Network Provider Interface Specification

UNIX International
OSI Work Group
Revision: 2.0.0 (August 17, 1992)

Published by:

UNIX International
Waterview Corporate Center
20 Waterview Boulevard
Parsippany, NJ 07054

for further information, contact:
Vice President of Marketing

Phone: +1 201-263-8400
Fax: +1 201-263-8401

International Offices:

UNIX International	UNIX International	UNIX International	UNIX International
Asian/Pacific Office	Australian Office	European Office	Pacific Basin Office
Shinei Bldg. 1F	22/74 - 76 Monarch St.	25, Avenue de Beaulieu	Cintech II
Kameido	Cremorne, NSW 2090	1160 Brussels	75 Science Park Drive
Koto-ku, Tokyo 136	Australia	Belgium	Singapore Science Park
Japan			Singapore 0511
			Singapore
Phone:(81) 3-3636-1122	Phone:(61) 2-953-7838	Phone:(32) 2-672-3700	Phone:(65) 776-0313
Fax: (81) 3-3636-1121	Fax: (61) 2 953-3542	Fax: (32) 2-672-4415	Fax: (65) 776-0421

NOTICE:

This document is based on the UNIX System Laboratories Network Provider Interface (NPI) specification which was used with permission by the UNIX International OSI Work Group (UI OSIWG). Participation in the UI OSIWG is open to UNIX International members and other interested parties. For further information contact UNIX International at the addresses above.

UNIX International is making this documentation available as a reference point for the industry. While UNIX International believes that these interfaces are well defined in this release of the document, minor changes may be made prior to products conforming to the interfaces being made available from UNIX System Laboratories or UNIX International members.

Trademarks:

1. Introduction

This document specifies a STREAMS-based kernel-level instantiation of the ISO/CCITT network service definition. The Network Provider Interface (NPI) enables the user of a network layer service to access and use any of a variety of conforming network layer service providers without specific knowledge of the provider's protocol. The service interface is designed to support any connection-mode network protocol and connectionless network protocol. This interface only specifies access to network layer service providers, and does not address issues concerning network layer management, protocol performance, and performance analysis tools.

The specification assumes that the reader is familiar with the OSI reference model terminology, ISO/CCITT Network Layer Service, and STREAMS.

1.1 Related Documentation

— **1986 CCITT X.213 Recommendation** [1]

— **ISO 8348** [2]

— **ISO 8348/AD1** [3]

— **ISO 8473** [4]

— **ISO 8208** [5]

— **ISO 8878** [6]

— **System V Interface Definition, Issue 2 - Volume 3** [7]

1.1.1 Role

This document specifies an interface that supports the service provided by the Network Services Definition for Open Systems Interconnection for CCITT Applications as described in CCITT Recommendation X.213 and ISO 8348 (for CONS) and ISO 8348/Addendum 1 (for CLNS). These specifications are targeted for use by developers and testers of protocol modules that require network layer service.

1.2 Definitions, Acronyms, and Abbreviations

Calling NS user
: An NS user that initiates a Network Connection (NC).

Called NS User
: An NS user with whom a calling NS user wishes to establish a network connection (NC).

CLNP
: Connection-less Network Protocol

CLNS
: Connection-less Network Service

CONP
: Connection Oriented Network Protocol

CONS
: Connection Oriented Network Service

DLSAP
: Data Link Service Access Point

ISO
: International Organization for Standardization

NC
: Network Connection

Network User
: Kernel level protocol or user level application that is accessing the services of the network layer.

Network Provider
: Network layer entity/entities that provide/s the services of the network interface.

NPI
: Network Provider Interface

NS
: Network Service

NIDU
: Network Interface Data Unit

NSAP
: Network Service Access Point

NSDU
: Network Service Data Unit

OSI
: Open Systems Interconnection

QOS
: Quality of Service

STREAMS
: A communication services development facility first available with UNIX System V Release 3

2. The Network Layer

The Network Layer provides the means to manage the operation of the network. It is responsible for the routing and management of data exchange between network-user entities.

2.1 Model of the NPI

The NPI defines the services provided by the network layer to the network-user at the boundary between the network layer and the network layer user entity. The interface consists of a set of primitives defined as STREAMS messages that provide access to the network layer services, and are transferred between the NS user entity and the NS provider. These primitives are of two types; ones that originate from the NS user, and others that originate from the NS provider. The primitives that originate from the NS user make requests to the NS provider, or respond to an event of the NS provider. The primitives that originate from the NS provider are either confirmations of a request or are indications to the NS user that the event has occurred. Figure 1 shows the model of the NPI.

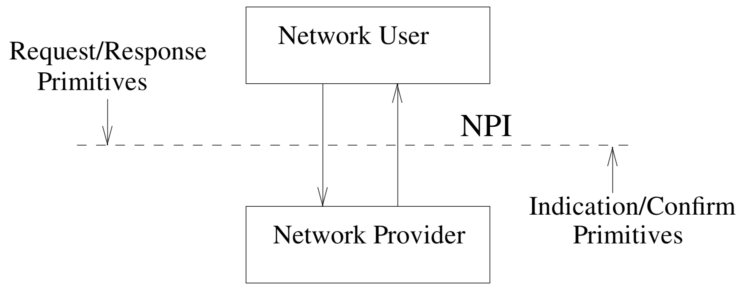

Figure 1. Model of the NPI

The NPI allows the NS provider to be configured with any network layer user (such as the OSI Transport Layer) that also conforms to the NPI. A network layer user can also be a user program that conforms to the NPI and accesses the NS provider via "putmsg" and "getmsg" system calls.

