ISTAT is (condition is set if bit is set):

bit 0: Interrupt enabled
1: Not used
2: Device busy
3: Device ready for transfer
4: Set if 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: Not used

17) Write a block in a unique format to indicate EOF. The disk address is incremented by one.

18) The disk address is set to zero.

19) The disk address is decremented by one.

20) The disk address is incremented by one.

21) The following formats are available:

<input format> = 0: 256 words/sector, 8 sectors/track
(Standard format used by Norsk Data A.S)
= 1: 128 words/sector, 15 sectors/track
= 2: 64 words/sector, 26 sectors/track

22) All data on the diskette is overwritten and the diskette is formatted (i.e. new addresses are written)

23) Read a record even if it has been flagged as deleted.

24) After the record is written it is flagged as deleted.

25) Versatec may be used similarly to other line-printers. OUTBT (MAC/NPL) WRITE, OUTPUT or OUTCH (FORTRAN) is used to print characters. In order to reserve the access to the device it should first be opened, then written to and finally closed. On closing, the remaining characters to be printed are transmitted to the Versatec. (RESERV and RELES may also be used.)

26) SINTRAN III can handle at maximum two floppy disk controllers each having a maximum of three drives. Before it can be used, the floppy disk must be formatted. (Function 41, see SINTRAN TIME-SHARING/BATCH GUIDE, section 3.12)

Instead of creating a directory the floppy disk can be used as a sequential medium. It is then first created as a periheral file (@SET-PER-FI, @SET-FI-ACC) and ordinary I/O calls (INBT, OUTBT, etc.) are used. End-of-file (EOF) must be written after the last write call by using @DEVICE-FUNCTION or MAGTP, function 12.

27) Read hardware status on last unit operated upon. It can be any unit on the device.

28) Read hardware status of the last operation on the own device.

29) Tape is positioned immediately after the EOF.

30) Tape is positioned immediately in front of the EOF.

Call Formats

<d> = dummy parameter. Use a variable for this parameter, for ex.:

```
IDUM=0
ISTAT = MAGTP(40B,IDUM,1000B,IDUM)
```

<LDN> = logical device no.

In all formats, except a, ISTAT will receive error status on return. If ISTAT=0 the call terminated correctly. If ISTAT>0 it contains the file system error number, see appendix D of SINTRAN III REFERENCE MANUAL (ND-60.128).

a

The device must be reserved in order to read hardware status.
(If not, a positive value of ISTAT may not be the correct status.)

```
ISTAT= MAGTP(<function no.>,<d>,<LDN>,<d>,<d>)
ISTAT = Hardware status on return.
```

b

```
ISTAT= MAGTP(<function no.>,<d>,<LDN>,<d>,<d>)
```

c

```
ISTAT= MAGTP(<function no.>,<array name>,<LDN>,<max. words>,<words read>)
```

d

```
ISTAT= MAGTP(<function no.>,<array name>,<LDN>,<words to be written>,<d>)
           <words to be written> is rounded off to whole words
```

e

```
ISTAT= MAGTP(<function no.>,<d>,<LDN>,<density/parity>,<d>)
ISTAT=0: OK
ISTAT>0: file system error
```

f

```
ISTAT= MAGTP(<function no.>,<d>,<LDN>,<input format>,<d>)
           For <input format> see note 21) above.
```

g

```
ISTAT= MAGTP(<function no.>,<d>,<LDN>,<d>,<output format>)
```

h

```
ISTAT= MAGTP(<function no.>,<d>,<LDN>,<d>,<disk address>)
```

i

```
ISTAT= MAGTP(<function no.>,<status array>,<LDN>,<d>,<d>)
           <status array> contains 4 word status on output.
```

