RCSL No: 43-GL11424

Edition: August 1981

Author: Claus Houlberg Hansen

Title:

NCP Data Structures
Reference Manual
Revision 1.02

RCSL 42-i1592

A/S REGNECENTRALEN
af 1979

Keywords:

Real Time Pascal, PAXNET, Centernet, Network Control Probe (NCP).

Abstract:

This manual describes the data structures of all the interfaces to the Network Control Probe (NCP).
The NCP is a central module in all network computers, because most modules (LCP) are communicating with the NCP.
The NCP is developed under the PAXNET and Centernet project.

(44 printed pages)

Printed by A/S Regnecentralen af 1979, Copenhagen

RCSL 42-i1592

CONTENTS PAGE

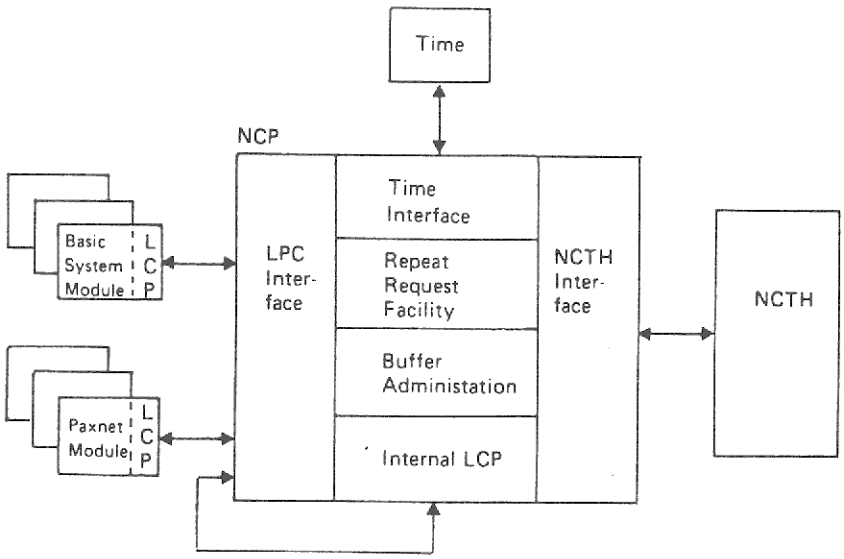
1. INTRODUCTION. 1.

This manual describes the data structures of the Network Control Probe (NCP), i.e.

- the data structures of the Local Control Probe (LCP) interface (user interface).
- the data structures of the Session Control/Network Control Handler Handler (SC/NCTH) interface.
- the data structures of the LCP functions of the NCP.

A detailed description of the NCP module is found in ref. (1) and ref. (4).

The NCP is implemented as one process and the main structure is shown below.



2. LCP INTERFACE. 2.

This chapter describes the LCP interface. The NCP is the passive part towards the LCP, i.e. that the LCP has the initiative, and that the LCP must know the main semaphore of the NCP.

Many of the constants and types used in the following chapter are defined in the environment XNCPENV (appendix B), and it should be used by all LCP modules.

The actual values of the result codes that are used in succeeding chapters can be found in the environment CNNETENV (appendix C).

2.1 Connect LCP. 2.1

User Fields:

	to NCP	from NCP
u1	4+0	4+0
u2	7	result
u3	-	index
u4	-	unch.

Data buffer to NCP:

```
record
  first, last, next: integer;
  lcp_ident: lcp_ident_type;
end;

lcp_ident_type = packed record
        i: bit;
        id: 0..32767;
      end;
```

Data buffer from NCP: Unchanged.

Function:

The specified LCP is connected to the NCP, i.e. that resources in the NCP are reserved for this LCP. An index is returned to the LCP. This index is used in all other communications with the NCP to identify the LCP.

Parameters:

result:

ok	
busy	(no resources)
fct_not_allw	(lcp already connected)

index:
This is the internal NCP identification of the LCP.

first, last, next:
Not used.

lcp_ident:
The LCP identification used in the receiver-id and sender-id fields in the supervisor head format.

i, id:
see section 6.2.3 in (2).

2.2 Disconnect LCP. 2.2

User fields:

	to NCP	from NCP
u1	8+0	8+0
u2	7	result
u3	-	unch
u4	-	unch.

Data buffer to NCP:

```
record
  first, last, next: integer;
  lcp_ident: lcp_ident_type;
end;
```

Data buffer from NCP: Unchanged.

Function:

This message will disconnect the LCP, i.e. that the allocated resources in the NCP are released. A pending 'Wait message' message will be returned.

Parameters:

result:
ok
rec_unkw (lcp not connected)

lcp_ident:
see section 2.1.

2.3 Wait Message

User fields:

	to NCP	from NCP
u1	12+0	12+0
u2	7	result
u3	index	index
u4	-	unch

Data buffer to NCP: Only message header.