2.2 NPI Services

The features of the NPI are defined in terms of the services provided by the NS provider, and the individual primitives that may flow between the NS user and the NS provider.

The services supported by the NPI are based on two distinct modes of communication, connection (CONS) and connectionless (CLNS). In addition, the NPI supports services for local management.

CONS

The main features of the connection mode communication are:

a. It is virtual circuit oriented;
b. It provides transfer of data via a pre-established path;
c. It provides reliable data transfer.

There are three phases to each instance of communication: Connection Establishment; Data Transfer; and Connection Termination. Units of data arrive at their destination in the same order as they departed their source and the data is protected against duplication or loss of data units within some specified quality of service.

CLNS

The main features of the connectionless mode communication are:

a. It is datagram oriented;
b. It provides transfer of data in self contained units;
c. There is no logical relationship between these units of data;
d. It is unreliable.

Connectionless mode communication has no separate phases. Each unit of data is transmitted from source to destination independently, appropriate addressing information is included with each unit of data. As the units of data are transmitted independently from source to destination, there are, in general, no guarantees of proper sequence and completeness of the data stream.

Local Management

The NPI specifications also define a set of local management functions that apply to both CONS and CLNS modes of communication. These services have local significance only.

Tables 1 and 2 summarizes the NPI service primitives by their state and service.

STATE	SERVICE	PRIMITIVES
Local Management	Information Reporting	N_INFO_REQ, N_INFO_ACK N_ERROR_ACK
	Bind	N_BIND_REQ, N_BIND_ACK, N_UNBIND_REQ, N_OK_ACK, N_ERROR_ACK
	Options Management	N_OPTMGMT_REQ, N_OK_ACK, N_ERROR_ACK
Connection Establishment	Connection Establishment	N_CONN_REQ, N_CONN_IND, N_CONN_RES, N_CONN_CON, N_TOKEN_REQ, N_TOKEN_ACK, N_OK_ACK, N_ERROR_ACK
Connection Mode Data Transfer	Data Transfer	N_DATA_REQ, N_DATA_IND, N_EXDATA_REQ, N_EXDATA_IND, N_DATACK_REQ, N_DATACK_IND
	Reset	N_RESET_REQ, N_RESET_IND, N_RESET_RES, N_RESET_CON
Connection Release	Connection Release	N_DISCON_REQ, N_DISCON_IND, N_OK_ACK, N_ERROR_ACK

TABLE 1. Service Primitives for Connection Mode Data Transfer

STATE	SERVICE	PRIMITIVES
Local Management	Information Reporting	N_INFO_REQ, N_INFO_ACK N_ERROR_ACK
	Bind	N_BIND_REQ, N_BIND_ACK, N_UNBIND_REQ, N_OK_ACK, N_ERROR_ACK
	Options Management	N_OPTMGMT_REQ, N_OK_ACK, N_ERROR_ACK
Connectionless Mode Data Transfer	Data Transfer	N_UNITDATA_REQ, N_UNITDATA_IND N_UDERROR_IND

TABLE 2. Service Primitives for Connectionless Mode Data Transfer

3. NPI Services Definition

This section describes the services of the NPI primitives. Time-sequence diagrams that illustrate the sequence of primitives are included. (Conventions for the time-sequence diagrams are defined in CCITT X.210 [8].) The format of the primitives will be defined later in this document.

3.1 Local Management Services Definition

The services defined in this section are outside the scope of the international standards. These services apply to both connection-mode as well as the connection-less modes of communication. They are invoked for the initialization/de-initialization of a stream connected to the NS provider. They are also used to manage options supported by the NS provider and to report information on the supported parameter values.

3.1.1 Network Information Reporting Service

This service provides information on the options supported by the NS provider.

- **N_INFO_REQ** : This primitive requests that the NS provider return the values of all the supported protocol parameters. This request may be invoked during any phase.
- **N_INFO_ACK** : This primitive is in response to the N_INFO_REQ primitive and returns the values of the supported protocol parameters to the NS user.

The sequence of primitives for network information management is shown in Figure 2.

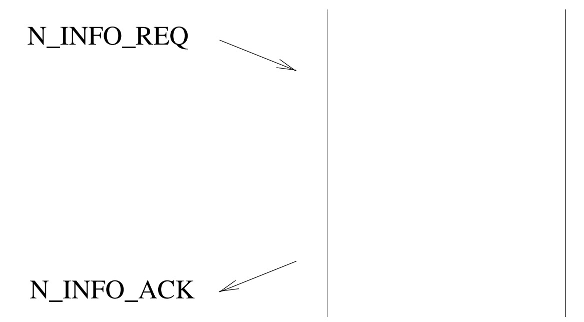

Figure 2. Sequence of Primitives: Network Information Reporting Service

3.1.2 NS User Bind Service

This service allows a network address to be associated with a stream. It allows the NS user to negotiate the number of connect indications that can remain unacknowledged for that NS user (a connect indication is considered unacknowledged while it is awaiting a corresponding connect response or disconnect request from the NS user). This service also defines a mechanism that allows a stream (bound to a network address of the NS user) to be reserved to handle incoming calls only. This stream is referred to as the listener stream.