A P P E N D I X B

XMSG -Summary description of Functions and Parameters

Function	T-reg	A-reg	D-reg	X-reg	Comment
XFDCT					Disconnect
XFOPN	Perm. flag	=Port no			Open port
XFCLS		Port no			Close port
XFGET	XFWTF/XFWAK	Size in bytes			Get-mess-space
	(XFWTF)	=Message ptr.			
XFREL		Message ptr.			Release mess.
XFRHD		Message ptr.			Read header
		=< First six bytes of message >			
XFWHD		< First six bytes of message >			Write header
XFREA		User buffer	Max no bytes =Actual no	Displacement	Read block
XFWRI		User buffer	No. of bytes	Displacement	Write block
XFDIB	XFWOK	User buffer	Size in bytes	Message ptr	Define ind buf
XFWIB		User buffer	Size in bytes =No NOT transfered	Displacement	Write indirect
XFRIB		User buffer	Size in bytes =No NOT transfered	Displacement	Read indirect
XFMST		Message ptr			Get mess stat
	=Mess type	=< Magic number >		=Length	
XFSCM		Message ptr (-1=>task default)	Port no (if 0 then task)		Set cur mess
XFSND	Send opts.	< Magic number > =Receiver Qlen		Port	Send cur mess
XFRTN	Send opts.	Message ptr =Receiver Qlen	Bytes 0/1	Port	Return message
XFRCV	XFWTF/XFWAK	Port no			Receive mess
	=Mess type	=Mcno/Port no	=Message ptr	=Length	
XFRRH	XFWTF/XFWAK	Port no			Rec. & Read hd
	=Mess type	=Mcno/Port no	=Message ptr	=Bytes 0/1	
XFPST	XFWTF/XFWAK	Port			Port status
	=Mess type	=Mcno/Port no	=Message ptr	=Queue length	
XFGST	XFWTF/XFWAK	Port =port			Wait general RT if not wait
XFDUB		Bank number	Address in bank	Length	Def. User buf
XFSIN		=Basefield add			Service init
XFPRV		Password or 0			Privilege
XFABR		Buffer add	Length	Absolute add	Absolute read
XFABW		Buffer add	Length	Absolute add	Absolute write
XFMLK					Lock
XFMUL					Unlock
XFM2P		< Magic number > =Port no	=Machine no		Magic to port
XFP2M		Port no =< Magic number >			Port to magic
XFRIN		Port no	Machine no	Routing tab	Routing init
XFCRD	PON bit	Level =Task address	Register blk		Create driver
XFSTD		Task address			Start driver

Notes

T register holds result status (<0 if error), except for XFDCT.

Send options: Secure message, High priority, Forward, Route, Bounce (XFSEC, XFHIP, XFFWD, XFROU, XFBNC.)

Functions that affect the current port message: XFRCV, XFRRH, XFSCM.

Functions that affect the current task message: XFRCV, XFRRH, XFGET, XFREL, XFRHD, XFSCM.

If a message pointer of -1 is used, the default port message will be used if possible (port specified and a port default message available), otherwise the task default will apply.

A P P E N D I X C

XMSG -Example of a Driver using Message System

This example is an abbreviated version of the kernel of the GPIB bus driver:

```
SUBR TGPIB
      INTEGER XRMES:=3              % FIRST TWO BYTES IN XROUT MESSAGE
      INTEGER XRLNG:=6              % NEXT TWO BYTES IN XROUT MESSAGE
      INTEGER XRP01:=177404         % PARAMETER LENGTH IN XROUT MESSAGE
      INTEGER NAME:='GPIB'          % PORT NAME

TGPIB:
L:=0;T:=XFOPN;*MON 2XMSG;JPL I (WT11                    % OPEN COMMAND PORT
IF T<0 GO FAR XERR                                      % CHECK IF ERROR RETURN
A=:PORT;A:=L=:XTBLK                                     % SAVE PORT NO. AND
                                                          XTBLK ADDRESS
T:=XFOPN;CALL MCALL;A=:DPORT                            % OPEN DMA  PORT
T:=XFGET;A:=2000;CALL MCALL                             % GET DMA BUFFER
A=:DMESA;T:=B;B:=A;AD:=XMDAD;B:=T;AD=:DBUFA             % SAVE DMA BUFFER ADDRESS
A:=12=:D;A:="XRMES";X:=0;T:=XFWRI;CALL MCALL            % WRITE TO MESSAGE
T:=XFSND BONE XFROU;X:=PORT;CALL MCALL                  % NAME PORT
T:=XFRCV BONE XFWTF;A:=PORT;CALL MCALL                  % REC. RESPONS FROM XROUT
AD:=DBUFA;*EXAM                                         % READ XROUT STATUS
IF T><0 THEN ;CALL XERR FI                              % IF T><0 FATAL ERROR

GCOM:
T:=XFRCV BONE XFWTF;A:=PORT;CALL MCALL                  % WAIT FOR COMMAND
T=:CURMTY;A:=D=:CURMES                                  % SAVE CURRENT MSG. ADDRESS
                                                          & TYPE
A:=32=:D;A:=B;X:=0;T:=XFREA;CALL MCALL                  % GET PARAM. BLOCK INTO
                                                          DATAFIELD
T:=CURMES;T:=:B;AD:=XMDAD;B:=T
X:=MDATA;*RADD SX DD;COPY SA DA ADC;STD CURAD,B          % SAVE BUFFER ADDRESS
A:=CURMTY
IF A=XMROU GO FAR LOGON                                 % IF ROUTED MESSAGE GO LOGON
IF A=XMTRE GO FAR LOGOF                                 % IF RETURNED MESSAGE GO LOGOF
<Handle Request as defined in message>
```

```
RETUR:
A:=CURMTY;IF A-XMTRE=0 GO GCOM                      % IF LOGOF WAIT NEXT COMMAND
T=:CURFUN;A:=32=:D;A:=B;X:=0;T:=XFWRI;CALL MCALL % STORE STATUS IN MESSAGE
T:=DINPT;IF T><0 THEN
A:=CURBC+32;T:=CURMES;B:=T;A=:XMLEN;B:=:T  FI     % SET MESSAGE LENGTH
T:=XFSND BONE XFSEC;X:=PORT;A:=-1=:D;CALL MCALL  % RETURN MESSAGE
GO GCOM

ERR:   T=:ERCOD;GO RETUR
LOGON: % Handle Connect Request from GPIB User
LOGOF: % Handle Disconnect Request from GPIB User

SRENT: CALL WT11
MCALL: A=:AREG;A:="SRENT"=:DRIVER;A:=XTBLK:=:L=:"MRETA":=AREG
       *MON 2XMSG;JMP I (WT11
       IF T<0 GO XERR;GO MRETA
FATER: T=:ERCOD;A:=L;A-1;CALL 9ERR(£93);GO HGPIB
XERR:  T=:XERCO;A:=L;A-1;CALL 9ERR(£92)
HGPIB: FOR X:=0 TO 17 DO
         0=:UMESS(X)
       OD
       T:=XFDCT;CALL MCALL
       GO WT11

RBUS
```

