

DD 118:1985

7.9 INACTIVE NETWORK LAYER PROTOCOL

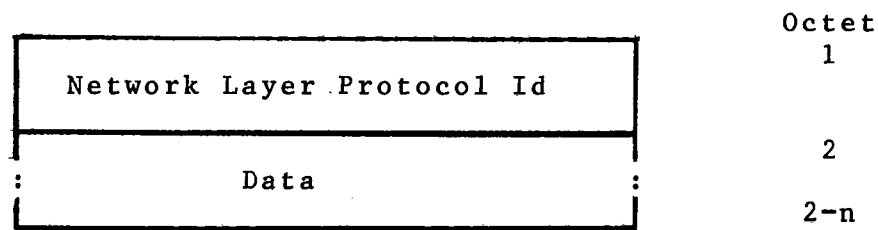


Figure 7-9. Inactive Network Layer Protocol

7.9.1 Network Layer Protocol Id

The value of the Network Layer Protocol Identifier field is binary zero (0000 0000).

7.9.2 Data Field

See Section 7.7.

The length of the NS_Userdata parameter is constrained to be less than or equal to the value of the length of the SN_Userdata parameter minus one.

[ISO page 42]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

7.10 ERROR REPORT PDU (ER)

7.10.1 Structure

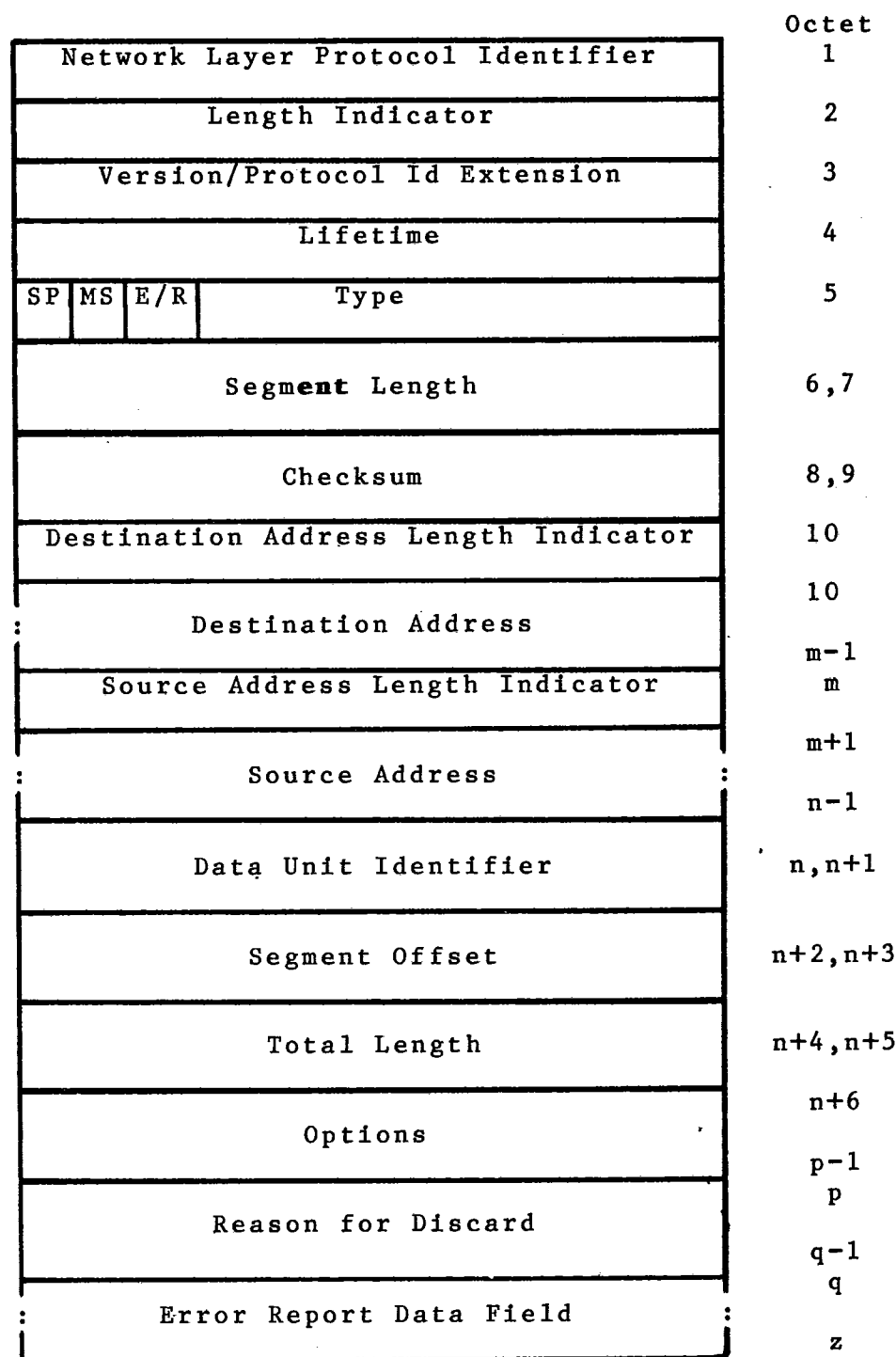


Figure 7-10. Error Report PDU.

[ISO page 43]

7.10.1.1 Fixed Part

The fixed part of the Error Report Protocol Data Unit is set as though this is a new (Initial) PDU. Thus, references are provided to previous sections describing the composition of the fields comprising the fixed part:

- | | |
|--------------------------------------|--------------------|
| 1) Network Layer Protocol Identifier | See Section 7.2.2. |
| 2) Length Indicator | See Section 7.2.3. |
| 3) Version/Protocol Id Extension | See Section 7.2.4. |
| 4) Lifetime | See Section 7.2.5. |
| 5) SP, MS, E/R | See Section 7.2.6. |
| 6) Type Code | See Section 7.2.7. |
| 7) Segment Length | See Section 7.2.8. |
| 8) Checksum | See Section 7.2.9. |

7.10.1.2 Addresses

See Section 7.3.

The Destination Address specifies the original source of the discarded PDU. The Source Address specifies the intermediate system or end system network-entity initiating the Error Report PDU.

7.10.1.3 Segmentation

See Section 7.4.

7.10.1.4 Options

See Section 7.5.

7.10.1.5 Reason for Discard

This parameter is only valid for the Error Report PDU. It provides a report on the discarded protocol data unit.

Parameter Code:	1100 0001
Parameter Length:	two octets
	type of error encoded in binary:
	0000 0000: Reason not specified.
	0000 0001: Protocol Procedure Error.
	other than below:
	0000 0010: Incorrect checksum.
	0000 0011: PDU discarded due to congestion.
	0000 0100: Header syntax error (header cannot be passed)

[ISO page 44]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