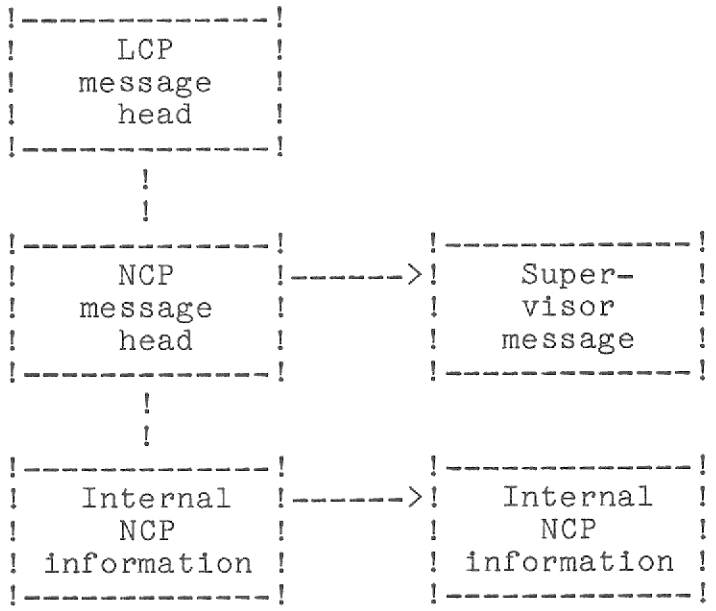
Function:

This message will be queued in the NCP and is only returned, when

- a supervisor message is sent to the LCP
- the LCP has sent a disconnect message
- a module reset statistics is sent from the NCC/NCTH.

When the NCP wants to send a buffer containing a supervisor message, then the 'wait message' head is pushed opon the buffer containing the supervisor message.

The format of the answer is:



Parameters:

result:

ok	
user_term	(if lcp disconnected)
format_err	(illegal index)
ab_term	(reset all statistics)

index:

see section 2.1.

2.3.1 Supervisor Message Buffer

2.3.1

The format of the supervisor message buffer that is returned as the answer to 'wait message', is described below.

User fields:

	to LCP	from LCP
u1	12+3	12+3
u2	7	result
u3	index	index
u4	-	unch

Data buffer to LCP:

```
packed record
  first, last, next: integer;
  sp_head: sp_head_type;
  sp_data: packed array(1..??) of byte;
end;

sp_head_type = packed record
        receiver_id: lcp_ident_type;
        sender_id: lcp_ident_type;
        seq_no: byte;
        sp_type: sp_type_type;
        lcp_oper: lcp_oper_type;
        status: set of sp_status_bit;
        time: packed array(1..12) of time_digit;
        bytecount: integer;
      end;
```

Data buffer from LCP: same format.

Parameters:

result:
only ok is accepted

index:
see section 2.1

A short description of all the above mensioned types is given in appendix B.

A description of all the fields in the supervisor head is given in section 6.2.3 in (2).

The following fields in the supervisor message are relevant for the LCP:
- sp_type (= message, request, stat_cntr)
- lcp_oper
- status (= 0)
- bytecount
- data

The following fields in the supervisor message must be updated by the LCP:
- status (= normally zero)
- bytecount
- data

2.4 Request/Wait Event Buffer 2.4

User fields:

	to NCP	from NCP
u1	16/20+0	16/20+0
u2	7	result
u3	index	index
u4	-	unch.

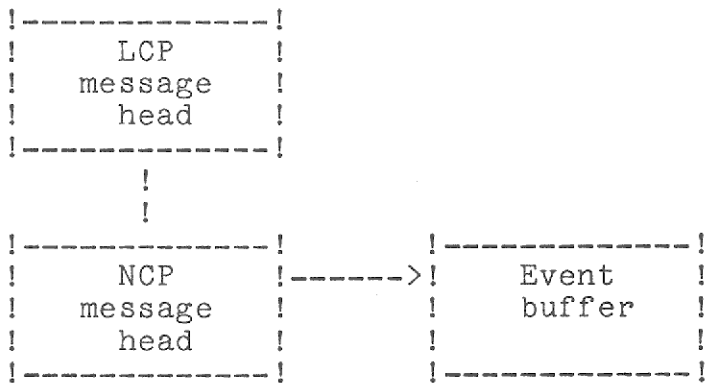
Data buffer to NCP: Only message header.

Function:

if u1 = 16 (request event buffer), then the NCP returns the answer immediately. If there are no buffers, it is indicated in u2 (Result = busy).

If u1 = 20 (wait event buffer), then the NCP does not return the request, until a buffer is avaiblable. When a buffer is avaiblable, then the 'wait event buffer' message is pushed opon the event buffer and returned.

The format of the answer is:



Parameters:

result:

ok	
format_err	(illegal index)
busy	(no resources)

index:

see section 2.1.