A P P E N D I X D

XMSG -Symbol Table

The following is a listing of the XMSG-VALUES file that defines the symbolic Names for error codes and function values used by XMSG:

```
%*****************************************************************************
%
%       XMSG-VALUES:SYMB  Defines the values for symbolic
%       ----------------  names for functions and error codes.
%
%*****************************************************************************

@LIB SINDA-,
SYMBOL 2XMSG=200 % Monitor call number for XMSG (in SIN-DATA)
@ELIB

%         F U N C T I O N   V A L U E S

SYMBOL XFDUM=0   % Dummy function
SYMBOL XFDCT=1   % Disconnect from message system
SYMBOL XFGET=2   % Get message space
SYMBOL XFREL=3   % Release message space
SYMBOL XFRHD=4   % Read header from a message (6 bytes)
SYMBOL XFWHD=5   % Write header to a message (6 bytes)
SYMBOL XFREA=6   % Read from message to user buffer
SYMBOL XFWRI=7   % Write from user to message
SYMBOL XFSCM=10  % Set current message
SYMBOL XFMST=11  % Get message status
SYMBOL XFOPN=12  % Open port
SYMBOL XFCLS=13  % Close port
SYMBOL XFSND=14  % Send message to a remote port
SYMBOL XFRCV=15  % Receive a message on a given port
SYMBOL XFPST=16  % Get local port status
SYMBOL XFGST=17  % General status or wait

       % SERVICE FUNCTIONS

SYMBOL XFSIN=20  % Service initialisation function
SYMBOL XFSRL=21  % Service release function (obsolete)
SYMBOL XFABR=22  % Absolute read block from POF area
SYMBOL XFABW=23  % Absolute write block to POF area
SYMBOL XFMLK=24  % Lock message system
SYMBOL XFMUL=25  % Unlock message system
SYMBOL XFM2P=26  % Magic number to port id.
SYMBOL XFP2M=27  % Port to magic number
SYMBOL XFRIN=30  % Routing initialise (obsolete)
SYMBOL XFCRD=31  % Create driver with context
SYMBOL XFSTD=32  % Start driver
```

```
        % INDIRECT BUFFER HANDLING FUNCTIONS

SYMBOL XFDIB=33  % Define indirect buffer
SYMBOL XFRIB=34  % Read from indirect buffer
SYMBOL XFWIB=35  % Write to indirect buffer

        % FUNCTIONS ADDED AFTER THE FIRST RELEASE

SYMBOL XFPRV=36  % Request privilege
SYMBOL XFRTN=37  % Write word 0 and return message
SYMBOL XFRRH=40  % Receive message and read word 0
SYMBOL XFDUB=41  % Define user buffer area for current message
SYMBOL X5FUN=42  % ** END MARKER ** LEAVE ME HERE PLEASE

% BIT VALUES IN FUNCTION CODE REGISTER (T-REG)

SYMBOL XFWTF=17  % If set then wait if operation not terminated
SYMBOL XFWAK=16  % In RCV/PST/GST: Do RTENTRY on status change
SYMBOL XFPRM=15  % In XFOPN: Permanent open requested
SYMBOL XFOPS=14  % In XFOPN: Specified port number required (not impl.)
SYMBOL XFPON=15  % In XFSTD: Driver to run with paging on
SYMBOL XFWOK=15  % In XFDIB: Allow write access to indirect buffer
SYMBOL XFHIP=15  % In XFSND: High-priority message
SYMBOL XFBNC=14  % In XFSND: Bounce message
SYMBOL XFFWD=13  % In XFSND: Forward message
SYMBOL XFROU=12  % In XFSND: Message to be sent to local XROUT
SYMBOL XFSEC=11  % In XFSND: Secure message (Return if not deliv'd)
% *** Warning: bits 10, 11 (octal) are used for bank no in XFABR, XFABW ***

%
% MESSAGE TYPES: RETURNED AS SUCCESSFULL STATUS FROM XFRCV

SYMBOL XMTNO=1   % Normal message
SYMBOL XMROU=2   % Routed message (Via XROUT)
SYMBOL XMTHI=3   % High priority message
SYMBOL XMTRE=4   % Return message (Abnormal condition)
SYMBOL XMKIK=5   % XROUT has been kicked (no message)
SYMBOL XMTPS=6   % Pseudo message (not used)
```

```
%
%       U S E R     E R R O R     S Y M B O L S    (Returned in T-reg)
%
SYMBOL XENOT=-1  % No more XT-blocks free
SYMBOL XEIRM=-2  % Non-local remote port illegal here
SYMBOL XETMM=-4  % Task is not allowed any more memory
SYMBOL XENIM=-5  % Facility not yet implemented
SYMBOL XEIBP=-6  % Illegal message buffer pointer
SYMBOL XEBNY=-7  % Message buffer not yours
SYMBOL XEISP=-10 % Illegal service program calling
SYMBOL XENOP=-11 % No more ports available
SYMBOL XEIDR=-12 % Function not available to drivers
SYMBOL XENDM=-13 % No default message
SYMBOL XEMCH=-14 % Message is already chained
SYMBOL XEBFC=-15 % Message is in a queue.
SYMBOL XEAIN=-16 % XMSG Kernel already initialised
SYMBOL XECRA=-17 % XMSG crash (Info in Basefield)
SYMBOL XEWNA=-20 % Write Not Allowed (Indirect buffer)
SYMBOL XENVI=-21 % No Valid Indirect buffer defined
SYMBOL XEILF=-22 % Illegal function code in monitor call
SYMBOL XEIMA=-23 % Invalid magic number
SYMBOL XEMFL=-24 % Message space full
SYMBOL XEILM=-25 % Illegal message size
SYMBOL XEIPN=-26 % Illegal port number
SYMBOL XEPRV=-27 % Privileged function called without privilege.
SYMBOL XEPVR=-30 % Privilege request refused