0000 0101: Segmentation is needed but is not permitted.

1000 xxxx: Addressing Error:
 1000 0000: Destination Address Unreachable.
 1000 0001: Destination Address Unknown.

1001 xxxx: Source Routing Error:
 1001 0000: Unspecified Source Routing error.
 1001 0001: Syntax error in Source Routing field.
 1001 0010: Unknown Address in Source Routing field.
 1001 0011: Path not acceptable.

1010 xxxx: Lifetime Expiration:
 1010 0000: Lifetime expired while data unit in transit.
 1010 0001: Lifetime expired during reassembly.

1011 xxxx: PDU discarded due to unsupported option:
 1011 0000: unsupported option not specified.
 1011 0001: unsupported padding option.
 1011 0010: unsupported security option.
 1011 0011: unsupported source routing option.
 1011 0100: unsupported recording of route option.
 1011 0101: unsupported QoS Maintenance option.

The second octet contains a pointer to the field in the associated discarded PDU which caused the error. If no one particular field can be associated with the error, then this field contains the value of zero.

7.10.1.6 Error Report Data Field

This field provides all or a portion of the discarded PDU. The octets comprising this field contain the rejected or discarded PDU up to and including the octet which caused the rejection/discard.

[ISO page 45]

8FORMAL DESCRIPTION

The operation of the protocol is modelled as a finite state automaton governed by a state variable with three values. The behavior of the automaton is defined with respect to individual independent Protocol Data Units. A transition of the automaton is prompted by the occurrence of an atomic event at one of three interfaces:

- 1) an interface to the Transport Layer, defined by the service primitives of the Addendum to the Network Service Definition Covering Connectionless-mode Transmission;
- 2) an interface to the subnetwork service provider, defined by the SN_UNITDATA primitive of Section 5.5 of this Standard;
- 3) an interface to an implementation-dependent timer function defined by the TIMER primitives described in Section 5.6 of this Standard.