- **N_BIND_REQ** : This primitive requests that the NS user be bound to a particular network address, and negotiate the number of allowable outstanding connect indications for that address.
- **N_BIND_ACK** : This primitive is in response to the N_BIND_REQ primitive and indicates to the user that the specified NS user has been bound to a network address.

The sequence of primitives for NS user bind service is shown in Figure 3.

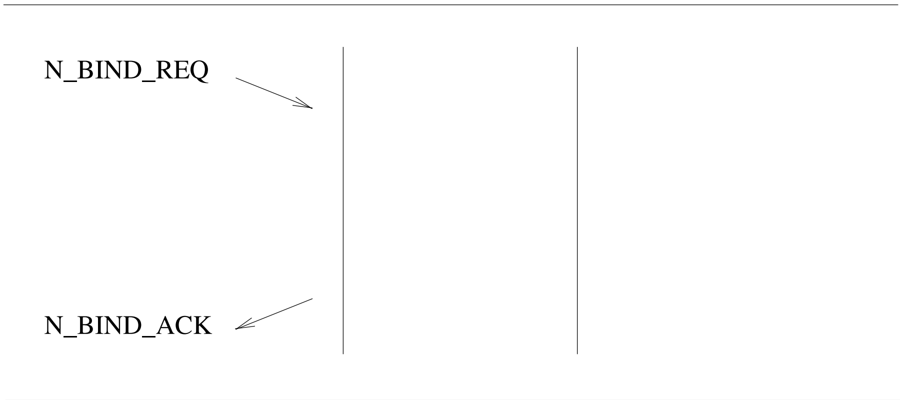

Figure 3. Sequence of Primitives: NS User Bind Service

3.1.3 NS User Unbind Service

This service allows the NS user to be unbound from a network address.

- **N_UNBIND_REQ** : This primitive requests that the NS user be unbound from the network address that it had previously been bound to.

The sequence of primitives for NS user unbind service is shown in Figure 4.

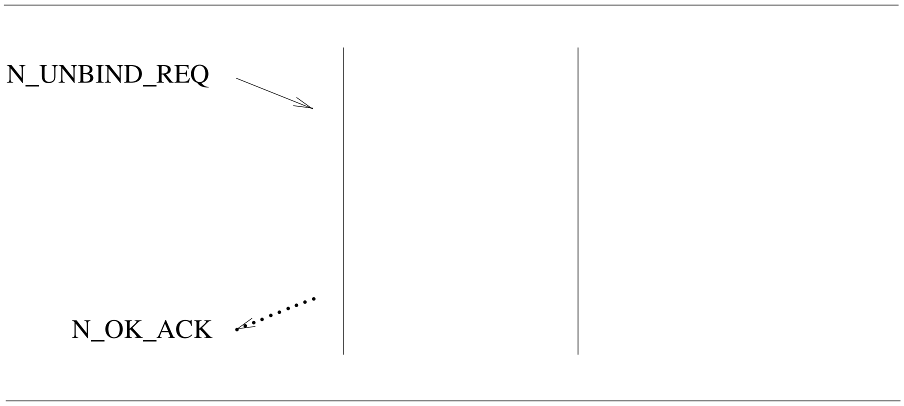

Figure 4. Sequence of Primitives: NS User Unbind & Receipt Acknowledgement Services

3.1.4 Receipt Acknowledgement Service

- **N_OK_ACK** : This primitive indicates to the NS user that the previous NS user originated primitive was received successfully by the NS provider.

An example showing the sequence of primitives for successful receipt acknowledgement is depicted in Figure 4.

3.1.5 Options Management Service

This service allows the NS user to manage the QOS parameter values associated with the NS provider.

- **N_OPTMGMT_REQ** : This primitive allows the NS user to select default values for QOS parameters within the range supported by the NS provider, and to indicate the default selection of receipt confirmation.

Figure 5 shows the sequence of primitives for network options management.

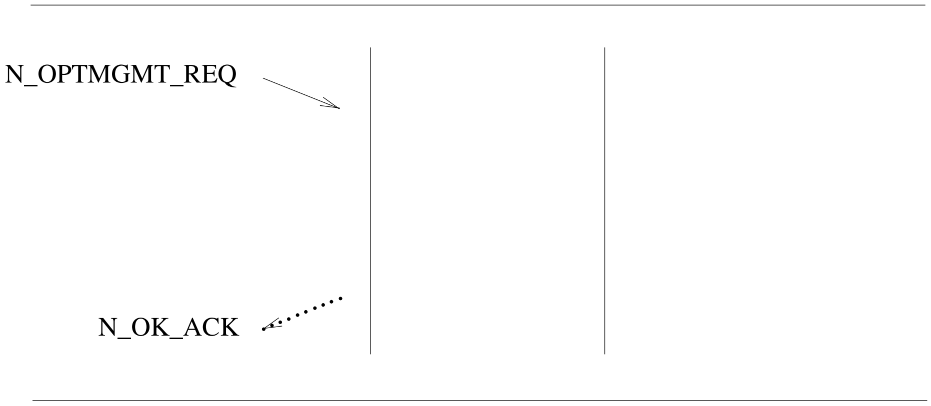

Figure 5. Sequence of Primitives: Options Management Service

3.1.6 Error Acknowledgement Service

- **N_ERROR_ACK** : This primitive indicates to the NS user that a non-fatal error has occurred in the last NS user originated request or response primitive (listed in Figure 6), on the stream.

