Title:

Terminal Access Module (TEM) (3rd Edition)

User's Guide/Reference Manual/Installation Guide

§ REGNECENTRALEN

af 1979

Keywords:

RC8000, RC4000, monitor, terminal access, multiplexing, spooling.

Abstract:

TEM is a service module, which on behalf of applications supports accessing of terminals.

This manual contains information of interest for the application programmer, the operator and the system staff.

(46 printed pages)

FOREWORD

First edition: RCSL No 31-D481.

Second edition: RCSL No 31-D513.
The manual describes revision 2 of the terminal access module
TEM.

No differences in functions and formats have been made, but new
facilities have been implemented. The extensions concern mainly
interface functions to the format 8000 system.

The TEM system has been designed and implemented by the authors.

This manual replaces the description of revision 1, January 1978.

Niels Møller Jørgensen
A/S REGNECENTRALEN, October 1978

Third edition: RCSL No 31-D689.
The manual describes revision 3 of the terminal access module
TEM.

TEM has been extended with new facilities for format 8000 device
control messages implemented in IBM 3270 Terminal Handler version
2 and entering of passwords in nondisplay mode as implemented in
Basis system (SW8001/1 release 4.2 and SW8001/2 release 1).

The changes are indicated by correction lines in left margin.

This manual replaces the description of revision 2 (RCSL No
31-D513).

Flemming Biggas
A/S REGNECENTRALEN af 1979, March 1983

TABLE OF CONTENTS   PAGE

1.   INTRODUCTION   1.

This manual contains information about the service module TEM.

Relevant information may be found by people, who are going to use, install, generate or work as operators for TEM.

The purpose of TEM is to support access of terminals. TEM aims specifically at multiplexing of terminal input and output to and from an application. In addition TEM offers spooling of data in order to smooth speed differences between an application program and (slow) external devices.

The facilities of TEM will make programming of on-line systems easier, because TEM allows an application written in a higher programming language to access a number of terminals through one stream by means of the standard input/output system included in the language. Besides TEM operates the devices in parallel with the application processing transactions, which will ensure a higher degree of service to the terminals and a better utiliza-tion of hardware, than if the accessing was performed by the application itself.

This manual contains the following information:

Chapter 2 describes the functions of TEM. The formats of the in-formation exchanged between TEM and an application program are also specified.

Chapter 3 describes the start up procedures and how to operate TEM in the day to day running.

Chapter 4 is a guide in system generation. In this chapter the procedures concerning installation and trimming of TEM is de-scribed.

2. REFERENCE MANUAL

2.1 Some TEM Concepts

2.1.1 Terminal Pools and Terminal Links

When an application program wants to access a set of devices via
TEM, it asks TEM to define a terminal pool. For every device the
application wants to use, a terminal link must be created between
the pool and the external process corresponding to the device.
Accessing of devices is performed by TEM on request from the
application. The mode of operation is partly defined in the link
creation phase and partly by explicit input/output commands to a
pseudo process corresponding to the pool.

2.1.2 Names of Pools

The name of a pool must obey the naming rules defined in the
monitor, ref. [3]. These rules state that a name is a textstring
consisting of 12 ISO characters beginning with a small letter
followed by at most 10 small letters or digits, terminated by
NULL characters.

2.1.3 Types and Names of Terminal Links

When a link from a pool to a device is created, the corresponding
external process must be specified. The process kind may be of
any type, but in the present version of TEM only the following
access protocols are supported:

1) TTY compatible processes.

For this protocol TEM offers two access modes:

a) link type = 0
   TTY multi-terminal.
   TEM offers spooling and multiplexing facilities.

b) link type = 2

   TTY single terminal.

   Only spooling facilities.

2) Format 8000 termin and termout processes ref. [5]

   link type = 4

   Multiplexing and spooling is supported.

A link is identified by a name (local id), consisting of 24 bits (3 bytes). This name is defined by the application, and must be unambiguous within the pool.

2.1.4   Blocks and Transactions   2.1.4

a) Type = 0 (TTY multi-terminal)

   1 transaction = 1 block

local id	data
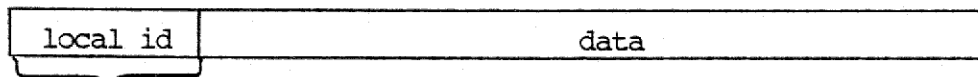
   3 bytes

b) Type = 2 (TTY single terminal)

   1 transaction = 1 block

data
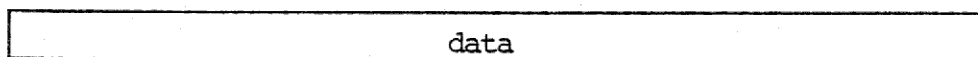
c) Type = 4 (termin and termout processes)

   1 transaction = n blocks   n >= 1

CU	DEV	DATA	...	DATA	...		ETX

1 byte 1 byte   1 byte

ref. [4]

2.1.5   Multiplexing
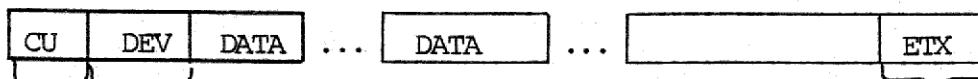
TEM is able to handle a number of external processes for a number
of applications (internal processes). An external process may not
be included in more than one pool at a time. When an application
sends output to a pool, the link is addressed as part of the
transaction. Depending on the linktype the address information is
stripped off or altered before the transaction is sent to the
device. Correspondingly address information is added to input
before data is delivered to the application.
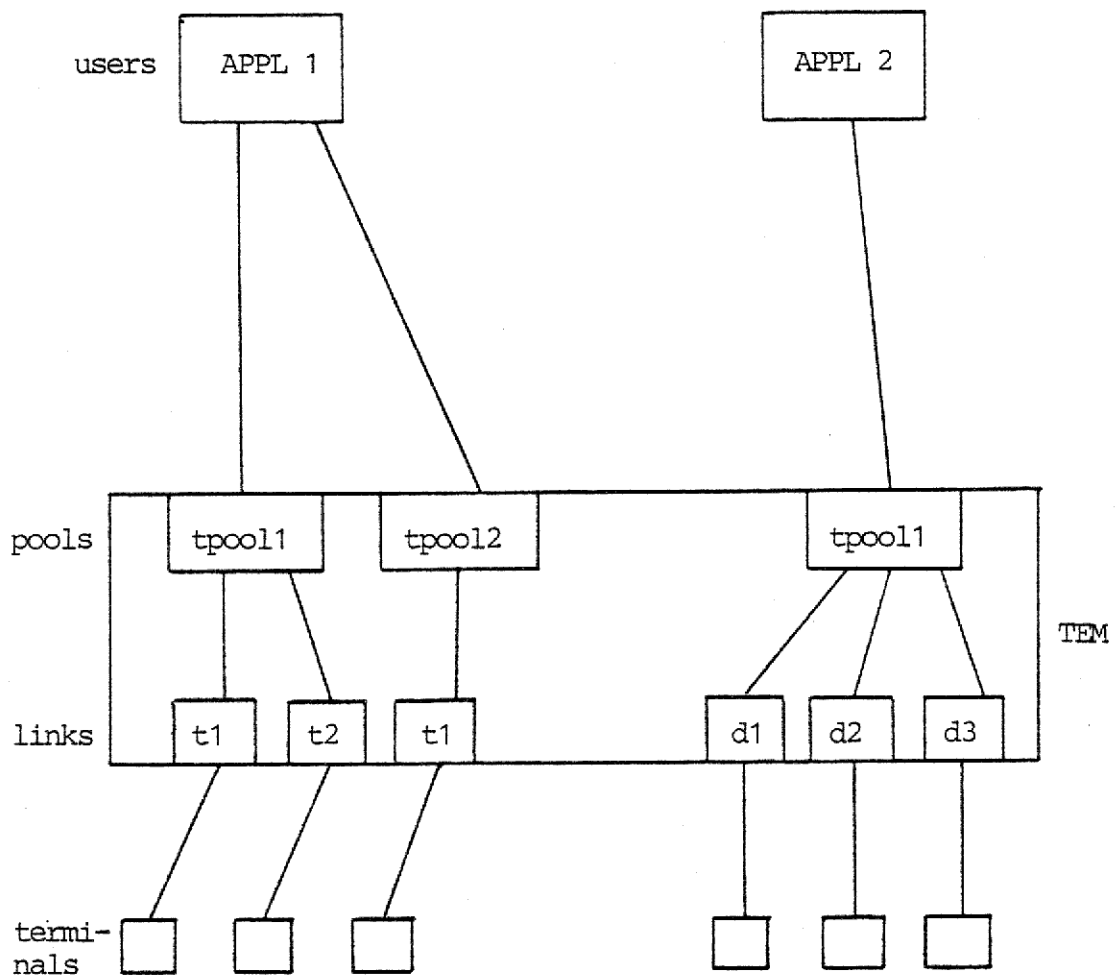
Figure 1: A TEM Configuration.