SYMBOL XERNA=-31 % Remote machine not available
SYMBOL XEROV=-32 % Remote task space overflow
SYMBOL XEXBF=-33 % Message already has XMSG buffer (XFDUB)
SYMBOL XELOK=-34 % XMSG locked
SYMBOL XENDP=-35 % No port open (so 'default port' param invalid)
SYMBOL XEITL=-36 % Illegal transfer length for read/write
SYMBOL XEIDP=-37 % Illegal displacement in read/write
SYMBOL XEILR=-40 % Illegal use of reentrant segment in XFDIB
SYMBOL XENOS=-41 % Indirect Buffer not on valid segment
SYMBOL XENSE=-42 % Network sequencing error
SYMBOL XERND=-43 % Remote machine not defined
```

```
%
%       X R O U T   -   S E R V I C E   V A L U E S
%
SYMBOL XSNUL=100  % Null command returns 0 status to sender
SYMBOL XSLET=101  % Send a letter
SYMBOL XSNAM=102  % Give name to this port
SYMBOL XSCNM=103  % Clear name of this port
SYMBOL XSGNM=104  % Get name of port (param: MAGNO)
SYMBOL XSGNI=105  % Get name (param: MC/PORTNO)
SYMBOL XSREM=106  % Get remote magic number (LOC, REM NAME, MC)
SYMBOL XSGMG=107  % Get magic number (PRIV)
SYMBOL XSCMG=110  % Clear magic number (PRIV)
SYMBOL XSDRN=111  % Define remote name (PRIV)
SYMBOL XSDMC=112  % Define routing for machine N (PRIV)
SYMBOL XSGMC=113  % Get routing info for machine N
SYMBOL XSLKI=114  % Start up specified link
SYMBOL XSTIN=115  % Initialise tracing (open file, ..)
SYMBOL XSTCL=116  % Close tracing
SYMBOL XSTDC=117  % Define tracing conditions
SYMBOL XSCRS=120  % Create connection port
SYMBOL XSNSP=121  % Increment number of free connections
SYMBOL XSMAX=XSNSP % Maximum legal service value
%
%       X R O U T   -   E R R O R S
%
%       Error values returned in byte 1 of return message.
%
SYMBOL XRISN=1   % Illegal service number
SYMBOL XRUNN=2   % No open port has this name
SYMBOL XRDDF=3   % Another port already has this name
SYMBOL XRNSP=4   % No space left for names
SYMBOL XRIPT=5   % Illegal parameter type
SYMBOL XRMMP=6   % Missing mandatory parameter
SYMBOL XRUNM=7   % Unknown magic number
SYMBOL XRMTL=10  % Resulting message too long
SYMBOL XRSMF=11  % Standard message format not handled
SYMBOL XRPRV=12  % Caller was not privileged
SYMBOL XRIMC=13  % Illegal machine number parameter
SYMBOL XRNRO=14  % Cannot access remote XROUT
SYMBOL XRICL=15  % Illegal cluster number parameter
SYMBOL XRIPI=16  % Illegal PIOC number parameter
SYMBOL XRNXM=17  % Invalid service request - no multi-mc XMSG
SYMBOL XRILN=20  % Illegal/Reserved Log. unit no. for link
SYMBOL XRNXL=21  % No more XL-Blocks (Link Descriptors)
SYMBOL XRNXD=22  % Not enough XD-Blocks for LKINI
SYMBOL XRNTR=23  % No trace generated
SYMBOL XRTRA=24  % Trace already active
SYMBOL XRTRP=25  % Trace passive
SYMBOL XRTFE=26  % Trace file open error (see param 1)
SYMBOL XRTRT=27  % Trace RT-prog (XTRACE) not found
SYMBOL XRTIS=30  % Illegal system number
SYMBOL XRBLK=31  % Bad link - open unsuccessful
SYMBOL XRMCD=32  % Attempt to redefine local machine no
SYMBOL XRNLM=33  % Local machine number not yet defined
SYMBOL XRTRE=34  % Too many remote names to this machine
SYMBOL XRRNA=35  % Old letter calls (service 2) cannot use XMSGM
```

```
SYMBOL XRBUS=36  % All connections with this name busy
SYMBOL XRNSE=37  % This is not a connect port
SYMBOL XRRPN=40  % Remote port statically declared.


%   XMSG Crash Codes (on System Console and saved in Basefield)


SYMBOL XXEIE=1   % Illegal entry ptr to XCRMG
SYMBOL XXIOW=2   % Illegal owner of buffer
SYMBOL XXBIN=3   % Memory allocn. inconsistency
SYMBOL XXMCE=4   % Message queue length inconsistency
SYMBOL XXIEN=5   % ZRALL gave port not in XQTAB
SYMBOL XXIFL=6   % INIT: ZFUNC Function >XFMX1
SYMBOL XXIRT=7   % Illegal RT-Description add used.
SYMBOL XXNBF=10  % INIT: No Buffer space available
SYMBOL XXRIN=11  % Inconsistency in resource allocation
SYMBOL XXNMM=12  % More memory released than owned
SYMBOL XXNIM=13  % Not implemented (Cannot recover)
SYMBOL XXCLS=14  % Inconsistency in port chain in CLOSE
SYMBOL XXCHE=15  % Double chaining attempted
SYMBOL XXNOR=16  % No XMSG-RESIDENT found by POF
SYMBOL XXICM=17  % Inconsistency in XMPRT/XPCMS Pair
SYMBOL XX100=20  % This can only be done on ND-100´S
SYMBOL XXMON=21  % Inconsistency in level 5 monitor queues
SYMBOL XXMMC=22  % Multimachine XMSG not implemented/generated
SYMBOL XXFBI=23  % Frame buffer handling inconsistency
SYMBOL XXPER=24  % Protocol error in communications system
SYMBOL XXILN=25  % Illegal LOG NO for HDLC (bad LOGPH)
SYMBOL XXROU=26  % No legal routing port defined
SYMBOL XXHER=27  % Error in HDLC Driver or interface to it
SYMBOL XXRO2=30  % Fatal error in XROUT - see XROUT basefield
SYMBOL XXTAS=31  % Task handling (wait,resume) error
@DEV 1
```