Figure 6 shows the sequence of primitives for the error management primitive.

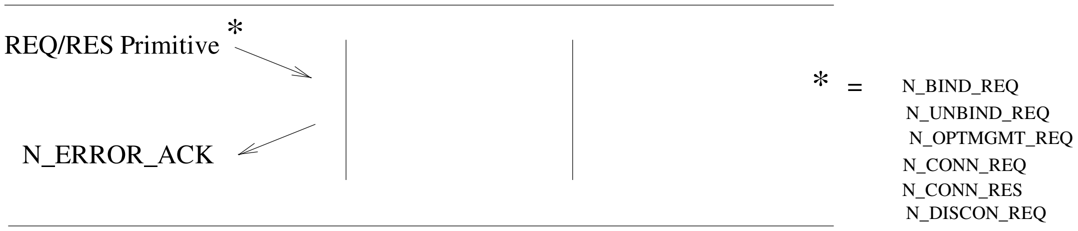

Figure 6. Sequence of Primitives: Error Acknowledgement Service

3.2 Connection-Mode Network Services Definition

This section describes the required network service primitives that define the CONS interface.

The queue model for CONS is discussed in more detail in CCITT X.213 section 9.2.

The queue model represents the operation of a network connection in the abstract by a pair of queues linking the two network addresses. There is one queue for each direction of information flow. Each queue represents a flow control function in one direction of transfer. The ability of a user to add objects to a queue will be determined by the behavior of the user removing objects from that queue, and the state of the queue. The pair of queues is considered to be available for each potential NC. Objects that are entered or removed from the queue are either as a result of interactions at the two network addresses, or as the result of NS provider initiatives.

- A queue is empty until a connect object has been entered and can be returned to this state, with loss of its contents, by the NS provider.
- Objects may be entered into a queue as a result of the actions of the source NS user, subject to control by the NS provider;
- Objects may also be entered into a queue by the NS provider.
- Objects are removed from the queue under the control of the receiving NS user.
- Objects are normally removed under the control of the NS user in the same order as they were entered except:
  - — if the object is of a type defined to be able to advance ahead of the preceding object (however, no object is defined to be able to advance ahead of another object of the same type), or
  - — if the following object is defined to be destructive with respect to the preceding object on the queue. If necessary, the last object on the queue will be deleted to allow a destructive object to be entered - they will therefore always be added to the queue. For example, "disconnect" objects are defined to be destructive with respect to all other objects. "Reset" objects are defined to be destructive with respect to all other objects except "connect", "disconnect", and other "reset" objects.

Table 3 shows the ordering relationships among the queue model objects.

Object X / Object Y	CONNECT	NORMAL DATA	EXP. NSDU	DATA ACK	RESET	DISC
CONNECT	N/A	-	-	-	-	DES
NORMAL DATA	N/A	-	AA	AA	DES	DES
EXP. NSDU	N/A	-	-	AA	DES	DES
DATA ACK	N/A	-	AA	-	DES	DES
RESET	N/A	-	-	-	-	DES
DISC	N/A	N/A	N/A	N/A	N/A	-

AA Indicates that Object X is defined to be able to advance ahead of preceding Object Y.

DES Indicates that Object X is defined to be destructive with respect to the preceding Object Y.

\- Indicates that Object X is neither destructive with respect to Object Y, nor able to advance ahead of Object Y.

N/A Indicates that Object X will not occur in a position succeeding Object Y in a valid state of a queue.

TABLE 3. Ordering Relationships Between Queue Model Objects

3.2.1 Connection Establishment Phase

A pair of queues is associated with an NC between two network addresses when the NS provider receives an N_CONNECT_REQ primitive at one of the network addresses resulting in a connect object being entered into the queue. The queues will remain associated with the NC until a N_DISCON_REQ primitive (resulting in a disconnect object) is either entered or removed from a queue. Similarly, in the queue from the called NS user, objects can be entered into the queue only after the connect object associated with the N_CONN_RES has been entered into the queue. Alternatively, the called NS user can enter a disconnect object into the queue instead of the connect object to terminate the NC.

The NC establishment procedure will fail if the NS provider is unable to establish an NC, or if the destination NS user is unable to accept the N_CONN_IND (see NC Release primitive definition).

3.2.1.1 User Primitives for Successful Network Connection Establishment

- **N_CONN_REQ** : This primitive requests that the NS provider make a connection to the specified destination.
- **N_CONN_RES** : This primitive requests that the NS provider accept a previous connection indication.

3.2.1.2 Provider Primitives for Successful Network Connection Establishment

- **N_CONN_IND** : This primitive indicates to the NS user that a connect request has been made by a user at the specified source address.

- **N_CONN_CON** : This primitive indicates to the NS user that a connect request has been confirmed on the specified responding address.