Spooling

In order to equalize differences in speed between the user pro-
cess and the relative slow devices, TEM provides spooling of
input and output. This spooling is done partly in primary store
and partly on backing store. Every link and pool has its own
spool queue with a maximal size which is defined when TEM is
installed.

The queues contain operations not yet performed. The link (queue)
accomodates input/output operations and output data, and the pool
(queue) accomodates answers to input operations and input data.

TEM Operations

An application uses TEM by calling the operations listed in the
next subsections. This is done by means of the send message/wait
answer procedures of the monitor, ref. [1].

The operations are divided in the two groups:

1) Control operations.
2) Input/output operations.

The conventions for input and output operations follow the stan-
dards defined in the monitor, ref. [1], while the control oper-
ations are designed especially for TEM.

Control Operations

Operations concerning pools are activated by sending a message to
TEM while link operations are sent to a pseudo process with the
same name as the corresponding pool. This pool is made by TEM
when the pool is created.

The first word of the answer from a control operation contains a
status mask, indicating the result of the operation. The status

word is only defined when the result of the answer is normal (W0 = 1). Dummy answers are delivered in the following situation:

Result = 3   unintelligible message

Interpretation of the status bits is defined below:

bit	meaning
13	pool exists or pseudoprocess does not exist and cannot be created
14	link exists
15	pool does not exist
16	link does not exist
17	no free pool
18	no free link
19	terminal in existing link
20	terminal not in existing link
21	terminal unknown
22	–
23	–

2.2.1.1   Create Pool   2.2.1.1

message :
  + 2:
  + 4
  + 6
  + 8
    :
  +14

| 90 | 0 | receiver TEM →
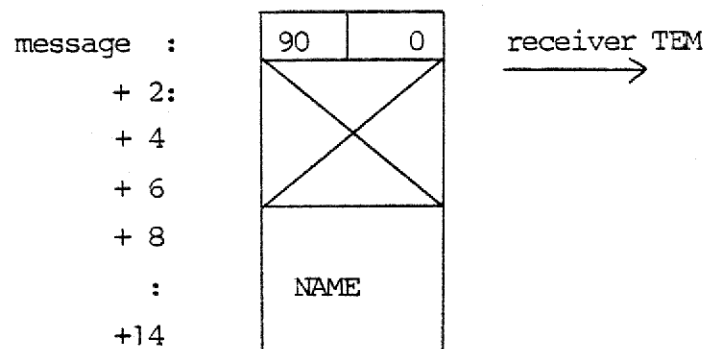
NAME

The operation creates a terminal pool with the sending process as exclusive user. A pseudoprocess with the name specified in message (8:15) is created. It is legal to define TEM as the pool. In this case no pseudoprocess is created, but TEM itself will act as receiver of messages concerning the pool.

N.B. In connection with the creation TEM sends a message with operation code (halfword 0) equal to -2. Answer on this message will cause the same actions as a call of the operation remove pool.

Possible status bits: 13, 17.

2.2.1.2   Remove Pool   2.2.1.2

```
message  :      | 92  |   0 |     receiver TEM
      + 2:      |      \  / |     ────────────→
      + 4       |       \/  |
      + 6       |       /\  |
      + 8       |      /  \ |
        :       |   NAME    |
      +14       |           |
```
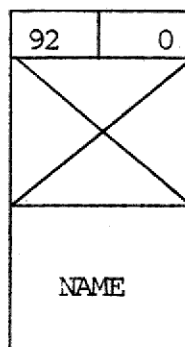
The operation removes the terminal pool.

Possible status bits: 15.

2.2.1.3   Lookup Pool   2.2.1.3

```
message  :      | 94  |   0 |     receiver TEM
      + 2:      |      \  / |     ────────────→
      + 4       |       \/  |
      + 6       |       /\  |
      + 8       |      /  \ |
        :       |   NAME    |
      +14       |           |
```
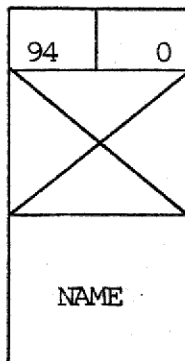
If the sender is user of the pool the answer contains:

```
answer    :    | status        |
                |  \         /  |
   + 2:         |   \       /   |
   + 4          |    \     /    |
   + 6          |     \   /     |
   + 8          |      \ /      |
                |      / \      |
  +10           | block full    |
  +12           | halfword free |
  +14           |  \         /  |
                |   X         X |
```

'block full' is the number of input blocks spooled for the pool, i.e. the number of blocks read by TEM, but not delivered to the user yet. 'halfword free' is the number of halfwords left for further input spooling.

Possible status bits: 15.

2.2.1.4   Create Link

```
message  :    | 100   | type    |     receiver terminal
   + 2:        | local id.       |
   + 4:        | ext. proc.      |     ————→ pool
               | descr. adr.     |
   + 6:        | bufs  | timer   |
   + 8:        | mask  | subst.  |
```

The operation includes a new terminal in a terminal pool. The local name of the link is stated in message (2:3). The terminal is identified by the process description address (message (4:5)).

'bufs' (message (6)) is the maximal number of spooled indata transactions. When the user asks for input, a number of input operations are initiated on all links with fewer input transactions spooled than defined by 'bufs'. If the link represents a 'termin' (format 8000) process the value is recommended to be: <no of terminals> * 8.

'timers' (message (7)) states the maximum number of timer pe-
riods, which may pass before the application is answered. I.e.
the user may extend the timerperiod for the device n times rela-
tive to standard, by setting message (7) to n-1.

'mask' and 'subst' are used in connection with format 8000 links
only. Explanation is given below.

Input and output operations are queued for the link and executed
in order of arrival.

Depending on the link type input and output are handled as de-
scribed in the following:

1) Type = 0   (TTY multiterminal)

   a) Output.
      A transaction matches the link if the first 3 bytes (24
      bits) of the transaction equals the local identification of
      the link.

      Before the transaction is sent to the device, the address
      information is stripped off.

   b) Input.
      The input transaction sent to the user is the local iden-
      tification (3 bytes) concatenated with the block received
      from the device. If the data is not terminated with the ISO
      character NL (value = 10), 3 bytes are inserted at the end
      of the transaction: NL NUL NUL.

2) Type = 2   (TTY single terminal)

      Input and output are spooled in TEM and routed between the
      user and the external process representing the device
      without modification of data.

3) Type = 4   (format 8000 termin and
                                     termout)

   a) Output.

      A transaction matches the link if:

      (extend CU) and mask = subst and mask

      Before the transaction is sent to the device the CU byte is
      changed:

         CU:= CU and (-,mask)

   b) Input.

      The CU-byte is modified, before the transaction is sent to
      the user:

         CU:= CU or (mask and subst) extract 8.

      One should notice, that while the length of CU is 8 bits,
      mask and subst are 12 bits, and when searching for the link
      in connection with output operations, 12 bits are compared,
      as CU is extended with 4 bits equal to 0.

      The purpose of this is the following:
      If a pair of links (termin, termout) is connected to the
      same pool, the value of CU received by the application from
      the input link ought to be returned unchanged on the output
      link. I.e. that mask (4:11) and subst (4:11) for these
      links should be equal.

      To ensure that the output operations are really sent on to
      the output link, one must in the call of the create link
      operation for the termin process put (mask (0:3 and subst
      (0:3)) <> 0 and for the termout process put (mask (0:3) and
      subst (0:3)) = 0.

      For the termout link 'bufs' must be equal to 0, while typi-
      cally the termin link should be multibuffered.

It is the responsibility of the user to ensure that the addres-
sing of output is unambiguous. This should be noticed particu-
larly if links of different types are connected to the same
pool.

Possible status bits: 14, 15, 18, 19, 21.