2.4.1 Event Buffer.

The format of the event buffer that is returned as the answer on 'request/wait event buffer', is described below.

User fields:

	to LCP	from LCP
u1	16/20+3	16/20+3
u2	7	result
u3	index	index
u4	-	unch.

Data buffer to LCP:

```
packed record
  first, last, next: integer;
  sp_head: sp_head_type;
end;
```

Data buffer from LCP:

```
packed record
  first, last, next: integer;
  sp_head: sp_head_type;
  event_data: packed array(1..??) of byte;
end;
```

parameters:

u1 = 16/20:
see section 2.4

result:
only ok is accepted.

index:
see section 2.1.

sp_head:
supervisor head.

sp_head_type:
see section 2.3.1.

The following fields in the supervisor head are initialized by the NCP:

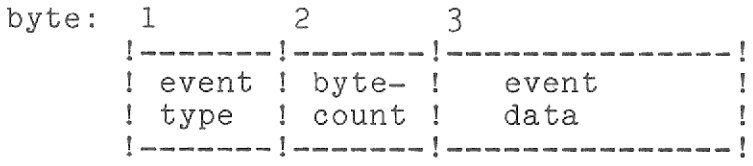
- receiver_id (= LCP_id)
- sp_type (= event, request)
- lcp_oper (= 3)
- seq_no
- status (= 0)
- bytecount (= 0)

The following fields in the supervisor message must be updated by the LCP:

- status (= normally zero)
- bytecount
- data

  The data may contain any number of event records. The first byte in a record identifies the event type, and the second byte specifies the the length of the rest of the record (event data).

  The format of an event record is:



3. SC/NCTH INTERFACE

The input/output messages to the SC/NCTH (in the following SC) are stacked messages. The first message (top message) in the stack contains information to the SC, and the succeeding messages contain the data that should be transferred.

The interface is based on the interface described in ref. (3).

3.1 Top Messages

The format of the top message is described in the following sections.

3.1.1 Request Input

User fields:

	to SC	from SC
u1	8+1	8+1
u2	7	result
u3	-	unch
u4	-	unch

Data buffer to SC:

```
record
  first, last, next: integer;
  local_port: integer;
end;
```

Data buffer from SC:

```
record
  first, last, next: integer;
  local_port: integer;
  send_sc: sc_comm_type;
end;

sc_comm_type = record
        port_no: integer;
        ack_req: byte;
        nuid_signf: byte;
        nuid: packed array(1..nuid_lgt) of byte;
      end;
```

Parameters:

local_port:
SC port number is 1 for the NCP.

send_sc:
SC address of the sender of the supervisor message.

port_no:
SC port number is 2 for the NCC.

ack_req:
see section 3.4.1 in (3).

nuid_signf:
specifies the length of significant bytes of the following field, nuid.

nuid:
the receiving SC address.

nuid_lgt:
maximum length of nuid.

3.1.2 Request Output

3.1.2

User fields:

	to SC	from SC
u1	8+2	8+2
u2	7	result
u3	-	unch
u4	-	unch

Data buffer to SC:

```
record
  first, last, next: integer;
  local_port: integer;
  rec_sc: sc_comm_type;
end;
```

Data buffer from SC: Unchanged

Parameters:

local_port:
see section 3.1.1.

rec_sc:
SC address of the receiver of the supervisor message.

sc_comm_type:
see section 3.1.1

3.2 Succeeding Messages after Top Message 3.2

The data part of the stacked messages to the SC is placed in succeeding messages, and has the following format:

3.2.1 SC Data 3.2.1

User fields:

	to SC	from SC
u1	-	unch
u2	-	unch
u3	-	unch
u4	-	unch

Data buffer to SC:

```
packed record
  first, last, next: integer;
  sp_head: sp_head_type;
  sp_data: data_type;
end;
```

Parameters:

sp_head:
supervisor head

sp_head_type:
see section 2.3.1.

sp_data:

More types of the data_type are possible.

- Normal supervisor message:
  sp_type = (message, request)
  data_type = packed array(1..??) of byte

- Start repeatable function:
  sp_type = (message, request, ncp_control,
  start_repeat)

  data_type = packed record
  period: integer;
  start: packed array(1..8) of (0..15);
  data: packed array(1..??) of byte;
  end;

  When this supervisor message is sent on to the LCP, then the fields 'period' and 'start' are removed.

- Stop repeatable function:
  sp_type = (message, request, ncp_control,
  stop_repeat)

  data_type is empty

period:

see section 4.1.1 of (1).

start:

see section 4.1.1 of (1).

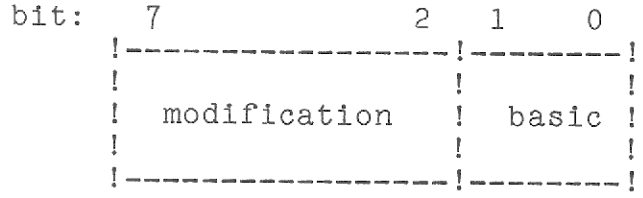
4. THE LOCAL LCP IN THE NCP

The LCP operations are subdivided into these groups:
- control
- sense
- get statistics
- event

The events are handled in a special way, but the other operations are handled in much the same way, as described below.