In addition, a transition of the automaton may be prompted by the occurrence of a condition of the automaton.

The atomic events are defined in Section 8.2. The occurrence of an atomic event is not in itself sufficient to cause a transition to take place; other conditions, called "enabling conditions" may also have to be met before a particular transition can take place. Enabling conditions are boolean expressions that depend on the values of parameters associated with the corresponding atomic event (that is, the parameters of some primitive), and on the values of locally maintained variables.

More than one enabling condition--and therefore, more than one possible transition-- may be associated with a single atomic event. In every such case, the enabling conditions are mutually exclusive, so that for any given combination of atomic event and parameter values, only one state transition can take place.

Associated with each transition is an action, or "output". Actions consist of changes to the values of local variables and the sequential performance of zero or more functions. The operation of the finite state automaton is completely specified in Section 8.3 by defining the action associated with every possible transition.

[ISO page 46]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

8.1 VALUES OF THE STATE VARIABLE

The protocol state variable has three values:

- 1) INITIAL The automaton is created in the INITIAL state. No transition may carry the automaton into the INITIAL state.
- 2) REASSEMBLING The automaton is in the REASSEMBLING state for the period in which it is assembling PDU segments into a complete PDU.
- 3) CLOSED The final state of the automaton is the CLOSED state. When the automaton enters the CLOSED state it ceases to exist.

8.2 ATOMIC EVENTS

An atomic event is the transfer of a unit of information across an interface. The description of an atomic event specifies a primitive (such as an N_UNITDATA.Request), and the service boundary at which it is invoked (such as the Network Service boundary). The direction of information flow across the boundary is implied by the definition of each of the primitives.

8.2.1 N_UNITDATA_request and N_UNITDATA_indication

The N_UNITDATA_request and N_UNITDATA_indication atomic events occur at the Network Service boundary. They are defined by the Addendum to the Network Service Definition Covering Connectionless Data Transmission (ISO 8348/DAD1).

N_UNITDATA_request (NS_Source_Address,
 NS_Destination_Address,
 NS_Quality_of_Service,
 NS_Userdata)

N_UNITDATA_indication (NS_Source_Address,
 NS_Destination_Address,
 NS_Quality_of_Service,
 NS_Userdata)

The parameters of the N_UNITDATA_request and N_UNITDATA_indication are collectively referred to as Network Service Data Unit (NSDUs).

[ISO page 47]

DD 118:1985

8.2.2 SN.UNITDATA_request and SN.UNITDATA_indication

The SN.UNITDATA_request and SN.UNITDATA_indication atomic events occur at the interface between the Protocol described herein and a subnetwork service provider. They are defined in Section 5.5 of this Standard.

```
SN.UNITDATA_request (SN_Source_Address,
                     SN_Destination_Address,
                     SN_Quality_of_Service,
                     SN_Userdata)
```

```
SN.UNITDATA_indication (SN_Source_Address,
                        SN_Destination_Address,
                        SN_Quality_of_Service,
                        SN_Userdata)
```

The parameters of the SN_UNITDATA request and SN_UNITDATA Indication are collectively referred to as Subnetwork Service Data Units (SNSDUs).

The value of the SN_Userdata parameter may represent an Initial PDU or a Derived PDU.

8.2.3 TIMER Atomic Events

The TIMER atomic events occur at the interface between the Protocol described herein and its local environment. They are defined in Section 5.6 of this Standard.

```
S.TIMER_request (Time,
                 Name,
                 Subscript)
```

```
S.TIMER_cancel (Name
                Subscript)
```

```
S.TIMER_response (Name,
                  Subscript)
```

[ISO page 48]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

8.3

OPERATION OF THE FINITE STATE AUTOMATON

The operation of the automaton is defined by use of the formal description technique and notation specified in ISO/TC97/SC16 N1825, A Formal Description Technique Based on an Extended State Transition Model. This technique is based on an extended finite state transition model and the Pascal programming language. The technique makes use of strong variable typing to reduce ambiguity in interpretation of the specification.

This specification formally specifies an abstract machine which provides a single instance of the Connectionless-mode Network Service by use of the Protocol. It should be emphasized that this formal specification does not in any way constrain the internal operation or design of any actual implementation. For example, it is not required that the program segments contained in the state transitions will actually appear as part of an actual implementation. A formal protocol specification is useful in that it goes as far as possible to eliminate any degree of ambiguity or vagueness in the specification of a protocol standard.