A P P E N D I X E

HDLC -Error Codes from the Monitor Call HDLC

These errors (octal) are related to the DCB transfer part. The A-register contains the code on a non-skip return.

ERROR CODE	MEANING
-1	The LDN is not reserved by the calling program
-2	Illegal LDN used. Not known by SINTRAN
-3	No DCB in receiver queue
-4	No vacant buffer for DCB
-5	Illegal DCB-usize
-6	Illegal LDN. Not to be used by MON HDLC
-7	DCB-msize less than DCB-usize
-10	Illegal function
-11	Fatal error. The table is inconsistent

A P P E N D I X F

HDLC -Status Codes in the DCB

These codes (octal) are issued by the driver, and found in the STATUS word in the DCB when returned from the driver.

STATUS	MEANING
0	Operation completed sucessfully
100	Interface not cleared before initiation
101	Interface not initiated before transfer
102	Underrun
103	Timeout, no output interrupt
104	Command timeout. Probably a hardware error.
105	Illegal command used in DCB
106	Hardware failure in initiation
107	DCB overflow. Receiver list is full
110	Untreated DCB due to Device Clear or Reset
111	Input LDN stopped. Possible errors are over-run, crc-error or lack of buffer space.
112	Illegal parameter in DCB
113	Frame data part of DCB greater than max. frame-size.
114	DCB is too small for expected data or info.
115	Attempt to transfer a frame containing less than 2 bytes
116	Connection broken by X.21
117	Illegal displacement specification
120	Link locked

A P P E N D I X G

HDLC -Example of use

```
        PROGRAM HDLCT,30
C
C       THE PROGRAM SENDS FRAMES OF DATA TO A REMOTE COMPUTER, AND
C       ASSUMES THAT THE DATA IS RETURNED UNALTERED.
C       THE REMOTE MACHINE MAY BE SIMULATED BY OPERATING THE
C       INTERFACE IN MAINTENANCE MODUS.
C


C
        INTEGER HDLC
        INTEGER IDCB(106)
        INTEGER SDCB,RDCB,OLUN,ILUN,BC
        INTEGER DEVCL,DEVINI,TRANS,LUNSTA
        INTEGER FRSIZ,DCBSIZ,MAINT,MAXERR,LISTM
        INTEGER CFRSIZ,CMAXER
        INTEGER FDPLX,COUNT,MAXDCB
        INTEGER COMAND,STAT,HSTAT,FRDAT,MODUS
C
C       HDLC PARAMETER DEFINITIONS
C
        SDCB = 0
        RDCB = 1
        OLUN = 753
        ILUN = 752
C
C       DCB ARGUMENT SYMBOLES
C
        COMAND = 1
        STAT       = 2
        HSTAT  = 3
        FRDAT  = 4

        MODUS  = 4
        FRSIZ  = 5
        MAXERR = 6
        LISTM  = 8
C
C       DCB ARGUMENT VALUES
C
        TRANS = 1
        DEVCL = 3
        DEVINI = 4
        LUNSTA = 5
        CFRSIZ = 100
        DCBSIZ = CFRSIZ+6
        MAINT  = 2
        CMAXER = 2
        COUNT = 0
C
C       RESERVE LDN
C
        CALL RESRV (OLUN,0,0)
        CALL RESRV (ILUN,0,0)
C
C       SEND DEVICE CLEAR TO INTERFACE
```

```
C
        IDCB(COMAND) = DEVCL
        ISTATE = HDLC (SDCB,OLUN,IDCB,6,6)
        IF (ISTATE .LT. 0) GO TO 2000
C
C       GET DRIVER RESPONSE TO DEVICE CLEAR
C
        ISTATE = HDLC (RDCB,OLUN,IDCB,BC,1)
        IF (ISTATE .LT. 0)GO TO 2000