The control, sense, and get statistics operations can be requested by the NCC (NCTH), and the actual operation is specified in the LCP-oper field in the supervisor head.

The LCP-oper field is subdivided as follows:



Basic is coded as follows:

0 control
1 sense
2 get statistics
3 event

In the following the LCP-oper and the supervisor data format are specified.

4.1 Control Operations

4.1.1 Set Event Mask

LCP-oper: 4 * 1 + 0

```
Supervisor data:
          packed record
            update_mask: set of ev_bit_mask;
            event_mask: set of ev_bit_mask;
          end;

  ev_bit_mask = (prod_stat, ?, ?, ?, ?, ?,
                 lack_of_res, ?, ?, ?, ?,
                 connection,
                 disconnection,
                 collision, ?, ?);
```

Answer:

```
Supervisor data:
          packed record
            act_event_mask: set of ev_bit_mask;
          end;
```

Parameters:

update_mask:
specifies the event bits that has to be updated.

event_mask:
specifies the new value of the event bits that has to be updated.

ev_bit_mask:
bit15:statistics are updated
bit9: lcp connection rejected (lack of resources)
bit4: lcp connected
bit3: lcp disconnected
bit2: lcp connection rejected (collision)

act_event_mask:
the updated event mask

4.1.2 Module Reset Statistics 4.1.2

LCP-oper: 60 * 4 + 0

```
Supervisor data:
          packed record
            lcp_ident: lcp_ident_type;
          end;
```

The supervisor data field may contain any number of these records, but the bytecount must be adjusted according to the number of records.

If the supervisor data field is empty (bytecount = 0), then it is interpreted as a ´node reset statistics´, and all connected LCP´s will receive a ´wait message´ with result = ab_term (reset all statistics).

If the supervisor data field is not empty, then the specified LCP´s will receive a ´wait message´ with result = ab_term.

Answer:

Supervisor data:
Unchanged

Parameters:

lcp_ident:
specifies the LCP that should reset statistics.

lcp_ident_type:
see section 2.1

4.1.3 Set Time

4.1.3

LCP-oper: 61 * 4 + 0

Supervisor data:

```
packed record
  new_time: packed array(1..12) of char;
end;
```

Answer:

Supervisor data:
Unchanged

Parameters:

new_time:
The first two bytes specify the year, the next two bytes specify the month, and so on until the last two bytes that specify the seconds.

4.1.4 Set Event Answer Address 4.1.4

LCP-oper: 62 * 4 + 0

Supervisor data:

```
packed record
  lcp_ident: lcp_ident_type;
  rec_sc: sc_data_type;
  rec_ident: lcp_ident_type;
end;

sc_data_type = packed record
  port_no: integer;
  facility: byte;
  nuid_signf: byte;
  nuid: packed array(1..nuid_lgt) of byte;
end;
```

The supervisor data field may contain any number of these records, but the bytecount must be adjusted according to the number of records.

Answer:

Supervisor data:

Unchanged

Parameters:

lcp_ident:
: specifies the LCP that this record concern

rec_sc:
: specifies the receiving SC address for any events from the denoted LCP

rec_ident:
: specifies the event collector utility ident

lcp_ident_type:
: see section 2.1

port_no:
: see section 3.1.1

facility:
: see section 3.4.1 in (3). Facility is converted to ack_req as described in section 3.1.1

nuid_signf:
see section 3.1.1

nuid:
see section 3.1.1

nuid_lgt:
see section 3.1.1

4.1.5 Set Exception Return Address

LCP-oper: 63 * 4 + 0

Supervisor data:

```
packed record
  rec_sc: sc_data_type;
  rec_ident: lcp_ident_type;
end;
```

Answer:

Supervisor data:
Unchanged

Parameters:

rec_sc:
specifies the exception SC address.

rec_ident:
specifies the exception utility ident.

sc_data_type:
see section 4.1.4

lcp_ident_type:
see section 2.1

4.2 Sense Operations 4.2

4.2.1 Get Event Mask 4.2.1

LCP-oper: 1 * 4 + 1

Supervisor data:
not used

Answer:

Supervisor data:
```
packed record
  act_event_mask: set of ev_bit_mask;
end;
```

Parameters:

see section 4.1.1

4.2.2 Get Event Answer Address 4.2.2

LCP-oper: 62 * 4 + 1

Supervisor data:
```
packed record
  lcp_ident: lcp_ident_type;
end;
```

Any number of these records may occur.