The formal specification contained herein specifies the behavior of a single finite-state machine, which provides the protocol behavior corresponding to a single independent service request. It is expected that any actual implementation will be able to handle behavior corresponding to many simultaneous finite state machines.

[ISO page 49]

Documents

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

Document Sharing Hub

8.3.1 Type and Constant Definitions

```

const
    ZERO    = 0;
    max_user_data = 64512;

type
    NSAP_addr_type = ...;

    { NSAP_addr_type defines the data type for NSAP
      addresses, as passed across the Network Service
      boundary. }

    NPAI_addr_type = ...;

    { NPAI_addr_type defines the data type for the
      addresses carried in PDUs. }

    SN_addr_type   = ...;

    { SN_addr_type defines the data type for addresses
      in the underlying service used by this protocol. }

    quality_of_service_type = ...;

    { quality_of_service_type defines the data type
      for the QOS parameter passed across the Network
      Service boundary. }

    SN_QOS_type    = ...;

    { SN_QOS_type defines the data type for the QOS
      parameter, if any, passed to the underlying
      service used by this protocol. }

    data_type      = ...;

    { data_type defines the data type for user data.
      Conceptually, this is equivalent to a variable
      length binary string. }

    buffer_type    = ...;

    { buffer_type defines the data type for the memory
      resources used in sending and receiving of user
      data. This provides capabilities required for
      segmentation and reassembly. }

```

[ISO page 50]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

```

timer_name_type = (lifetime_timer);

timer_data_type = ...;

network-layer-protocol-id-type = (ISO_8473_protocol_id);

version_id_type = (version1);

pdu_tp_type      = (DT, ER);

options_type      = ...;

{ options_type defines the data type used to store
  the options part of the PDU header.}

subnet_id_type    = ...;

{ subnet_id_type defines the data type used to
  locally identify a particular underlying service
  used by this protocol. In general there may be
  multiple underlying subnetwork (or data link)
  services.}

error_type        = (NO_ERROR,
                     TOO_MUCH_USER_DATA,
                     PROTOCOL_PROCEDURE_ERROR,
                     INCORRECT_CHECKSUM,
                     CONGESTION,
                     SYNTAX_ERROR,
                     SEG_NEEDED_AND_NOT_PERMITTED,
                     DESTINATION_UNREACHABLE,
                     DESTINATION_UNKNOWN,
                     UNSPECIFIED_SRC_ROUTING_ERROR,
                     UNKNOWN_ADDRESS_IN_SRC_ROUTING,
                     PATH_NOT_ACCEPTABLE_IN_SRC_ROUTING,
                     LIFETIME_EXPIRED_IN_TRANSIT,
                     LIFETIME_EXPIRED_IN_REASSEMBLY,
                     UNSUPPORTED_OPTION_NOT_SPECIFIED,
                     UNSUPPORTED_SRC_ROUTING_OPTION,
                     UNSUPPORTED_QOS_MAINTENANCE_OPTION);

```

[ISO page 51]

```

nsdu_type = record
    da      : NSAP_addr_type;
    sa      : NSAP_addr_type;
    qos     : quality_of_service_type;
    data    : data_type;
end;

pdu_type = record
    nlp_id   : network_layer_protocol_id_type;
    hli      : integer;
    vp_id    : version_id_type;
    lifetime : integer;
    sp       : boolean;
    ms       : boolean;
    er_flag  : boolean;
    pdu_tp   : pdu_tp_type;
    seg_len  : integer;
    checksum : integer;
    da_len   : integer;
    da       : NPAI_addr_type;
    sa_len   : integer;
    sa       : NPAI_addr_type;
    du_id    : optional integer;
    so       : optional integer;
    tot_len  : optional integer;

    { du_id, so, and tot_len are present
      only if sp has the value TRUE. }

    options  : options_type;
    data     : data_type;
end;

route_result_type =
    record
        subnet_id   : subnet_id_type;
        sn_da       : SN_addr_type;
        sn_sa       : SN_addr_type;
        segment_size : integer;
    end;

```

8.3.2 Interface Definitions

channel Network_access_point (User, Provider);

by User:

UNITDATA_request

(NS_Destination_address : NSAP_addr_type;
NS_Source_address : NSAP_addr_type;
NS_Quality_of_Service : quality_of_service_type;
NS_Userdata : data_type);

by Provider:

UNITDATA_indication

(NS_Destination_address : NSAP_addr_type;
NS_Source_address : NSAP_addr_type;
NS_Quality_of_Service : quality_of_service_type;
NS_Userdata : data_type);

channel Subnetwork_access_point (User, Provider);

by User:

UNITDATA_request

(SN_Destination_address : SN_addr_type;
SN_Source_address : SN_addr_type;
SN_Quality_of_Service : SN_QOS_type;
SN_Userdata : pdu_type);

by Provider:

UNITDATA_indication

(SN_Destination_address : SN_addr_type;
SN_Source_address : SN_addr_type;
SN_Quality_of_Service : SN_QOS_type;
SN_Userdata : pdu_type);

channel System_access_point (User, Provider);

by User:

TIMER_request

(Time : integer;
Name : timer_name_type;
Subscript : integer);

TIMER_cancel

(Name : timer_name_type;
Subscript : integer);

by Provider:

TIMER_response

(Name : timer_name_type;
Subscript : integer);

[ISO page 53]

8.3.3 Formal Machine Definition

```
module Connectionless_Network_Protocol_Machine
  (N: Network_access_point (Provider) common queue;

   SN: array [subnet_id_type] of
     Subnetwork_access_point (User) common queue;

   S : System_access_Point (User) individual queue);

var
  nsdu      : nsdu_type;
  pdu       : pdu_type;
  rcv_buf   : buffer_type;

state : (INITIAL, REASSEMBLING, CLOSED);
```

```

procedure send_error_report (error : error_type;
                             pdu   : pdu_type);

var
  er_pdu : pdu_type;

begin
  if (pdu.er_flag) then
    begin
      er_pdu.nlp_id := ISO_8473_protocol_id;
      er_pdu.vp_id  := version1;
      er_pdu.lifetime := get_er_lifetime(pdu.sa);
      er_pdu.sp      := get_er_seg_per(pdu);
      er_pdu.ms      := FALSE;
      er_pdu.er_flag := FALSE;
      er_pdu.pdu.tp  := ER;
      er_pdu.da_len  := pdu.sa_len;
      er_pdu.da      := pdu.sa;
      er_pdu.sa_len  := get_local_NPAI_addr_len;
      er_pdu.sa      := get_local_NPAI_addr;

      er_pdu.options := get_er_options(error,
                                         er_pdu.da,
                                         pdu.options);

      er_pdu.hli := get_header_length
                    (er_pdu.da_len,
                     er_pdu.sa_len,
                     er_pdu.sp,
                     er_pdu.options);

      er_pdu.data := get_er_data_field(error, pdu);

      if (er_pdu.sp) then
        begin
          er_pdu.du_id := get_er_du_id(er_pdu.da);
          er_pdu.so    := ZERO;
          er_pdu.tot_len := er_pdu.hli +
                             size(er_pdu.data)
        end;
        if (NPAI_addr_local(er_pdu.da)) then
          post_error_report(er_pdu)
        else
          send_pdu(er_pdu);
        end;
      end;
    end;
  end;
end;

```

[ISO page 55]

```

procedure send_pdu (pdu : pdu_type);

var
  rte_result : route_result_type;
  error_code : error_type;
  send_buf   : buffer_type;
  data_maxsize : integer;
  more_seg   : boolean;
  sn_qos     : SN_QOS_type;

begin
  send_buf := make_buffer(pdu.data);
  more_seg := pdu.ms;

  repeat
    begin
      error_code := check_parameters(pdu.hli,
                                     pdu.sp,
                                     pdu.da,
                                     pdu.options,
                                     size(pdu.data));

      if (error_code = NO_ERROR) then
        begin
          rte_result := route(pdu.hli,
                              pdu.sp,
                              pdu.da,
                              pdu.options,
                              size(pdu.data));

          data_maxsize := rte_result.segment_size - pdu.hli;
          pdu.data      := extract(send_buf, data_maxsize);
          pdu.seg_len   := pdu.hli + size(pdu.data);

          if (size(send_buf) = ZERO) then
            begin
              pdu.ms := more_seg;
            end
          else
            pdu.ms := TRUE;

          pdu.checksum := get_checksum(pdu);
          sn_qos       := get_sn_qos(rte_result.subnet_id,
                                     pdu.options);

          out SN[rte_result.subnet_id].UNITDATA_request
            (rte_result.sn_da,
             rte_result.sn_sa,
             sn_qos,
             pdu);

          pdu.so := pdu.so + data_maxsize;
        end
      end
    end
  until error_code = NO_ERROR;
end