        IF (IDCB(STAT) .NE. 0) GO TO 3000
C
C       DEVICE CLEAR SUCCESSFULLY DONE, DO DEVICE INITIALIZATION
C
        IDCB(COMAND) = DEVINI
        IDCB(MODUS) = MAINT
        IDCB(FRSIZ) = CFRSIZ
        IDCB(MAXERR) = CMAXER
        ISTATE = HDLC (SDCB,OLUN,IDCB,12,12)
        IF (ISTATE .LT. 0) GO TO 2000
C
C       GET DRIVER RESPONSE TO DEVICE INITIALIZATION
C
        ISTATE = HDLC (RDCB,OLUN,IDCB,BC,1)
        IF (ISTATE .LT. 0) GO TO 2000
        IF (IDCB(STAT) .NE. 0) GO TO 3000
C
C       DEVICE INITIALIZATION SUCCESSFULLLY COMPLETED.
C       START THE RECEIVER (INPUT) PART OF THE INTERFACE.
C       TO PREVENT OVERRUN SUPPORT THE DRIVER WITH SOME BUFFER SPACE
C       FIRST WE HAVE TO FIND THE MAXIMUM BUFFER SPACE (NUMBER OF
C       IDCBS) THE DRIVER MAY HANDLE.
C
        IDCB(COMAND) = LUNSTA
        ISTATE = HDLC (SDCB,ILUN,IDCB,20,20)
        IF (ISTATE .LT. 0) GO TO 2000
        ISTATE = HDLC (RDCB,ILUN,IDCB,BC,1)
        IF (ISTATE .LT. 0) GO TO 2000
        IF (IDCB(STAT) .NE. 0) GO TO 3000
        MAXDCB = IDCB(8)
C
C       THE MAXIMUM NUMBER OF DCBS HELD BY INPUT DRIVER IS "MAXDCB"
C       SO GIVE THEM TO HIM
C
        IDCB(COMAND) = TRANS
        DO FOR I = 1,MAXDCB
                  ISTATE = HDLC (SDCB,ILUN,IDCB,6,DCBSIZ)
                  IF (ISTATE .LT. 0 ) GO TO 2000
        ENDDO
C
C       SEND A FRAME OF DATA TO REMOTE COMPUTER
C
900     COUNT=COUNT+1
        IDCB(4)=COUNT
        IDCB(COMAND) = TRANS
        ISTATE = HDLC (SDCB,OLUN,IDCB,DCBSIZ,DCBSIZ)
        IF (ISTATE .LT. 0) GO TO 2000
C
```

```
C       ANY REACTION FROM THE DRIVER? IF NOT CALL RTWAIT
C
C       FIRST CHECK INPUT PART
C
1000      ISTATE = HDLC (RDCB,ILUN,IDCB,BC,0)
          IF (ISTATE .GT. 0) THEN
                    IF (IDCB(STAT) .EQ. 0) THEN
                              IF (IDCB(4) .NE. COUNT) GO TO 4000
                              IDCB(COMAND) = TRANS
                              ISTATE = HDLC (SDCB,ILUN,IDCB,6,DCBSIZ)
                              IF (ISTATE .LT. 0) GO TO 2000
                              GO TO 900
                    ELSE
                              GO TO 3000
                    ENDIF
          ELSEIF (ISTATE .NE. -3) GO TO 2000
          ENDIF
C
C       SO, CHECK THE OUTPUT PART
C
          ISTATE = HDLC (RDCB,OLUN,IDCB,BC,0)
          IF (ISTATE .GT. 0) THEN
                    IF (IDCB(STAT) .EQ. 0) THEN
                              GO TO 1000
                    ELSE
                              GO TO 3000
                    ENDIF
          ENDIF
          IF (ISTATE .NE. -3) GO TO 3000
          CALL RTWT
          GO TO 1000
C
C       ERROR IN DCB TRANSFER
C
2000      CALL ERMON (2H50,ISTATE)
          GO TO 9999
C
C       DRIVER ERROR
C
3000      CALL ERMON (2H51,IDCB(STAT))
          GO TO 9999
C
C       ERROR IN DATA TRANSFER
C
4000      CALL ERMON (2H52,IDCB(4))
C
9999      END
;
```