Answer:

Supervisor data:
```
packed record
  lcp_ident:lcp_ident_type;
  rec_sc: sc_data_type;
  rec_ident: lcp_ident_type;
end;
```

Parameters:

see section 4.1.4

4.2.3 Get Exception Return Address 4.2.3

LCP-oper: 63 * 4 + 1

Supervisor data:
not used

Answer:

Supervisor data:
```
packed record
  rec_sc: sc_data_type;
  rec_ident:lcp_ident_type;
end;
```

Parameters:

see section 4.1.5

4.2.4 Get Connected LCP´s 4.2.4

LCP-oper: 2 * 4 + 1

Supervisor data:
not used

Answer:
Supervisor data:
```
packed record
  lcp_ident: lcp_ident_type;
end;
```

Any number of these records may occur.

Parameters

lcp_ident:
specifies a connected LCP.

lcp_ident_type:
see section 2.1

4.2.5 Get Repeatable Functions 4.2.5

LCP-oper: 5 * 4 + 1

Supervisor data:
```
          not used
```

Answer:

Supervisor data:
```
          packed record
            lcp_ident: lcp_ident_type;
            seq_no: byte;
            lcp_oper: lcp_oper_type;
          end;
```

Any number of these records may occur.

Parameters:

The specified parameters are fields in the supervisor head in the supervisor packet that has initiated the repeatable function. See section 2.3.1. and 3.2.1.

4.3 Get Statistics 4.3

LCP-oper: 1 * 4 + 2

Supervisor data:
```
          packed record
            lcp_ident: lcp_ident_type;
          end;
```

Any number of these records may occur.

Answer:

Supervisor data:
```
          packed record
            ncp_stat: ncp_stat_type;
            lcp_statis: array(0..??) of lcp_stat_elem;
          end;

  ncp_stat_type = packed record
            lcp_stat: lcp_stat_type;
            repeat_stat: rep_stat_type;
          end;
```

```
lcp_stat_elem = packed record
                  lcp_ident: lcp_ident_type;
                  lcp_stat: lcp_stat_type;
                  repeat_stat: rep_stat_type;
                end;

lcp_stat_type = record
                  messages: integer;
                  events: integer;
                  pending_msg: integer;
                end;

rep_stat_type = record
                  repeat_opers: integer;
                  lost_repeat: integer;
                end;
```

Parameters:

lcp_ident:
specifies the LCP from where the statistical information is wanted.

lcp_ident_type:
see section 2.1

ncp_stat:
statistical information, concerning the total NCP. These statistics are not reset event though stat_cntr = reset_stat in the type field of the supervisor head.

lcp_statis:
statistical information, concerning the specified LCP. Any number of these records may occur.

messages:
number of supervisor messages.

events:
number of event messages.

pending_msg:
number of supervisor messages that has been pending in the NCP.

repeat_opers:
actual number of repeatable operations, concerning the specified LCP.

lost_repeat:
number of lost repeatable operations.

4.4 Events 4.4

The general format of an event record is:

```
packed record
  event_type: byte;
  bytecount: byte;
  params: packed array(1..??) of byte;
end;
```

The supervisor data may contain any number of event records. The 'event type' identifies the record, and the bytecount specifies the length of the rest of the record (see section 2.4.1).

4.4.1 Lack of Resources 4.4.1

event type: 2 * 4 + 3

Supervisor data:

```
packed record
  event_type: byte;
  bytecount: byte;
  lcp_ident: lcp_ident_type;
end;
```

Parameters:

bytecount:
2

lcp_ident:
specifies the LCP that has not been connected.

lcp_ident_type:
see section 2.1

4.4.2 LCP connected

4.4.2

event type: 5 * 4 + 3

Supervisor data:

```
packed record
  event_type: byte;
  bytecount: byte;
  lcp_ident:lcp_ident_type;
end;
```

Parameters:

bytecount:
2

lcp_ident:
specifies the LCP that has been connected.

lcp_ident_type:
see section 2.1

4.4.3 LCP Disconnected

4.4.3

event type: 6 * 4 + 3

Supervisor data:

```
packed record
  event_type: byte;
  bytecount: byte;
  lcp_ident:lcp_ident_type;
  cause: integer;
end;
```

Parameters:

bytecount:
4

lcp_ident:
specifies the LCP that has been disconnected.

cause:
specifies why the LCP has been disconnected.

1 The LCP has sent a disconnect message.

lcp_ident_type:
see section 2.1

4.4.4 LCP Collision 4.4.4

event type: 11 * 4 + 3

Supervisor data:

```
packed record
  event type: byte;
  bytecount: byte;
  lcp_ident: lcp_ident_type;
end;
```

Parameters:

bytecount:
2

lcp_ident:
specifies the LCP that has caused the collision.

lcp_ident_type:
see section 2.1

4.4.5 Events Lost 4.4.5

event type: 63 * 4 + 3

Supervisor data:

```
packed record
  event_type: byte;
  bytecount: byte;
  lost_events: integer;
end;
```

Parameters:

bytecount:
2

lost_events:
number of event records that are lost.