```

[ISO page 56]

```
else if (error_code = CONGESTION) then
  begin
    if (send_er_on_congestion(pdu)) then
      send_error_report(CONGESTION, pdu);
    end
  else
    send_error_report(error_code, pdu);
  end;
until (size_buf(data_buf) = ZERO) or
      (error_code <> NO_ERROR);
end;
```

[ISO page 57]

DD 118:1985

```

procedure allocate_reassembly_resources
  (pdu_tot_len : integer);
primitive;

```

```

{ This procedure allocates resources required for reassembly
  of a PDU of the specified total length. It is a local
  matter as to whether this procedure is called once (upon
  reception of the first Derived PDU or the Initial PDU),
  or each time a Derived PDU is received. If this requires
  discarding of a PDU in which the ER flag is set, then an
  error report is returned to the source of the discarded
  data unit. }

```

```

function check_parameters
  (hli      : integer;
   sp       : boolean;
   da       : NPAI_addr_type;
   options  : options_type;
   datalen  : integer) : error_type;
primitive;

```

```

{ This function determines various parameters associated
  with a PDU, to determine whether forwarding of the PDU
  can continue. If a result of NO_ERROR is returned,
  then the primitive route can be called to specify the
  route and segment size. Otherwise, otherwise this
  function specifies the reason that an error has occurred.}

```

```

function data_unit_complete
  (buf : buffer_type) : boolean;
primitive;

```

```

{ This function returns a boolean value specifying whether
  the PDU stored in the specified buffer has been completely
  received. }

```

```

function elapsed_time : integer;
primitive;

```

```

{ This function returns an estimate of the time elapsed,
  in 500 millisecond increments, since the PDU was
  transmitted by the previous peer network-entity. This
  estimate includes both time spent in transit, and any
  time to be spent in buffers within the local system.
  Although this estimate need not be precise, overestimates
  are preferable to underestimates, as underestimating the
  time elapsed may defeat the intent of the lifetime
  function. }

```

[ISO page 58]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

```

procedure empty_buffer
    (buf : buffer_type);
primitive;

{ This procedure empties the specified buffer. }

function extract
    (buf : buffer_type;
     amount : integer_type) : data_type;
primitive;

{ This function removes the specified amount of data from
  the specified buffer, and returns this data as the
  function value. }

procedure free_reassembly_resources;
primitive;

{ This procedure releases the resources that had been
  previously allocated by the procedure
  allocate_reassembly_resources. }

function get_checksum
    (pdu : pdu_type) : integer_type;
primitive;

{ This function returns the 16-bit integer value to be
  placed in the checksum field of the PDU. If the
  checksum facility is not being used, then this function
  returns the value zero. The algorithm for producing a
  correct checksum value is specified in Annex C.}

function get_data_unit_id
    ( da : NPAI_addr_type ) : integer;
primitive;

{ This function returns a data unit identifier which is
  unique for the specified destination address. }

function get_er_data_field
    (error : error_type;
     pdu : pdu_type) : data_type;
primitive;

{ This function returns the correct data field for an error
  report, based on the information that the specified PDU is
  being discarded due to the specified error. The data field
  of an error report must include the header of the discarded
  PDU, and may optionally contain additional user data. }

```

[ISO page 59]

```

function get_er_flag
    (nsdu : nsdu_type) : boolean;
primitive;

{ This function returns a boolean value to be used as the
  error report flag in a PDU which transmits the specified
  nsdu. If the PDU must be discarded at some future time,
  an error report can be returned only if this value is set
  to TRUE. }

function get_er_lifetime
    (da : NPAI_addr_type) : integer;
primitive;

{ This function returns the lifetime value to be used for
  an error report being sent to the specified destination
  address. }

function get_er_options
    (error : error_type;
     da : NPAI_addr_type;
     options : options_type) : options_type;
primitive;

{ This function returns the options field of an error
  report, based on the reason for discard, and the
  destination address and options field of the discarded
  PDU. The options field contains the reason for discard
  option, and may contain other optional fields. }

function get_er_seg_per
    (pdu : pdu_type) : boolean_type;
primitive;

{ This function returns the boolean value which will be used
  for the segmentation permitted flag of an error report. }

function get_header_len
    (da_len : integer;
     sa_len : integer;
     sp : boolean;
     options : options_type) : integer;
primitive;

{ This function returns the header length, in octets. The
  header length depends upon the lengths of the source and
  destination addresses, whether the segmentation part is
  of the header is present, and the length of the options
  part. }