A P P E N D I X H

X.21 -Facility Bits

In the facility parameter each bit (FACILITY BIT) has a special meaning. By setting a specific bit, the corresponding facility will be requested.

FACILITY BIT	FACILITY REQUESTED
0	Charging requested
1	Called line identification
2	Direct call
3	Connect when free

A P P E N D I X I

X.21 -Call Progress Signals

CODE	FUNCTION
00	Reserved for further use
01	Terminal called
02	Redirected call
03	Connect when free
20	No connection
21	Number busy
22	Selection signal Procedure error
23	Selection signal Transmission error
41	Access barred
42	Changed number
43	Not obtainable
44	Out of order
46	Uncontrolled not ready
47	DCE power off
48	Invalid facility request
49	Network fault in local loop
50	Controlled not ready
51	Call information service
52	Incompatible user class of service
61	Network congestion
71	Long term network congestion
72	RPOA out of order
81	Registration/Cancellation confirmed
82	Redirection activated
83	Redirection deactivated

From the user's point of view, group 0 means wait; group 2 and 6 mean try again, next try may result in a call set up; groups 4 and 5 and 7 mean there is no reason for a new try, because the answer will be the same for a longer time. As group 8 is the result of a procedure between the DTE and the network, no further action needs to be taken.

A P P E N D I X J

Error Codes

These errors (octal) are related to the DCB transfer part. The code will be in the A-reg when the monitor call X.21 gives non skip return.

ERROR CODE	FUNCTION
-1	The LDN is not reserved by the calling program
-2	Illegal LDN used. Not known by SINTRAN
-3	No DCB in receiver queue
-4	No vacant buffer for DCB
-5	Illegal DCB-usize
-6	Illegal LDN. Not to be used by MON X.21
-7	DCB-msize less than DCB-usize
-10	Illegal function

A P P E N D I X K

X.21 -Status Codes in the DCB

The codes (octal) are given by the Driver, and found in the STATUS word in the DCB when returned from the driver.