5. COMPILATION AND CREATION OF THE NCP. 5.

5.1 Compilation 5.1

When the NCP is compiled, the following files are used:

```
SNCP        (NCP source and job description)
XTENV
CNNETENV
XNCPENV
TESTENV
```

5.2 Creation 5.2

The NCP process is defined as follows:

```
PROCESS ncp(
  var sys_vector: system_vector;
  var ncp_sem: ! tap_pointer;
  var sc_sem: ! tap_pointer;
  ncp_ident: ! integer);
```

A. REFERENCES A.

1 PAXNET report no. 5
Description of the Network Control Probe (NCP).
September 1980

2 CENTERNET System Specification
RCSL No: 43-GL-10190.

3 CENTERNET, Session Control
Reference Manual
Per Høgh
23/10-1980

4 PAXNET report no. 1
General Introduction and Specification of PAXNET
May 1980

B. XNCPENV

B.

This appendix consists of the external NCP environment, XNCPENV. It defines the most common used constants and types, and it should be used by all LCP modules.

```
xncpenv;

  (*****************************************************************)
  (*                                                               *)
  (*                  external ncp environment                     *)
  (*                                                               *)
  (* description:                                                  *)
  (* the xncpenv defines the most common used constants and types that  *)
  (* all lcp's use. it should be used by all lcp modules.          *)
  (*                                                               *)
  (*   date    init   changes                                      *)
  (*  ------------------------------------------------------------ *)
  (*  810217   chh    first released version                       *)
  (*  810408   chh    event and supervisor pool sizes inserted    *)
  (*                                                               *)
  (*****************************************************************)


CONST

   (*************************************)
   (****   configuration constants   ****)
   (*************************************)

  time_lgt = 12;              (* digits in time in supervisor message *)
  sp_head_lgt = 17;           (* length of supervisor head (bytes) *)
  event_buf_size = 300;       (* no. of bytes in an event buffer *)
  sup_buf_size = 300;         (* no. of bytes in a supervisor message buffer *)


   (*************************************************)
   (****   u1 values in messages to the ncp   ****)
   (*************************************************)

  connect_lcp = 4;
  disconnect_lcp = 8;
  wait_message = 12;          (* wait for a supervisor message *)
  req_event_buf = 16;         (* get an event buffer, if any free *)
  wait_event_buf = 20;        (* wait forever for an event buffer *)


   (*************************************************)
   (****   u1 values in messages from the ncp   ****)
   (*************************************************)
```

```
    sup_mess_buf = 12 + 3;      (* supervisor message buffer from the ncp *)
    event_buf_req = 16 + 3;     (* event buffer, when no wait *)
    event_buf_wait = 20 + 3;    (* event buffer, when waited forever *)


      (*********************************************)
      (****    u2 values in answers from the ncp   ****)
      (*********************************************)

      (*   see cnnetenv. these are used   *)


  TYPE

      (*********************************)
      (****   supervisor messages   ****)
      (*********************************)

    lcp_ident_type = PACKED RECORD
                             !  i: bit;
                             !  id: 0..32767;
                             END;

    req_ans_type = (req, ans);

    mess_ev_type = (mess, event);

    ncp_cntr_type = (no_ncp_cntr, ncp_cntr);

    rep_func_type = (start_rep, stop_rep);

    rej_func_type = (no_reject, reject);

    stat_cntr_type = (no_reset_stat, reset_stat);

    sp_type_type = PACKED RECORD
                          !  req_ans: req_ans_type;
                          !  mess_ev: mess_ev_type;
                          !  ncp_control: ncp_cntr_type;
                          !  rep_func: rep_func_type;
                          !  rej_func: rej_func_type;
                          !  stat_cntr: stat_cntr_type;
                          !  ?,?: bit;
                          END;

    basic_type = (lcp_cntr,          (* lcp control operation *)
```

```
                    lcp_sense,          (* lcp sense operation *)
                    lcp_get_stat,       (* lcp get statistics operation *)
                    lcp_event);         (* lcp event operation *)

lcp_oper_type = PACKED RECORD
                    !  modif: 0..63;
                    !  basic: basic_type;
                    END;

sp_status_bit = (sp_status_15,      (* module dependent status bit *)
                 sp_status_14,      (* module dependent status bit *)
                 sp_status_13,      (* module dependent status bit *)
                 sp_status_12,      (* module dependent status bit *)
                 sp_status_11,      (* module dependent status bit *)
                 sp_status_10,      (* module dependent status bit *)
                 sp_status_9,       (* module dependent status bit *)
                 rep_res_lack,      (* lack of repeat function resources *)
                 no_free_res,       (* no free resources available *)
                 dupl_func,         (* duplicate function *)
                 out_of_range,      (* out of range *)
                 illegal_state,     (* illegal state *)
                 data_error,        (* error found in supervisor data *)
                 lcp_unknown,       (* lcp with specified receiver_id not connected *)
                 ill_lcp_oper,      (* illegal lcp_oper *)
                 chain);            (* indicate chained supervisor messages *)

time_digit = 0..15;

sp_head_type = PACKED RECORD
                    !  receiver_id: lcp_ident_type;
                    !  sender_id: lcp_ident_type;
                    !  seq_no: byte;
                    !  sp_type: sp_type_type;
                    !  lcp_oper: lcp_oper_type;
                    !  status: SET OF sp_status_bit;
                    !  time: PACKED ARRAY(1..time_lgt) OF time_digit;
                    !  bytecount: integer;
                    END;


  (**************************)
  (****   lcp messages   ****)
  (**************************)

lcp_conn_type = RECORD      (* used in lcp connection *)
                  !  first,
```

```
                    !  last,
                    !  next: integer;
                    !  lcp_ident: lcp_ident_type;
                    END;

lcp_disc_type = lcp_conn_type;      (* used in lcp disconnection *)


  (*****************************************************************)
  (*                                                               *)
  (*                  end of external ncp environment              *)
  (*                                                               *)
  (*****************************************************************)
```