2.2.1.5   Remove Link   2.2.1.5

```
message  :      | 102        0/1 |   receiver terminal
       + 2:     | localid        |            pool
                |       X        |
```
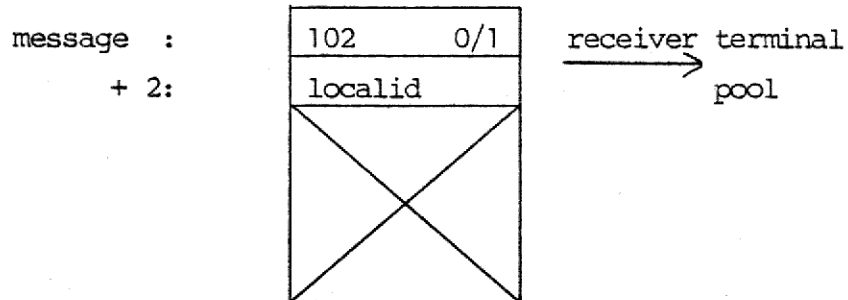
The operation removes a terminal link from a pool. The removal
may be performed soft or hard, i.e. activities in progress may be
terminated or suspended before the removal. In the first case
message (1) must equal 0, in the second 1. The answer on a soft
removal is given when the operation is initiated and tells thus
nothing about the termination of the last activity on the link.

Possible status bits: 15, 16.

2.2.1.6   Lookup Link   2.2.1.6

```
message  :      | 104          0 |   receiver terminalpool
       + 2:     | localid        |
                |       X        |
```
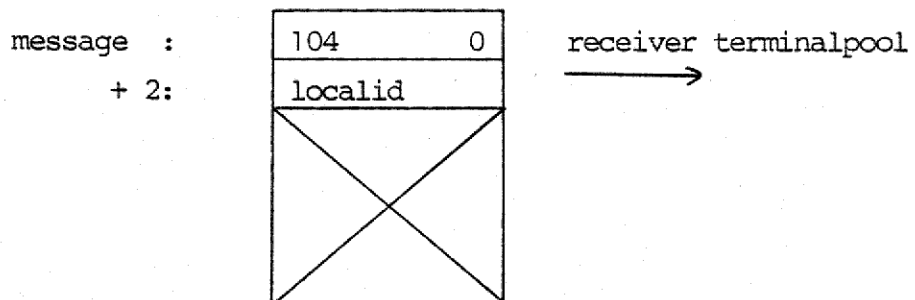
If the link is known the answer will be:

```
answer   :      | status = 0          |
       + 2:     | localid             |
       + 4:     | term.proc.descr.adr.|
       + 6:     | bufs    | timers    |
       + 8:     | pool                |
      +10:      | blocks full         |
      +12:      | halfwords free      |
```
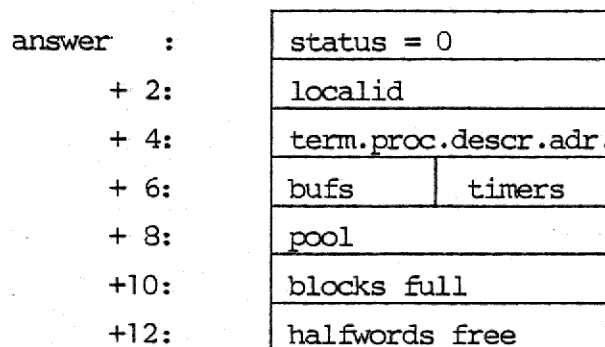
Message (2:7) contains the same information as the corresponding fields in the create link message.

Message (8:9) is the process description address of the pseudo process corresponding to the pool.

Message (10:11) contains the number of operations in queue for the link. I.e. the number of input and output operations sent to the link, but not yet executed.

Message (12:13) tells whether the pool queue to the link is full or not. If the value is 0 further input/output operations will be delayed until some of the activities in progress to the device has been completed.

Possible status bits: 15, 16.

2.2.1.7   Lookup Terminal   2.2.1.7

```
message  :   ┌─────────────────────┐   receiver TEM
             │ 106           0     │   ──────────→
      + 2:   │      ╳              │
             ├─────────────────────┤
      + 4:   │ term.proc.descr.adr.│
             │      ╳              │
             │      ╳              │
             └─────────────────────┘
```
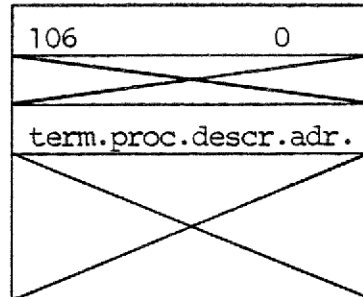
Answer as for lookup link.

Possible status bits: 16, 19, 21.

If bit 19=1, the terminal is in a link, but the corresponding pool is created by another user.

2.2.2   Input/Output Operations   2.2.2

The input/output operations of TEM are similar to the operations known from the external processes in the monitor ref. [1], ref. [5], ref. [6].

The following functions have been implemented:

1) Sense.
2) Sense ready.
3) Format 8000 device control.
4) Input.
5) Output.

This means that an application may use the basic I/O-system of the file processor or the high level languages.

In addition a few operations to control the multiplexing and spooling in TEM is introduced:

1) Simulate input.
2) Start input.

The formats for answers are as defined in ref. [1] for external processes.

TEM generates dummy results (W0 $\diamond$ 1) in the following situations:

2: Application not user of the pool.
3: Message unintelligible.
4: Link not known in pool.

2.2.2.1   Sense   2.2.2.1

message :
    + 2:
    + 4:

0	0
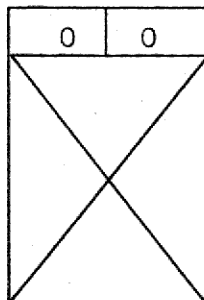
receiver terminal pool
$\longrightarrow$

```
answer    :     |        0        |
   + 2:         |        0        |
   + 4:         |        0        |
                |_____X_____|
```

2.2.2.2   Sense Ready

```
message   :     |   0   |    2    |  →  receiver terminal pool
                |_____X_____|


answer    :     |     status      |
   + 2:         |        0        |
   + 4:         |        0        |
                |_____X_____|
```

First input operations are initiated as described in 2.2.1.4.
Then an answer from an external process is waited for. If data is
ready the sense ready operation is answered with status = 0. Else
the operation is returned with the answer delivered by the exter-
nal process.

2.2.2.3   Format 8000 Device Control

```
message   :     |   2   | operation  |
   + 2:         |    modifier        |
   + 4:         |  physical address  |
   + 6:         |  logical address   |
   + 8:         |   local-id *)      |
```

```
answer    :    | status           |
      + 2:     | result           |
      + 4:     | physical address |
      + 6:     | logical address  |
```

*) The two byte device address used in communication with TEM,
   i.e. LOCAL-ID:

```
           | 0 | CU | DEVICE |
```
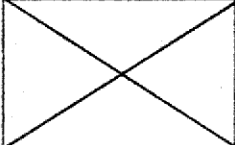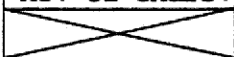
The message is forwarded to the termout process indicated by
local-id and the answer is returned unchanged by TEM, ref. [6].

The message must be sent on an outputlink.

2.2.2.4   Input   2.2.2.4

```
message   :    | 3        | mode |    receiver terminal pool
      + 2:     | first adr.      |
      + 4:     | last adr.       |
                |                |
         .      |                |
         .      |      ><        |
         .      |                |
      +14:     |                |
```

```
answer    :    | status          |
      + 2:     | halfwords       |
      + 4:     | no. of chars.   |
      +14:     |      ><         |
```

First input operations are initiated as described in 2.2.1.4. Se-
condly an answer from some external process in the pool is waited
for. The answer including the indata delivered is returned to the
user, perhaps modified with address information.

Because input may be initiated asynchronous with the input mes-
sage from the user, some comments may be necessary.