The sequence of primitives in a successful NC establishment is defined by the time sequence diagram as shown in Figure 7. The sequence of primitives for the NC response token value determination is shown in Figure 8 (procedures for NC response token value determination are discussed in sections 4.1.3 and 4.1.4.).

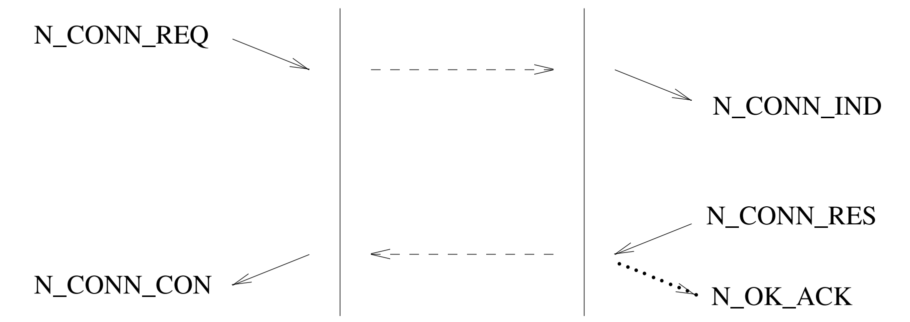

Figure 7. Sequence of Primitives: Successful NC Establishment

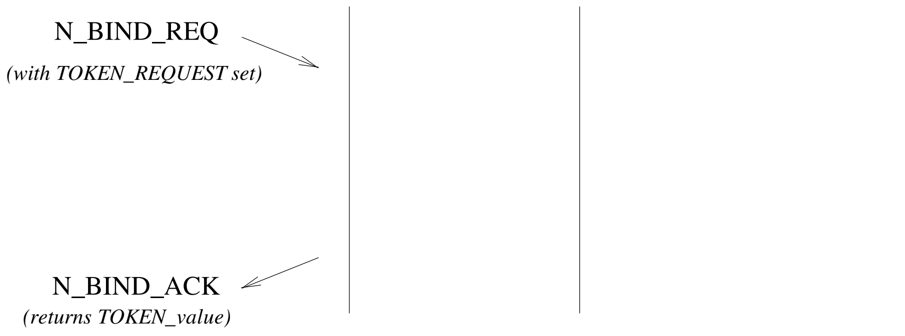

Figure 8. Sequence of Primitives: NC Response Token Value Determination

3.2.2 Data Transfer Phase

Flow control on the NC is done by management of the queue capacity, and by allowing objects of certain types to be inserted to the queues, as shown in Table 4.

OBJECT X OBJECT Y	OCTETS OF NORMAL DATA/	EXPEDITED DATA	DATA ACKNOWLEDGEMENT
Octets of Normal Data	Yes	Yes	No
Expedited Data	No	Yes	No
Data Acknowledgement	No	No	No

Yes The addition of Object X may prevent further addition of Object Y.

No The addition of Object X may not prevent the addition of Object Y.

TABLE 4. Flow Control Relationships Between Queue Model Objects

3.2.2.1 User Primitives for Data Transfer

- **N_DATA_REQ** : This primitive requests that the NS provider transfer the specified data.
- **N_DATACK_REQ** : This primitive requests that the NS provider acknowledge the data that had previously been received with receipt confirmation requested.
- **N_EXDATA_REQ** : This primitive requests that the NS provider transfer the specified expedited network service data unit.

3.2.2.2 Provider Primitives for Data Transfer

- **N_DATA_IND** : This primitive indicates to the NS user that this message contains data.
- **N_DATACK_IND** : This primitive indicates to the NS user that the remote NS user has acknowledged the data that had previously been sent with receipt confirmation requested.
- **N_EXDATA_IND** : This primitive indicates to the NS user that this message unit contains expedited data.

Figure 9 shows the sequence of primitives for successful normal data transfer. The sequence of primitives may remain incomplete if a N_RESET or N_DISCON primitive occurs.

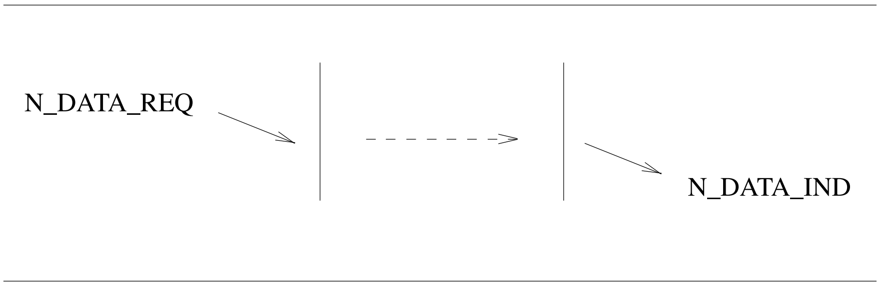

Figure 9. Sequence of Primitives: Data Transfer

The sequence of primitives in a successful confirmation of receipt is defined in the time sequence diagram as shown in Figure 10.