C. CNNETENV C.

This appendix consists of the standard Centernet environment, CNNETENV. It defines the values of all used result codes in the u2-field, and it should be used by all LCP modules.

```
centernet_enviroment;

(*****************************************************************************)
(*                                                                           *)
(* This enviroment contains constants (such as results codes) and types      *)
(* used in most of the modules in a CENTERNET TERMINAL CONCENTRATOR           *)
(*                                                                           *)
(* Every time the enviroment is updated, please make a copy and place        *)
(* it in the CENTERNET Enviroment File placed in room 1045                   *)
(*                                                                           *)
(*                                                                           *)
(*                    CENTERNET TC Enviroment                                *)
(*                                                                           *)
(*                    version 003, data 81.01.16                             *)
(*                                                                           *)
(*****************************************************************************)
```

```
CONST

    (*-------------------------------------------------*)
    (*                                                 *)
    (*                  result codes                   *)
    (*                                                 *)
    (*-------------------------------------------------*)

      (* basic function 0 *)
    ab_term           =  16;  (* abnormal termination/disconnection *)
    ab_user_term      =  24;  (*  -"-                 -"-            upon user request *)
    ok                =   0;  (* processed successfully *)
    rem_ass_req       =   8;  (* connection/setup/establish request received *)

      (* basic function 1 *)
    rejected          =   1;  (* mess rejected by module or remote user *)
    not_processed     =   9;  (* not processed *)

      (* basic function 2 *)
    busy              =   2;  (* no internal resources, module busy *)
    net_not_accs      =  18;  (* network not accessible *)
    timeout           =  10;  (* timeout has occured *)

      (* basic function 3 *)
    no_user_res       =  11;  (* the user has not delivered the needed resource *)
    retr_exh          =  27;  (* number of retries exhausted *)
    rem_user_term     =  19;  (* disconnection/clear by remote user *)
    user_term         =   3;  (* local user has closed/cleared/disconnected the data path *)

      (* basic function 4 *)
    buf_lgth_err      =  28;  (* buffer length error *)
    fct_not_allw      =  12;  (* function not allowed *)
    format_err        =  20;  (* format error in message *)
    ill_opcode        =   4;  (* illegal ( unknown ) opcode *)
    not_mess          =  44;  (* value in u2 of message header <> 7 *)
    rec_unkw          =  36;  (* receiver unknown *)
```

```
TYPE

   (*-------------------------------------------------*)
   (*                                                 *)
   (* this section contains types used in the process *)
   (* head of every module's main process (the pro-   *)
   (* cess created by CNADAM )                        *)
   (*                                                 *)
   (*-------------------------------------------------*)


     (* input semaphore (pointer) of each module      *)

cn_sem_type  =  (ncpsem,hdlc0sem, hdlc1sem, hdlc2sem, hdlc3sem, dtesem, tssem,
                 scsem, timeout_sem, ?, ?, ?, ?, ?, new1sem, new2sem, new3sem);

cn_sem_vector = ARRAY (cn_sem_type) OF ! sempointer;


  (* record used in communication with the NCP      *)

lcp_ident_type = PACKED RECORD
                        !  i    : bit;
                        !  id   : integer;
                        END;



. (* end of centernet general enviroment *)
```

RETURN LETTER

Title: NCP Data Structures, Reference Manual Revision 1.02 RCSL No.: 43-GL11424

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

42-i 1288

. . . . . Fold here . . . . .

. . . . . Do not tear - Fold here and staple . . . . .