The mode field in the first input operation defines the input mode in the whole lifetime of the pool. Correspondingly, the indata buffer that the application makes av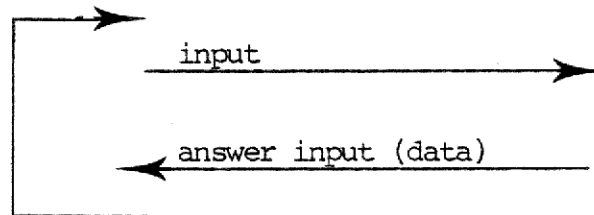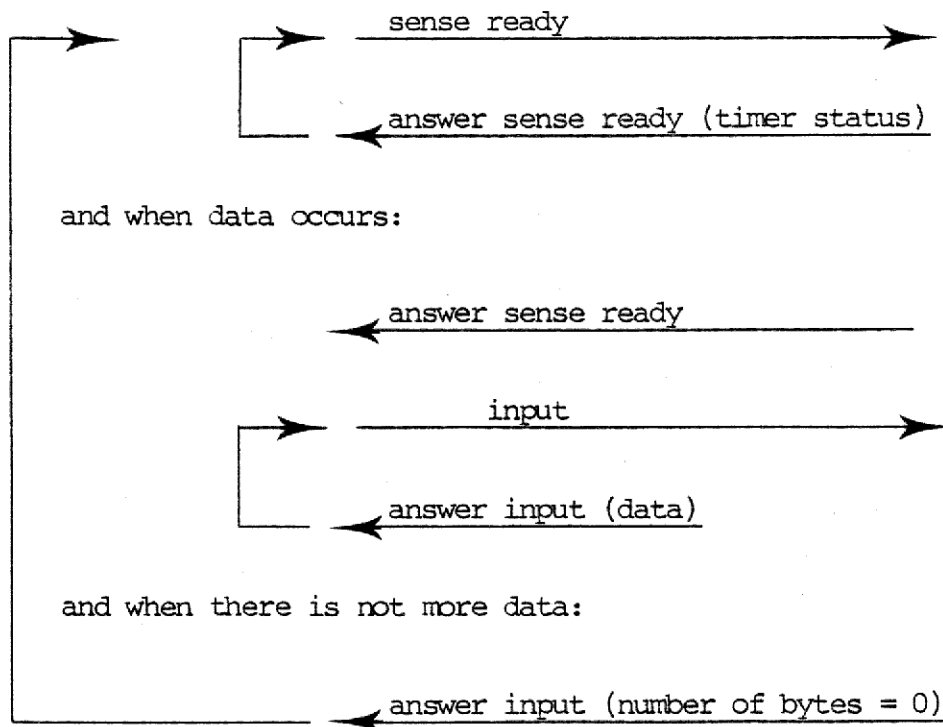ailable, may be too small for the block received from an external process. Normally TEM delivers data with the original blocking, but in this case indata is divided into smaller portions.

The first time the indata stream runs empty after a sense ready operation has been called, the input operation is answered immediately with zero answer. This means that the user may chose freely between two different input protocols, namely the traditional protocol and the sense ready protocol.

Traditional protocol:

```
  ┌──────►
  │         input ──────────────────►
  │
  │        ◄─ answer input (data)
  └──────
```

Sense ready protocol:

```
┌───►        ┌──►  sense ready ──────────────────►
│            │
│            │     ◄─ answer sense ready (timer status)
│            └──
│
│  and when data occurs:
│
│                  ◄─ answer sense ready
│
│            ┌──►        input ──────────────────►
│            │
│            │     ◄─ answer input (data)
│            └──
│
│  and when there is not more data:
│
│                  ◄─ answer input (number of bytes = 0)
└────────────
```

```
message   :      | 5  | mode    |        receiver terminal pool
    + 2:         | first adr.   |        ──────────>
    + 4:         | last adr.    |
         :       |      ╳       |
    +14:         |      ╳       |


answer    :      | status = 0   |
    + 2:         | halfwords    |
    + 4:         | no. of chars.|
                 |      ╳       |
                 |      ╳       |
```
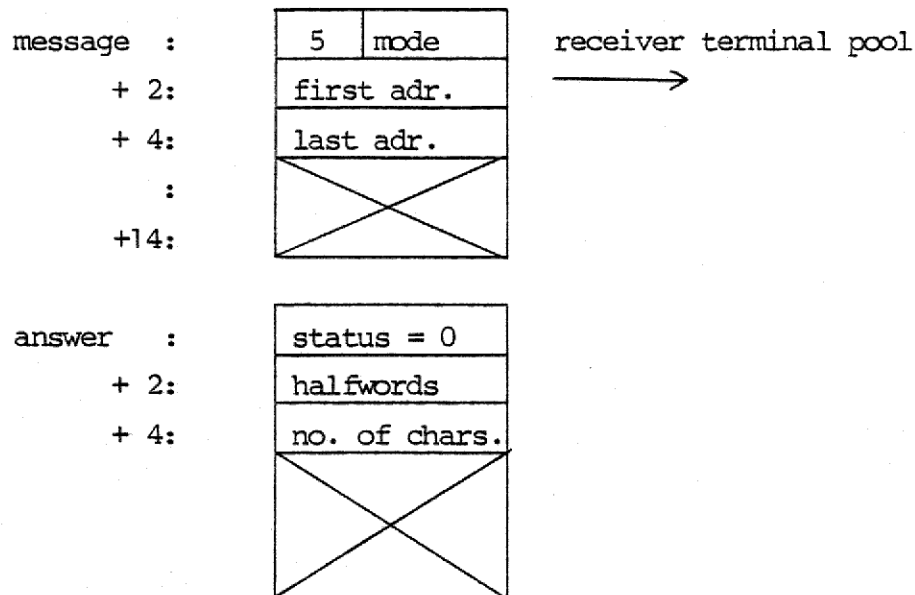
If the link exists, the result of the operation will be normal
and status = 0. As the operation is taken over by TEM asynchron-
ous with the working of the device, hard errors are not reported
to the application.

In type 0 and type 2 links the mode field of the output operation
is used to signal whether the next input operation should be in
non- display mode (e.g. entering of password information):

    if mode shift (-3) extract 1 = 1 then
    begin
      mode := mode-8;
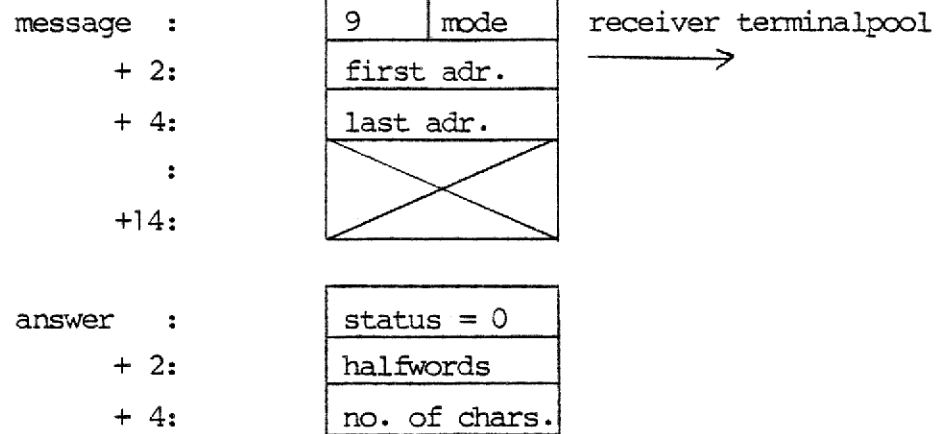   non_display_mode := true;
    end
    else non_display mode := false;

The spooling facilities of TEM means that the pool will act as a
(very) fast device. Only if the spool queue is full, the answer
will be delayed by the working of the device.

The largest blocksize accepted by TEM is 450 halfwords. This
number is independent of the trimming of TEM.
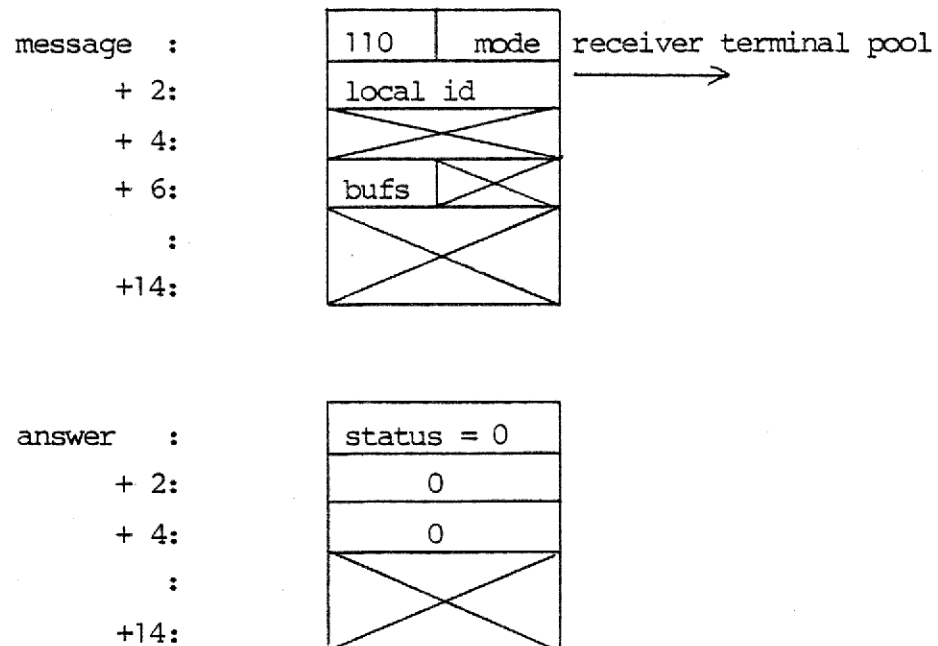
2.2.2.6   Simulate Input   2.2.2.6

```
message    :   | 9        | mode   |   receiver terminalpool
      + 2:     | first adr.        |   ------->
      + 4:     | last adr.         |
         :     |  \      /          |
     +14:      |   \    /           |

answer     :   | status = 0        |
      + 2:     | halfwords         |
      + 4:     | no. of chars.     |
```

This operation is defined for links of type 0 only. The link ad-
dressed in the data area referenced by message (2:3) and message
(4:5) is removed and created again. The data area specified will
be handled by TEM in the same way as indata from the device. I.e.
it will be delivered as data to the user process in a later input
operation.