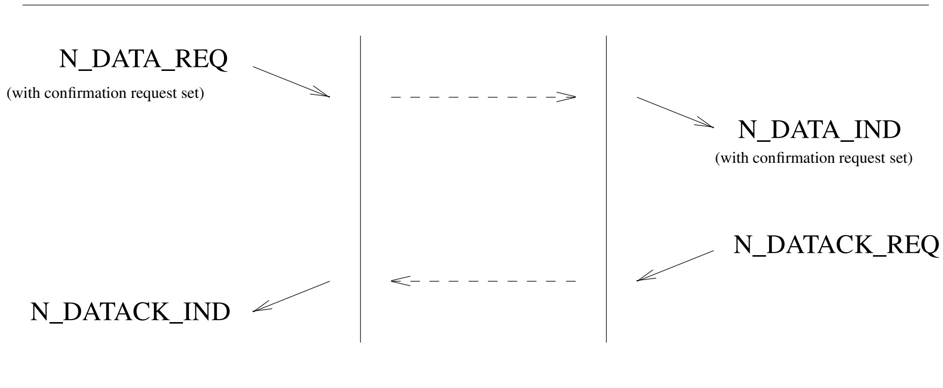

Figure 10. Sequence of Primitives: Successful Confirmation of Receipt

The sequence of primitives as shown above may remain incomplete if an N_RESET or an N_DISCON primitive occurs (see Table 3). A NS user must not issue an N_DATACK_REQ primitive if no N_DATA_IND with confirmation request set has been received, or if all such N_DATA_IND have been previously acknowledged. Following a reset procedure (N_RESET_REQ or N_RESET_IND), a NS user may not issue a N_DATACK_REQ to acknowledge an outstanding N_DATA_IND received before the reset procedure was signaled.

Note -- The withholding of confirmation of receipt by a NS user can have an effect on the attainable throughput on the NC.

The sequence of primitives for expedited data transfer is shown in the time sequence diagram in Figure 11. This sequence of primitives may remain incomplete if a

N_RESET or N_DISCON primitive is issued.

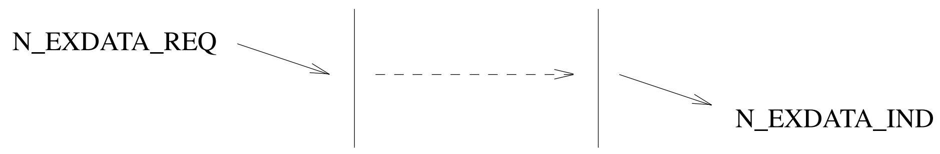

Figure 11. Sequence of Primitives: Expedited Data Transfer

3.2.3 Reset Operation Primitives

The reset service is used by the NS user to resynchronize the use of the NC, or by the NS provider to report detected loss of unrecoverable data.

The reset procedure involves the following interactions:

A. a N_RESET_REQ from the NS user, followed by a N_RESET_CON from the NS provider; or

B. a N_RESET_IND from the NS provider, followed by a N_RESET_RES from the NS user.

The complete sequence of primitives depends upon the origin/s of the reset action. The reset service may be:

1. invoked by one NS user, leading to interaction (A) with that NS user and interaction (B) with the peer NS user;
2. invoked by both NS users, leading to interaction (A) with both NS users;
3. invoked by the NS provider, leading to interaction (B) with both NS users;
4. invoked by one NS user and the NS provider, leading to interaction (A) with the originating NS user and (B) with the peer NS user.

The N_RESET_REQ acts as a synchronization mark in the flow of N_DATA, N_EXDATA, and N_DATACK primitives transmitted by the issuing NS user; the N_RESET_IND acts as a synchronization mark in the flow of N_DATA, N_EXDATA, and N_DATACK primitives received by the receiving NS user. Similarly, N_RESET_RES acts as a synchronization mark in the flow of N_DATA, N_EXDATA, and N_DATACK primitives transmitted by the responding NS user, while the N_RESET_CON acts as a synchronization mark in the flow of N_DATA, N_EXDATA, and N_DATACK primitives received by the NS user that originally issued the reset. The resynchronizing properties of the reset service are the following:

i. All N_DATA, N_EXDATA, and N_DATACK primitives issued before issuing the N_RESET_REQ/N_RESET_RES that have not been delivered to the other NS user before the N_RESET_IND/N_RESET_CON are issued by the NS provider, should be discarded by the NS provider.

ii. Any N_DATA, N_EXDATA, and N_DATACK primitives issued after the synchronization mark will not be delivered to the other NS user before the synchronization mark is received.

3.2.3.1 User Primitives for Reset Operations

- **N_RESET_REQ** : This primitive requests that the NS provider reset the network connection.
- **N_RESET_RES** : This primitive indicates to the NS provider that the NS user has accepted a reset indication.

3.2.3.2 Provider Primitives for Reset Operations

- **N_RESET_IND** : This primitive indicates to the NS user that the network connection has been reset.
- **N_RESET_CON** : This primitive indicates to the NS user that the reset request has been confirmed.

The sequence of primitives as shown in Figures 12, 13, 14, and 15 may remain incomplete if a N_DISCON primitive occurs.