STATUS	FUNCTION
0	Operation sucessfully completed
1	Too small message for appropriate information
2	Illegal LDN used in CONNECT
3	Illegal command
4	Illegal command in data phase
5	Different hardware device numbers for ILDN and OLDN
6	ILDN or OLDN not reserved
7	No matching ident entry found in ident table
10	Network error, is modem power on? (state 1)
11	No incomming call, ready state terminated.
12	No LDN previously connected
13	Missing terminator in selection signals
14	"No charge" received for last call
15	Call progress signals received
16	Multiple call progress signals received
17	X.21 LDN already connected
20	Call terminated
21	Call with facility "CONNECT WHEN FREE" unsuccessfully terminated
32	Network timout (in state 2)
33	Network timout (in state 3)
35	Network timout (in state 5)

Numbers shown as x) are references to notes shown on next page

<——— Function ———> Code.	Name	C. F. 1)	<——mag. tape——> Tandb. Pertec	Hew- Pack.	STC	Phil- lips Cass.	Versa- tec 25)	Floppy Disk 26)	NORNET commun chan.	Dynamic Logical Dev. No
0	Read record	c	x	x	x	x		x	x	x
1	Write record	d	x	x	x	x	x	x	x	x
2	Read odd numbers of bytes	c	x	x	x					
3	Loop write to read in FCU				x					
4	Read one rec. backwards				x					
5	Unlock and stop	b				x				
6	Lock cassette	b				x				
7	Erase tape	b				x				
10	Advance through EOF 29)	b	x	x	x	x		x		
11	Reverse through EOF 30)	b	x	x	x			x		
12	Write EOF	b	x	x	x			x 17)		
13	Rewind	b	x	x	x			x 18)		
14	Write 4 inch erase gap	b	x	x	x					
15	Backspace rec.	b	x	x	x			x 19)		
16	Advance rec.	b	x	x	x			x 20)		
17	Unload	b	x	x	x					
20	Read status 27)	a	x 11)	x 12)	x 11)	14)	x 15)			
21	Clear device	b	x 2)	x 2)	x 2)		x	x 3)		
22	Clear device w/error exit	a	x	x	x		x			
23	Select parity and density,or Select density 4)	e	x 5)		x 6)					
24	Read last Status 28)	a	x 11)	x 12)	x 11)	14)	x 15)	x 16)		
25	Read tape status	i	x	x	x					
26	Read byte rec.	c	x	x	x					
27	Write byte rec.	d	x	x	x					

<——— Function ———> Code.	Name	C. F. 1)	<——mag. tape——> Tandb. Pertec	Hew- Pack.	STC	Phil- lips Cass.	Versa- tec 25)	Floppy Disk 26)	NORNET commun chan.	Dynamic Logical Dev. No
30	Set alphanum mode	b					x			
31	Set graphic mode	b					x			
32	Give form feed	b					x			
33	Clear selected unit	b			x 7)					
34	Set diagnostic mode				x					
35	n.a.									
36	n.a.									
37	n.a.									
40	Set floppy form	f						x 21)		
41	Format floppy	b						x 22)		
42	A) read density B) read parity and density C) read format	g	x 9)		x 8)			x 10)		
43	Read deleted record	c						x 23)		
44	Write deleted record	d						x 24)		
45	n.a., 13)							(x)		
46	Get current disk address	h						x		

A P P E N D I X L

Index

This index includes terms which are not complete headings. For names of commands and monitor calls, the reader should first check with the table of contents.

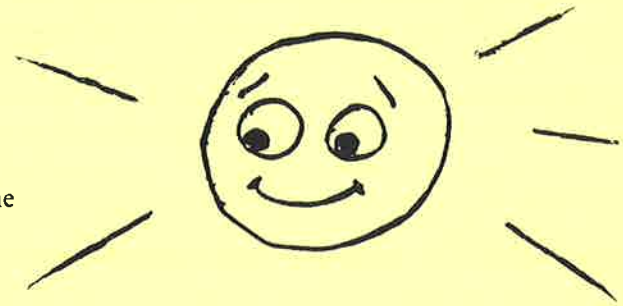
* * * * * * * * * * SEND US YOUR COMMENTS!!! * * * * * * * * * *

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 Communication Guide Manual number: ND–60.134.02

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

Do you have suggestions for improving this manual? _____

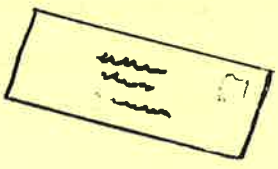
Your name: _____ Date: _____

Company: _____ Position: _____

Address: _____

What are you using this manual for? _____

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



Norsk Data's answer will be found on reverse side

Answer from Norsk Data _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

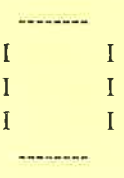
_____

_____

_____

_____

Answered by _____ _____ _____ _____ Date _____



Norsk Data A.S.

Documentation Department

P.O. Box 4, Lindeberg Gård

Oslo 10, Norway