2.2.2.7   Start Input   2.2.2.7

```
message    :   | 110      | mode   |   receiver terminal pool
      + 2:     | local id          |   ------->
      + 4:     |  \      /          |
      + 6:     | bufs  |X|          |
         :     |   \    /           |
     +14:      |  /      \          |

answer     :   | status = 0        |
      + 2:     |        0          |
      + 4:     |        0          |
         :     |  \      /          |
     +14:      |   \    /           |
```

Independent of the normal input spooling 'bufs' input operations
on the link are initiated.

The automatic activation of input operations as defined in the creation of the link is not resumed before the number of outstanding transactions becomes less than standard.

3.   OPERATING GUIDE   3.

The operator's tasks in the day to day running of TEM comprise
the following:

1. The start up.
2. Closing down.
3. How to handle error situations.

In the following the operating system 's' is implied. If another
operating system is used, commands and messages may be different.

3.1   The Start-Up   3.1

During start-up the system calculates the optimal set of re-
sources. If TEM is started with more or less resources than ne-
cessary, the optimal value is displayed as a parent message. If
resources are missing the message is marked with '***' and the
run is terminated at once.

The name of the program to be loaded is 'btem'. Formulas for
computing the resource claims are given in subsection 4.4.2.

Example 1:
Output from the computer in this and the following examples are
written with capital letters, while input from the operator is
indicated by small letter.

    ATT s
    new tem size 10000 buf 30 area 4
    READY
    ATT s
    prog btem base -8388607 8388605 run
    READY
    MESSAGE TEM VERSION: 830401 0
    MESSAGE TEM SIZE   9320
    MESSAGE TEM *** AREA   6
    MESSAGE TEM BUF   20
    PAUSE TEM *** INIT TROUBLES

```
ATT s
remove area 6 buf 20 run
READY
MESSAGE TEM VERSION: 830401 0
MESSAGE TEM SIZE      9320
MESSAGE TEM STARTED
```

TEM is started with too few area processes. Therefore the run is
terminated. The optimal values of coresize, area processes and
message buffers are displayed. Then the TEM process is removed
and started again with a reasonably set of resources, but the
coresize is still larger than necessary.

Syntax of a start up message:

$$\left\{ \begin{array}{l} \text{message} \\ \text{pause} \end{array} \right\} \quad \text{tem} \quad \left\{ \begin{array}{c} \text{***} \\ \text{<sp> <sp> <sp>} \end{array} \right\} \quad \text{<message text>}$$

List of start up message texts	
version:<i> <i>	the date of the TEM release and the date of options are displayed.
size <i>	optimal value of coresize.
area <i>	optimal value of area processes.
buf <i>	optimal value of message buffers.
<name of area> <i>	too few resources for creating work areas.
started	tem is running.
init troubles	resources missing, execution terminated.

3.2   Closing the System   3.2

There exists no close command in TEM. Closing down after a normal
run is done by simply removing the process as seen below:

```
ATT s
proc tem remove
READY
```

3.3   How to Handle Error Situations   3.3

During the run the system may break down in one of the following
ways:

1. A program error may cause the system to break down, and the
   following error message will be printed on the terminal
   from where the system was started:

   ```
   PAUSE TEM    *** FAULT
   ```

2. The system dies without printing a message. Then the pro-
   cess ought to be 'breaked' in order to have the last por-
   tion of testoutput generated, written on the testarea:

   ```
   ATT s
   proc tem break
   READY
   ```

3. A hard error in a work area makes continued running impos-
   sible and the system stops after printing the message:
   ```
   PAUSE TEM STATUS <status word>    <area>
   ```

In all error situations one should, if the system has been trim-
med with 'testoutput' move this from the test area TEMTEST to a
work area, from which the TRACE-program can print it for further
analyses.

The TRACE program is automatically generated by the installation of the system. The program is called as follows:

    trace <testarea> . <segments>

<testarea> is the name of the area, from which the testoutput is to be printed (the work area the testoutput has been moved to, or the test area itself).

<segments> are the maximum number of segments to be analyzed. TRACE always finds the latest generated segments, and counts the number of segments backwards from there. <segments> are automatically cut to the size of the area, if something larger has been specified.

Example 2:

An s-run; testoutput is printed before a restart.

    ATT s
    proc tem remove new tem run
    READY
    o lp
    trace temtest.10000 (everything is printed)
    o c
    ATT s
    proc tem remove
        -       (a new start-up)

Example 3:

A BOSS-run; the testoutput has been moved to the area TESTCOPY.

    10 o pip
    20 trace testcopy.10000 (everything is printed)
    30 o c
    40 convert pip
    50 finis
    go

4.   SYSTEM GENERATION   4.

4.1   Installation   4.1

TEM may be installed on the RC4000 and the RC8000 series compu-
ters. Before installation check that the version of

    a) your monitor is $\geq$ 5.0
    b) your algolcompiler is ALGOL7 or newer.

In order to ensure a high degree of flexibility and a good util-
ization of hardware, the system staff may adapt the system.

Before the system is trimmed one has to consider the following
quantities:

"options"   At start up a constant showing the date of the TEM
                 system will be listed together with this constant.
                 At each trimming the constant should be changed to
                 show the date of the trimming (e.g. 830415). The
                 standard value is 0 indicating that standard op-
                 tions are used.

"thcount"   The maximal number of terminals and format 8000
                 links running under TEM at the same time.

"phcount"   The maximal number of terminal pools.

"phspoolsegm"   Number of spoolsegments per terminal pool. I.e.
                 the maximal size of the queue used to spool input
                 to the pool (used by answers to input operations
                 and input data).

"thspoolsegm"   Number of spoolsegment per terminal link. I.e. the
                 maximal size of the queue used to spool operations
                 to a terminal (used by input/output operations and
                 output data).

"thbufsize"   Number of spoolbuffers in core (segment buffers).
If this trim parameter is made larger, the number
of transports to and from backing storage in con-
nection with spooling of data between the appli-
cations and the terminals will decrease. The ex-
pense will be 512 halfwords in primary store pr.
buffer.

"testsegments" The number of testoutput segments. If this number
is zero no testoutput is generated. Performance is
higher if testoutput is suspended, but the possi-
bilities for discovering system errors will be
minimal. If TEM is running together with systems
also producing testoutput (e.g. SOS) the need for
testoutput will be less and the generation of
testoutput may be stopped.

Please observe that:

a) input/output operations and output data are accomodated in
'thspoolsegn',

b) answers to input, sense and sense ready operations and in-
put data are accomodated in 'phspoosegm'.

In installation servicing format 8000 terminals one should con-
sider that:

a) the time any CU will wait to deliver an input block is
approx. 15 seconds,

b) a screen image may consist of more blocks (1 block =
approx. 258 bytes, 1 image = 2000 bytes or more),

c) input blocks from the CU may be delayed if the IBM 3270
terminal handler is short of free input buffers,

d) input buffers in the IBM 3270 terminal handler are released
when honouring input operations from e.g. TEM.

As a consequence it is recommended to:

a) accomodate enough space in the spool queue to honour input requests from all terminals serviced by a pool at any time (phspoolsegm = 3*(no of terminals); approx.),

b) prepare the application to utilize the spooling capacities of TEM by assigning an appropriate value to the parameter 'bufs' in the call of the 'create link' message to TEM (bufs = (no of terminals serviced by this link)*8 approx).

The system trimming is done by means of the file temtrim (see appendix C), which contains a set of standard variables plus comments for generating the trimmed version of the program.