```

```
functions get_lifetime
    (da      : NSAP_addr_type;
     qos     : quality_of_service_type_ : lifetime_type;
primitive;

{ This function returns the lifetime value to be used for
  a PDU, based upon the destination address and requested
  quality of service. }

function get_local_NPAI_addr : NPAI_address_type;
primitive;

{ This function returns the local address as used in the
  protocol header. }

function get_local_NPAI_addr_len : integer;
primitive;

{ This function returns the length of the local address
  as used in the protocol header. }

function get_NPAI
    (addr : NSAP_addr_type) : NPAI_addr_type;
primitive;

{ This function returns the network address as used in
  the protocol header, or Network Protocol Address
  Information, corresponding to the specified NSAP
  address. }

function get_NPAI_len
    (addr : NSAP_addr_type) : integer;
primitive;

{ This function returns the length of the Network
  Protocol Address Information corresponding
  to a specified NSAP address. }

function get_NSAP_addr
    (addr : NPAI_addr_type;
     len  : integer_type) : NSAP_addr_type;
primitive;

{ This function returns the NSAP address corresponding to
  the Network Protocol Address Information (as it
  appears in the protocol header) of the specified length. }
```

```

function get_options
    (da      : NSAP_addr_type;
     qos     : quality_of_service_type) : options_type;
primitive;

{ This function returns the options field for a PDU, based
  on the requested destination address and quality of
  service. }

function get_seg_permitted
    (da      : NSAP_addr_type;
     qos     : quality_of_service_type) : boolean;
primitive;

{ This function returns the boolean value to be used in the
  segmentation permitted field of a PDU. This value may
  depend upon the destination address, requested quality of
  service, and the length of the user data. }

function get_sn_qos
    (subnet_id : subnet_id_type;
     options   : options_type) : SN_QOS_type;
primitive;

{ This function returns the quality of service to be used
  on the specified subnetwork, in order to obtain the
  quality of service (if any) and other parameters requested
  in the options part of the PLU. }

function get_qos
    (options : options_type) quality_of_service_type;
primitive;

{ This function determines, to the extent possible, the
  quality of service that was obtained for a particular
  PDU, based upon the quality of service and other
  information contained in the options part of the PDU
  header. }

function make_buffer
    (data : data_type) ; buffer_type;
primitive;

{ This function places the specified data in a newly
  created buffer. The precise manner of handling buffers
  is a local matter. This newly created buffer
  is returned as the function value. }

```

[ISO page 62]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

```
procedure merge_seg
    (buf      : buffer_type;
     so       : integer;
     data     : data_type);
primitive;

{ This procedure merges the specified data into the
  specified buffer, based on the specified segment
  offset of the data. }

function NPAI_addr_local
    (addr : NPAI_addr_type) : boolean;
primitive;

{ This function returns the boolean value TRUE only if the
  specified Network Protocol Address Information
  specifies a local address. }

function NSAP_addr_local
    (addr : NSAP_addr_type) : boolean;
primitive;

{ This function returns the boolean value TRUE only if the
  specified NSAP address specifies a local address. }

procedure post_error_report
    (er_pdu : pdu_type);
primitive;

{ This procedure posts the specified error report (ER) type
  PDU to the appropriate local entity that handles error
  reports. }
```

[ISO page 63]

```

function route
    (hli      : integer;
     sp       : boolean;
     da       : NPAI_addr_type;
     options  : options_type;
     datalen  : integer) : route_result_type;
primitive;

```

```

{ This function determines the route to be followed by a
  PDU segment, as well as the segment size. Note that
  in general, the segment size and route may be mutually
  dependent. This determination is made on the basis of
  the header length, the segmentation permitted flag, the
  destination address, several parameters (such as source
  routing) contained in the options part of the PDU header,
  and the length of data. This function returns a structure
  that specifies the subnetwork on which the segment should
  be transmitt, the source and destination addresses to
  be used on the subnetwork, and the segment size. This
  routine may only be called if the primitive function
  check_parameters has already determined that an error
  will not occur. }

```