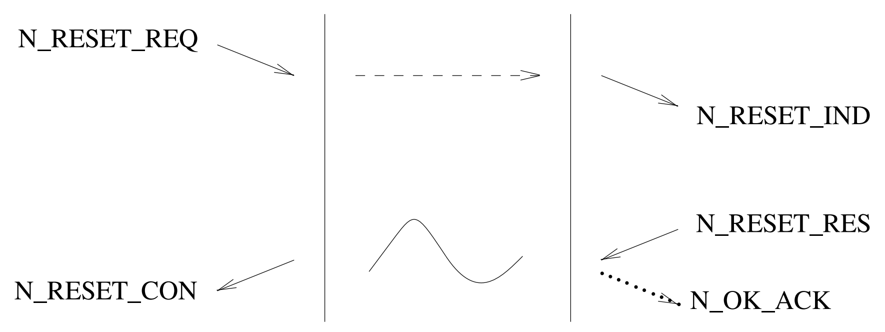

Figure 12. Sequence of Primitives: NS User Invoked Reset

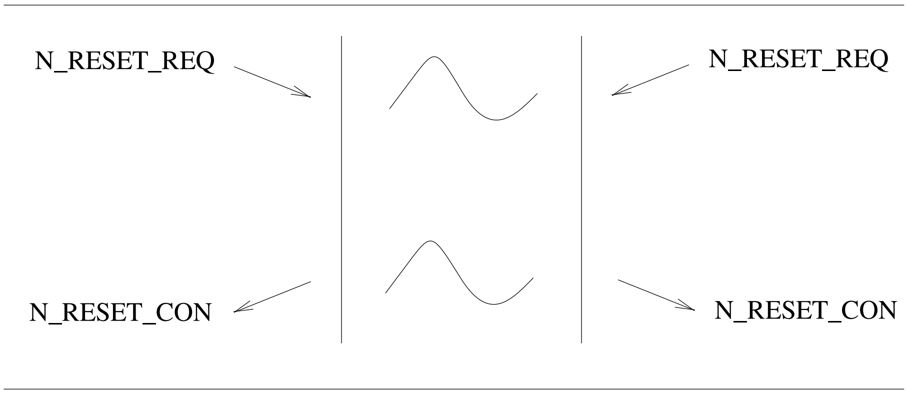

Figure 13. Sequence of Primitives: Simultaneous NS User Invoked Reset

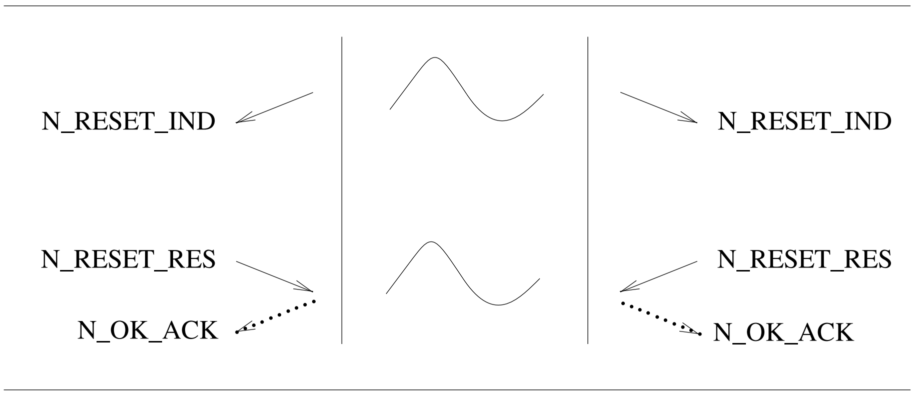

Figure 14. Sequence of Primitives: NS Provider Invoked Reset

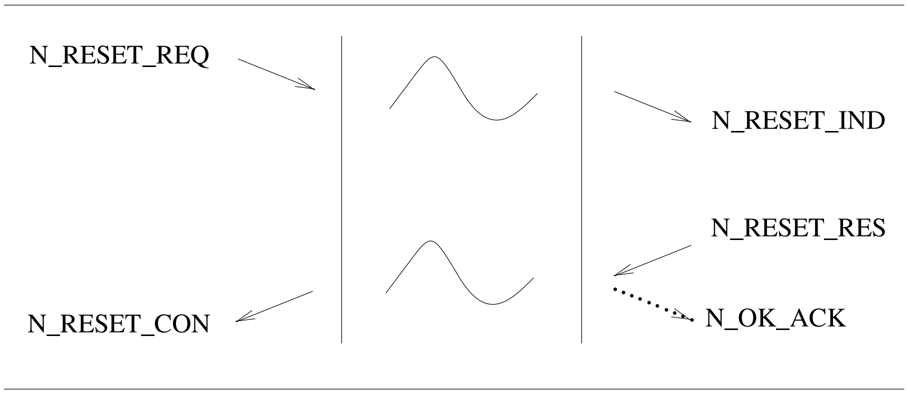

Figure 15. Sequence of Primitives: Simultaneous NS User & NS Provider Invoked Reset

3.2.4 Connection Termination Phase

The NC release procedure is initialized by the insertion of a disconnect object (associated with a N_DISCON_REQ) into the queue. As shown in Table 3, the disconnect procedure is destructive with respect to other objects in the queue, and eventually results in the emptying of queues and termination of the NC connection.

The sequence of primitives depends on the origin of the release action. The sequence may be:

1. invoked by one NS user, with a request from that NS user leading to an indication to the other;
2. invoked by both NS users, with a request from each of the NS users;
3. invoked by the NS provider, with an indication to each of the NS users;
4. invoked independently by one NS user and the NS provider, with a request from the originating NS user and an indication to the other.