Installation may be done after the files have been 'loaded to disc' or direct from tape.

a) Installation from tape.

If the system tape is called mtsw8100, the installation is performed as shown below:

```
temdoc    =    set 1 <discname> ; default = disc
mipshelp =    set mto mtsw8100 0 2
i        mipshelp
i        temhelp
xtrim = edit temtrim

        EDIT COMMANDS
i xtrim
```

b) Installation from diskette.

If the system diskettes are called S18100 and S28100 the

installation is performed as shown below:

"fdload S18100.1"
fdload itself will ask for mounting of the continuation
volume.

When the files have been loaded, use the fp command:

'i mipshelp'

You may now proceed with paragraph c), installation from
backing storage.

c) Installation from backing storage.

```
temdoc  =     set 1 <discname> ; default = disc
i       temhelp
xtrim = edit temtrim
```

EDIT COMMANDS
```
i xtrim
```

Example 4:
The installation is done from the tape mtsw8100, and the trim
parameters 'thcount' and 'testsegments' are changed to 10 and
0 respectively.

```
temdoc  = set 1 disc 2
mipshelp = set mto mtsw8100 0 2
i       mipshelp
i       temhelp
```

```
xtrim = edit temtrim
```

```
1./thcount/, r/15/10/,          (10 active terminals)
1./testsegments/, r/42/0/,      (suspend testoutput)
f
i       xtrim
```

4.2   Resource Demands

4.2.1   When Installing

The process used for installation may run with standard resources except that:

a) Coresize must be >= 50000 halfwords, 60000 reasonable.

b) User scope must equal system scope (-8388607: 8388605). If this is not the case, the scope of the files btem and trace must be changed by hand to system scope after the installation.

c) At the first installation, permanent backing storage resources must be available for the above mentioned files.

4.2.2   When Running

In the go through below the resource demands of TEM when running are listed. As it may be seen from the formulas the demands vary much depending on the TEM trimming.

Primary store (halfwords):
Standard consumption approx:	6800
Terminal pool descriptions:	phcount * 74
Terminal link descriptions	thcount (108 + termbufsize)
Spoolbuffer	spoolbus * 516
Test buffer (optional)	1024

Message buffers:
Constant consumption	2
pool consumption	phcount x 2
Link consumption	thcount

Area processes:
Constant consumption	2
Pool consumption	phcount

Backing storage segments:

Testarea	testsegments
Spool area	phcount * (phspool segm +1) +
	thcount * (thspool segm +1)

Example 5:

If the trim parameters of TEM are set 10

options:=	830401,
thcount:=	10,
phcount:=	4,
termbufsize:=	104,
phspoolsegm:=	8,
thspoolsegm:=	8,
spoolbus:=	2,
testsegments:=	42,

the resource demands will be

Primary store:	10212
Message buffers:	20
Area processes:	6
Backing storage segments:	168

Example 6:

```
INTEGER PROCEDURE CREATEPOOL(Z);
ZONE Z;
BEGIN
  INTEGER I;
  INTEGER ARRAY ZIA(1:20),SIA(1:12);
  ZONE ZTEM(1,1,STDERROR);
  OPEN(ZTEM,0,<:TEM:>,0);
  GETZONE6(Z,ZIA);
  GETSHARE6(ZTEM,SIA,1);
  SIA(4):=90 SHIFT 12;
  FOR I:=0 STEP 1 UNTIL 3 DO SIA(8+I):=ZIA(2+I);
  SETSHARE6(ZTEM,SIA,1);
  MONITOR(16,ZTEM,1,SIA);
  CREATEPOOL:=IF MONITOR(18,ZTEM,1,SIA) <> 1 THEN -1 ELSE SIA(1);
  CLOSE(ZTEM,TRUE);
END CREATEPOOL;


INTEGER PROCEDURE REMOVEPOOL(Z);
ZONE Z;
BEGIN
  INTEGER I;
  INTEGER ARRAY ZIA(1:20),SIA(1:12);
  ZONE ZTEM(1,1,STDERROR);
  OPEN(ZTEM,0,<:TEM:>,0);
  GETZONE6(Z,ZIA);
  GETSHARE6(ZTEM,SIA,1);
  SIA(4):=92 SHIFT 12;
  FOR I:=0 STEP 1 UNTIL 3 DO SIA(8+I):=ZIA(2+I);
  SETSHARE6(ZTEM,SIA,1);
  MONITOR(16,ZTEM,1,SIA);
  REMOVEPOOL:=IF MONITOR(18,ZTEM,1,SIA) <> 1 THEN -1 ELSE SIA(1);
  CLOSE(ZTEM,TRUE);
END REMOVEPOOL;


INTEGER PROCEDURE CREATELINK(Z,TYPE,ID,PROCREF,BUFS,TIMERS,
                            MASK,SUBST);
ZONE Z;
INTEGER TYPE,ID,PROCREF,BUFS,TIMERS,MASK,SUBST;
BEGIN
  INTEGER I;
  INTEGER ARRAY ZIA(1:20),SIA(1:12);
  LONG ARRAY ARR(1:2);
  ZONE ZTEM(1,1,STDERROR);
  GETZONE(Z,ZIA);
  ARR(1):=ZIA(2); ARR(1):=ARR(1) SHIFT 24 ADD ZIA(3);
  ARR(2):=ZIA(4); ARR(2):=ARR(2) SHIFT 24 ADD ZIA(5);
  I:=1;
  OPEN(ZTEM,0,STRING ARR(INCREASE(I)),0);
  GETSHARE6(ZTEM,SIA,1);
  SIA(4):=100 SHIFT 12 ADD TYPE;
  SIA(5):=ID;
  SIA(6):=PROCREF;
  SIA(7):=BUFS SHIFT 12 ADD TIMERS;
  SIA(8):= MASK SHIFT 12 ADD SUBST;
  SETSHARE6(ZTEM,SIA,1);
  MONITOR(16,ZTEM,1,SIA);
  CREATELINK:=IF MONITOR(18,ZTEM,1,SIA) <> 1 THEN -1 ELSE SIA(1);
  CLOSE(ZTEM,TRUE);
END CREATELINK;


INTEGER PROCEDURE REMOVELINK(Z,ID,IMMEDIATE);
ZONE Z;
INTEGER ID;
BOOLEAN IMMEDIATE;
BEGIN
  INTEGER I;
  INTEGER ARRAY ZIA(1:20),SIA(1:12);
  LONG ARRAY ARR(1:2);
  ZONE ZTEM(1,1,STDERROR);
  GETZONE6(Z,ZIA);
  ARR(1):=ZIA(2); ARR(1):=ARR(1) SHIFT 24 ADD ZIA(3);
  ARR(2):=ZIA(4); ARR(2):=ARR(2) SHIFT 24 ADD ZIA(5);
  I:=1;
  OPEN(ZTEM,0,STRING ARR(INCREASE(I)),0);
  GETSHARE6(ZTEM,SIA,1);
  SIA(4):=102 SHIFT 12 +(IF IMMEDIATE THEN 1 ELSE 0);
  SIA(5):=ID;
  SETSHARE6(ZTEM,SIA,1);
  MONITOR(16,ZTEM,1,SIA);
  REMOVELINK:=IF MONITOR(18,ZTEM,1,SIA) <> 1 THEN -1 ELSE SIA(1);
END REMOVELINK;


INTEGER PROCEDURE TERMINALID(TERMINALNUMBER);
INTEGER TERMINALNUMBER;
TERMINALID:=((TERMINALNUMBER//10 + 48) SHIFT 8 ADD
            (TERMINALNUMBER MOD 10) + 48) SHIFT 8 ADD 32;
```