Affix
postage
here

REGNECENTRALEN
af 1979

Information Department
Lautrupbjerg 1
DK-2750 Ballerup
Denmark

internt

EMNE			Side:
NCTH-nyt			1/2
Udarbejdet af:	Afd.nr.:	Dato:	
UH/jfe	25	1981.08.19	PAXNET.UH.31

DISTRIBUTION	
TIL	ADR.
JLI	
CHH	
UH	
ERLP	
PEHØ	
PET	
VIL	
IMTH	
PEHO	

HALLO HALLO!

Nogle ændringer i NCTH/FORMATTER:

1. Messages fra NCP/LCP'er med LCPOPER = 3 kan tolkes som eventbufre, hvor datadelen overholder formatet:

   $\{ \langle eventnr \rangle \langle bytecount \rangle \{ \langle databyte \rangle \}^{bytecount} \}_{1}^{*}$

   Hvor hvert event udskrivens på formen:

   YY.MM.DD.HH.MM.SS. <lcpadr><eventnr>: $\{ \langle databyte \rangle \}^{bytecount}$

   Hvis antallet af databytes er for stort til at kunne være på én linie, benyttes istedet det sædvanlige dataudskriftsformat.

   Event-udskriftsformatet aktiveres ved at sætte

   form:prevent = 1;

2. Det er nu muligt at få udskrevet data på ord-basis (16 bit) ved at aktivere

   form:prword = 1;

3. Det er også muligt at få udskrift af data tilpasset 120-tegns-format ved aktivering af

   form:pform = 1;

4. Endelig et par småændringer i udskriftsformatet af Lcp-messages. De to første linier er slået sammen og slutlinien 'END OF MESSAGE' er fjernet.

Følgende er en opsummering af gældende format-options:

form:prhead = 0/1	Udskrift af LCP-head på simpel/ udvidet form.
form:prdata = 0/1;	Undertryk/udskriv data.
form:prvert = 0/1;	Udskriv data horisontalt/vertikalt (tabelform).
form:prword = 0/1;	Udskriv data på byte/word-basis.

internt

EMNE			Side:
NCTH-nyt			2/2
Udarbejdet af: UH/jfe	Afd.nr.: 25	Dato: 1981.08.10	PAXNET.UH.31

DISTRIBUTION

TIL	ADR.

form:pform = 0/1; Benyt 80/120-tegns format.

form:prevent = 0/1; Undertryk/benyt speciel event-format.

form:keeplm = 0/1; Retuner/behold sidst udskrevne.

Ved opstart af NCTH er alle på nær prdata default = 0.

GOD FORNØJELSE

Udarbejdet af: UH

Afd.nr.: 70

Dato: 810519

intern

DISTRIBUTION

TIL	ADR.
PET	
IMTH	
PEHØ	
PEHO	
VIL	
BIJ	
TSG	
CHH	
UH	
JLI	

Nogle modifikationer af NCTH.

1. Ved formatter-option:

   prhead=1

   vil TIME-feltet blive udskrevet som:

   TIME: <y> <m> <d> <h><mi><s>

   Ved formatter-option:

   prhead=0

   vil status-feltet blive skrevet ud, hvis status <>0.

2. En parameter kan tildeles værdien NIL (svarer til I i NCTH-manualen) ved operationen:

   <parm_id> = ;

   Hvis f. eks. d1 i 'session' sættes til NIL vil operationen:

   sendm: 1,1 ;

   sende en supervisormessage med bytecount = 0 .

3. PAXNET-relevant.

   De implementerede lcp-operationer i paxnet-modulerne

   ncp, router, lic og dce

   kan nu kaldes fra NCTH med operationer af typen:

   <module_id> : <lcpop>,<inc>,<data> ;

   Beskrivelsen af disse operationer er under vejs.

   (Eksisterer for routeren)

   Operationerne kan også aktiveres som subsessioner. Dette sker ved et kald af typen:

   <module_id> $\{$:,<inc>,<data>$\}_0^1$;

   d.v.s. <lcpop> udelades.
   Herefter vil man kunne (re-)specificere <inc> og <data> som iøvrigt vil være permanente under subsessionsfasen.

   Lcp-kommandoer fyres af ved operationer af typen

   <lcpop> $\{$,<inc>,<data>$\}_0^1$;

   Mellem hver af disse operationer svarer NCTH med:

   message submitted
   ?

Udarbejdet af: UH

Afd.nr.: 70

Dato: 810519

DISTRIBUTION

TIL	ADR.

Subsessionen afsluttes med :

```
end;
```

Husk at data er permanent under subsessionen, d.v.s. data specificeret for en forgående lcp-operation vil være uændret i den næste operation, med mindre det respecificeres evt. som ovenfor under punkt 2.

Eksempel på subsession:

```
→ rout;
  ?
→ crecon,0,0-1-1-1-0-3-0-125-0-4-0-0-0-0;
  message submitted
  ?
→ sencon,,0-1;
  message submitted
  ?
→ end;
  <
```

De med → angivne linier er bruger-input.

M . V . H.

Uffe Harksen

Uffe Harksen.