Example 7:

```
;                *** TTEMTEST ***
;
;
; A TESTPROGRAM FOR SIMPEL TESTING OF THE TEM SYSTEM
;
; PROGRAM CALL:
;     TEMTEST TERM.<TERMINALNAME=1>.<TERMINALNAME=2>. ...  <TERMINALNAME=N>
;
; THE PROGRAM ACTS LIKE THIS:
;
;     CREATE TERMINAL POOL
;     CREATE LINKS TO ALL TERMINALS SPECIFIED IN PROGRAM CALL
; LOOP
;     READ AN INPUT LINE FROM A CONNECTED TERMINAL
;         (THIS INPUT LINE STARTS WITH A TERMINAL NUMBER)
;     INCREASE LINECOUNT(TERMINAL NUMBER)
;     WRITE TERMINAL IDENTIFICATION
;     WRITE TERMINAL NUMBER
;     WRITE LINE COUNT
;     WRITE CONTENT OF INPUT LINE
;     GOTO LOOP


BEGIN
  ZONE Z(26,1,STDERROR);
  INTEGER I,ACTIVETERMINALS,MAXTERMINALS,CURRTERMINAL,RESULT,TERMINALREF;
  REAL ARRAY ARR(1:2);

  ALGOL COPY.1; <* COPY TEM PROCEDURES *>


  <*   CREATE TERMINAL POOL   *>

  OPEN(Z,8,<:TEM:>,0);
  CREATEPOOL(Z);
  MAXTERMINALS:=ACTIVETERMINALS:=0;


  <*   CONNECT ALL TERMINALS SPECIFIED IN PROGRAM CALL   *>

  BEGIN
    INTEGER J;
    INTEGER ARRAY IA(1:10);
    ZONE DUMMY(1,1,STDERROR);
    I:=2;
    FOR I:=I WHILE SYSTEM(4,I,ARR) = H SHIFT 12 + 10 DO
    BEGIN
      MAXTERMINALS:=MAXTERMINALS+1;
      J:=1;
      OPEN(DUMMY,0,STRING ARR(INCREASE(J)),0);
      TERMINALREF:=MONITOR(4,DUMMY,0,IA);
      RESULT:=CREATELINK(Z,
              0,TERMINALID(MAXTERMINALS),TERMINALREF,1,0,0,0);
      IF RESULT <> 0 THEN
        WRITE(OUT,<:<10>CREATELINK(:>,<<D>,TERMINALREF,<:) = :>,RESULT) ELSE
        ACTIVETERMINALS:=ACTIVETERMINALS+1;
      I:=I+1;
      CLOSE(DUMMY,TRUE);
    END;
  END;
  IF ACTIVETERMINALS < 1 THEN GOTO STOP;

  BEGIN
    INTEGER I,J;
    INTEGER ARRAY LINEBUF(1:100),LINECOUNT(1:MAXTERMINALS);
    FOR I:=1 STEP 1 UNTIL MAXTERMINALS DO LINECOUNT(I):=0;

  <*   READ A LINE AND DISPLAY IT ON CORRESPONDING TERMINAL   *>

LOOP:
    READ(Z,CURRTERMINAL);
    I:=1;
    FOR I:=I WHILE READCHAR(Z,LINEBUF(I)) <> 8 DO I:=I+1;
    SETPOSITION(Z,0,0);
    LINECOUNT(CURRTERMINAL):=LINECOUNT(CURRTERMINAL)+1;
    WRITE(Z,<<ZD>,CURRTERMINAL,<: TERM = :>,<<ZD>,CURRTERMINAL,
            <: LINE = :>,<<DDD>,LINECOUNT(CURRTERMINAL),<:: :>);
    FOR J:=1 STEP 1 UNTIL I DO OUTCHAR(Z,LINEBUF(J));
    IF LINEBUF(1) = 42 THEN
    BEGIN  <*   A STAR  IN FIRST POSITION MEANS LOGOUT   *>
      WRITE(Z,<:TERMINAL LOGGED OUT<10>:>);
      SETPOSITION(Z,0,0);
      REMOVELINK(Z,TERMINALID(CURRTERMINAL),FALSE);
      ACTIVETERMINALS:=ACTIVETERMINALS-1;
    END;

    SETPOSITION(Z,0,0);
    IF ACTIVETERMINALS > 0 THEN GOTO LOOP;
  END;

STOP:
  REMOVEPOOL(Z);

END
```

Example 8:

```
;               ***   TEM SENSE READY TEST   ***
;
;
; A TESTPROGRAM FOR SIMPEL TESTING OF THE TEM SYSTEM
;
; PROGRAM CALL:
;     <PROGRAMNAME>
;
; THE PROGRAM ACTS LIKE THIS:
;
;     CREATE TERMINAL POOL
; LOOP
;     WAIT ATTENTION OR INPUT READY
;     IF ATT THEN LOGIN GOTO LOOP
;     READ LINE FROM TERMINAL
;     WRITE TERMINAL NUMBER AND LINE NUMBER
;     ECHO INDATA
;     IF FIRST CHAR = * THEN LOGOUT
;     GOTO LOOP


BEGIN
  INTEGER MAXTERMINALS;

  ALGOL COPY.1; <* COPY TEM PROCEDURES *>



  MAXTERMINALS:= 10;

  BEGIN
    BOOLEAN ARRAY PASSIVETERM(1:MAXTERMINALS);
    INTEGER ARRAY LINEBUF(1:100),LINECOUNT(1:MAXTERMINALS);
    ZONE ZIN(26,1,ENDOFDATA),ZOUT(26,1,STDERROR),
          SENSEREADY, ZHELP(1,1,STDERROR);
    INTEGER I,J,ACTIVETERMINALS,CURRTERMINAL,RESULT,
            TERMINALREF,BUFFERBASE;
    BOOLEAN POOLSENSED;
    INTEGER ARRAY IA(1:20);



    PROCEDURE ENDOFDATA(Z,S,B);
    ZONE Z;
    INTEGER S, B;
    BEGIN
      IF B=0 AND S=2 THEN
      GOTO CENTRALWAIT;
    END;

    <* CREATE TERMINAL POOL *>


    OPEN(ZIN,K,<:TEM:>,2);
    OPEN(ZOUT,8,<:TEM:>,0);
    CREATEPOOL(ZOUT);
    OPEN(ZHELP,0,<::>,0);
    OPEN(SENSEREADY,0,<:TEM:>,0);
    GETSHARE6(SENSEREADY,IA,1);
    IA(4):= 0 SHIFT 12 + 2; <* PREPARE SENSE READY OPERATION *>
    SETSHARE6(SENSEREADY,IA,1);
    ACTIVETERMINALS:= 0;
    BUFFERBASE:= 0;
    POOLSENSED:= FALSE;
    FOR I:= 1 STEP 1 UNTIL MAXTERMINALS DO PASSIVETERM(I):= TRUE;
```

(continued)

```
CENTRALWAIT:
    IF ACTIVETERMINALS>0 AND -,POOLSENSED THEN
    BEGIN
        MONITOR(16) SENDMESSAGE:(SENSEREADY,1,IA);
        POOLSENSED:= TRUE;
    END;
    I:= BUFFERBASE;

    RESULT:= MONITOR(24)WAITEVENT:(ZHELP,I,IA);

    IF RESULT=0 THEN
    BEGIN <* (ATTENTION) MESSAGE ARRIVED *>
        IF IA(1)<>0 THEN
        BEGIN
            BUFFERBASE:= I;
            GOTO CENTRALWAIT;
        END;
        MONITOR(26)GET EVENT:(ZHELP,I,IA);
        IA(9):= 1;
        MONITOR(22) SEND ANSWER:(ZHELP,I,IA);
        TERMINALREF:= MONITOR(4) GET DESCRIPTION:(ZHELP,0,IA);
        FOR I:= MAXTERMINALS STEP -1 UNTIL 1 DO
        IF PASSIVETERM(I) THEN CURRTERMINAL:= I; <* FIND FREE TERMINAL NO *>
        RESULT:=CREATELINK(ZOUT,0,TERMINALID(CURRTERMINAL),TERMINALREF,
                            1,2047,0,0);
        IF RESULT<>0 THEN
        BEGIN
            WRITE(OUT,<:<10>CREATELINK(:>,<<DD>,TERMINALREF,<:) = :>,
                  RESULT,<:<10>:>);
            SETPOSITION(OUT,0,0);
        END
        ELSE
        BEGIN
            WRITE(ZOUT,<<ZO>,CURRTERMINAL,FALSE ADD 32,1,
                  <:TERMINAL LOGGED IN<10>:>);
            SETPOSITION(ZOUT,0,0);
            ACTIVETERMINALS:= ACTIVETERMINALS+1;
            PASSIVETERM(CURRTERMINAL):= FALSE;
            LINECOUNT(CURRTERMINAL):= 0;
        END;
        GOTO CENTRALWAIT;
    END
    ELSE
    BEGIN <* ANSWER ( SENSE READY ) *>
        MONITOR(18)WAIT ANSWER:(SENSEREADY,1,IA);
        POOLSENSED:= FALSE;

        REPEAT
            READ(ZIN,CURRTERMINAL); <* END OF DATA HANDLED BY BLOCKPROCEDURE *>
            I:= 1;
            FOR I:= I WHILE READCHAR(ZIN,LINEBUF(I)) <>8 DO I:= I+1;
            SETPOSITION(ZIN,0,0);
            LINECOUNT(CURRTERMINAL):= LINECOUNT(CURRTERMINAL)+1;
            WRITE(ZOUT,<<ZO>,CURRTERMINAL,FALSE ADD 32,1,
                  <: TERM = :>,CURRTERMINAL,
                  <: LINE = :>,<<DDD>,LINECOUNT(CURRTERMINAL),<:: :>);
            FOR J:= 1 STEP 1 UNTIL I DO OUTCHAR(ZOUT,LINEBUF(J));
            IF LINEBUF(1) = 42 THEN
            BEGIN <* A STAR IN FIRST POSITION MEANS LOGOUT *>
                WRITE(ZOUT,<:TERMINAL LOGGED OUT<10>:>);
                SETPOSITION(ZOUT,0,0);
                REMOVELINK(ZOUT,TERMINALID(CURRTERMINAL),FALSE);
                ACTIVETERMINALS:= ACTIVETERMINALS-1;
                PASSIVETERM(CURRTERMINAL):= TRUE;
            END
            ELSE SETPOSITION(ZOUT,0,0);
        UNTIL ACTIVETERMINALS=0;
    END
    REMOVEPOOL(ZOUT);
    CLOSE(ZIN,TRUE); CLOSE(ZOUT,TRUE);
  END;
END
```

Example 9:

```
*********** TEM TEST CREATE POOL AND CREATE LINK *************


PROGRAM CALL:
   <PROGRAMNAME> <POOLNAME>(,<TYPE>,<LOCID>,<PROCESS NAME>,<BUFS>,
                            <TIMERS>,<MASK>,<SUBST>) 0->N


   <POOLNAME>,<LOCID>,<PROCESS NAME>::= <TEXT>
   <TYPE>,<BUFS>,<TIMERS>,<MASK>,<SUBST>::= <INTEGER>


THE PROGRAM CREATES A TERMINAL WITH THE NAME <POOLNAME>. FOR EVERY
SET OF LINK PARAMETERS A TERMINAL LINK IS CREATED
BEGIN
  ALGOL COPY.1; <* COPY TEM CONTROL PROCEDURES *>

  INTEGER I, J, RESULT,
          TYPE, LOCID, TERMINALREF, BUFS,TIMERS, MASK, SUBST;
  INTEGER ARRAY IA(1:20);
  REAL    ARRAY ARR(1:2);
  ZONE Z, DUMMY(1,1,STDERROR);

  IF SYSTEM(4,I,ARR)<>4 SHIFT 12+10 THEN SYSTEM(9,1,<:PARAM:>);
  I:= 1;
  OPEN(Z,8,STRING(ARR(INCREASE(I))),0);
  RESULT:= CREATEPOOL(Z);
  IF RESULT<>0 THEN SYSTEM(9,RESULT,<:CRPOOL:>);

  OPEN(DUMMY,0,<::>,0);
  I:= 0;
  REPEAT <* GET DUMMY MESSAGE FROM TEM *>
    RESULT:= MONITOR(24) WAIT EVENT:(DUMMY,I,IA);
    IF RESULT=0 THEN
    BEGIN
      IF IA(1) = -2 SHIFT 12 THEN
      BEGIN
        MONITOR(26) GET EVENT:(DUMMY,I,IA);
        I:= 0;
      END;
    END;
  UNTIL I=0;
  CLOSE(DUMMY,TRUE);

  I:= 1;
  FOR I:= I+1 WHILE SYSTEM(4,I,ARR)=8 SHIFT 12+4 DO
  BEGIN
    TYPE:= ARR(1);
    I:= I+1;
    IF SYSTEM(4,I,ARR)<>8 SHIFT 12+10 THEN SYSTEM(9,I,<:PARAM:>);
    LOCID:= ARR(1) SHIFT (-24) EXTRACT 24;
    I:= I+1;
    IF SYSTEM(4,I,ARR)<>8 SHIFT 12 +10 THEN SYSTEM(9,I,<:PARAM:>);
    J:= 1;
    OPEN(DUMMY,0,STRING(ARR(INCREASE(J))),0);
    TERMINALREF:= MONITOR(4,DUMMY,0,IA);
    CLOSE(DUMMY,TRUE);
    I:= I+1;
    IF SYSTEM(4,I,ARR)<>8 SHIFT 12+4 THEN SYSTEM(9,I,<:PARAM:>);
    BUFS:= ARR(1);
    I:= I+1;
    IF SYSTEM(4,I,ARR)<> 8 SHIFT 12+4 THEN SYSTEM(9,I,<:PARAM:>);
    TIMERS:= ARR(1);
    I:= I+1;
    IF SYSTEM(4,I,ARR)<> 8 SHIFT 12+4 THEN SYSTEM(9,I,<:PARAM:>);
    MASK:= ARR(1);
    I:= I+1;
    IF SYSTEM(4,I,ARR)<>8 SHIFT 12+4 THEN SYSTEM(9,I,<:PARAM:>);
    SUBST:= ARR(1);

    RESULT:= CREATELINK(Z,TYPE,LOCID,TERMINALREF,BUFS,TIMERS,MASK,SUBST);
    IF RESULT<>0 THEN SYSTEM(9,RESULT,<:CRLINK:>);
  END;

  IF SYSTEM(4,I,ARR)<>0 THEN SYSTEM(4,I,<:PARAM:>);
  CLOSE(Z,TRUE)
END

',F
```

C.   TEMTRIM   C.

Example 10:

```
;                  ***  TEMTRIM  ***
;
;
; CONTAINS OPTIONS FOR TRIMMING TEM SYSTEM
; AND COMMANDS FOR AUTOMATIC SYSTEM GENERATION FROM THE TEM SYSTEM TAPE

MESSAGE TEM RELEASE 2.0


TEMDUMMYOUT=SET 1

XTEM = EDIT TTEM              ; EDIT OPTIONS INTO THE PROGRAM TEXT
L./BODY OF INIT/,
L./===TRIMSTART/,
D./===TRIMFINIS/,
I/


! DATE OF OPTIONS                                  ! OPTIONS      :=    0,
! NUMBER OF ACTIVE TERMINALS                       ! THCOUNT      :=   15,
! NUMBER OF ACTIVE TERMINAL GROUPS                 ! PHCOUNT      :=    5,
! NUMBER OF SPOOL SEGMENTS FOR EACH TERMINAL GROUP ! PHSPOOLSEGM  :=   10,
! NUMBER OF SPOOL SEGMENTS FOR EACH TERMINAL       ! THSPOOLSEGM  :=   10,
! SIZE OF TERMINAL BUFFER IN CORE (HALF WORDS)     ! TERMBUFSIZE  :=  104,
! NUMBER OF SPOOL SEGMENT BUFFERS IN CORE          ! SPOOLBUFS    :=    2,
! SIZE OF TESTOUTPUT AREA (SEGMENTS)               ! TESTSEGMNTS  :=   42,

/,
F

O TEMDUMMYOUT
MODE 1.NO
LOOKUP TEMDOC                 ; IF <TEMDOC> NOT PRESENT
IF OK.NO
MODE 1.YES
O C
IF 1.YES
TEMDOC = SET 1               ; THEN CREATE IT PREFERRABLY ON DISC

RCMOL = ALGOL TRCMOL          ;


BTEM = ENTRY 20 TEMDOC
BTEM = RCMOL XTEM             ; TRANSLATE TRIMMED PROGRAM TEXT

TRACE = ENTRY 40 TEMDOC
TRACE = ALGOL TTRACE         ; GENERATE PROGRAM FOR ANALYSING TESTOUTPUT

SCOPE USER BTEM TRACE

O TEMDUMMYOUT

CLEAR TEMP XTEM RCMOL TEMTRIM TRCMOL TTEM TTRACE TTEMTEST TEMLOAD TEMSAVE,
       TEMLIST

O C
CLEAR TEMP TEMDUMMYOUT
```

RETURN LETTER

Title: Terminal Access Module (TEM)
User's Guide/Reference Manual/   RCSL No.: 31-D689
Installation Guide

A/S Regnecentralen af 1979/RC Computer A/S maintains a continual effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback, your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability:

_____

_____

_____

_____

Do you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

How can this manual be improved?

_____

_____

_____

_____

Other comments?

_____

_____

_____

_____

_____

Name:_____ Title: _____

Company: _____

Address: _____

Date:_____

Thank you

§ REGNECENTRALEN
af 1979

Information Department
Lautrupbjerg 1
DK-2750 Ballerup
Denmark