3.2.4.1 User Primitives for Connection Termination

- **N_DISCON_REQ** : This primitive requests that the NS provider deny an outstanding request for a connection or disconnect an existing connection.

3.2.4.2 Provider Primitives for Connection Termination

- **N_DISCON_IND** : This primitive indicates to the NS user that either a request for connection has been denied or an existing connection has been terminated.

The sequence of primitives are shown in the time sequence diagrams in Figures 16, 17, 18, and 19.

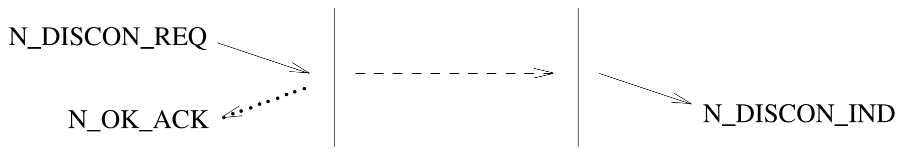

Figure 16. Sequence of Primitives: NS User Invoked Release

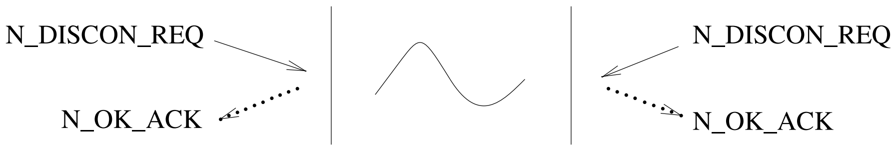

Figure 17. Sequence of Primitives: Simultaneous NS User Invoked Release

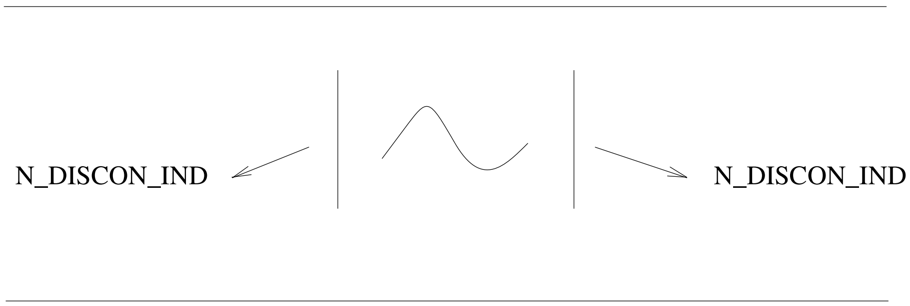

Figure 18. Sequence of Primitives: NS Provider Invoked Release

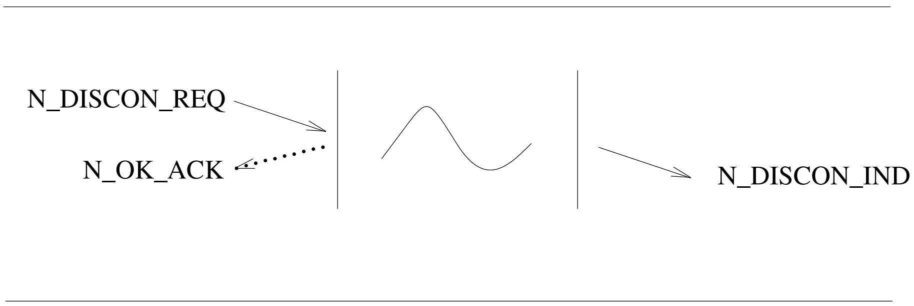

Figure 19. Sequence of Primitives: Simultaneous NS User & NS Provider Invoked Release

A NS user may reject an NC establishment attempt by issuing a N_DISCON_REQ. The originator parameter in the N_DISCON primitives will indicate NS user invoked release. The sequence of events is shown in Figure 20.

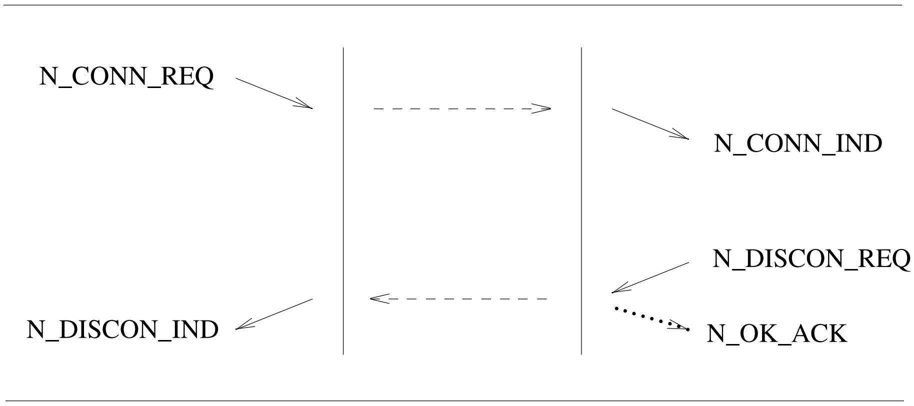

Figure 20. Sequence of Primitives: NS User Rejection of an NC Establishment Attempt

If the NS provider is unable to establish an NC, it indicates this to the requester by an N_DISCON_IND. The originator in this primitive indicates an NS provider invoked release. This is shown in Figure 21.

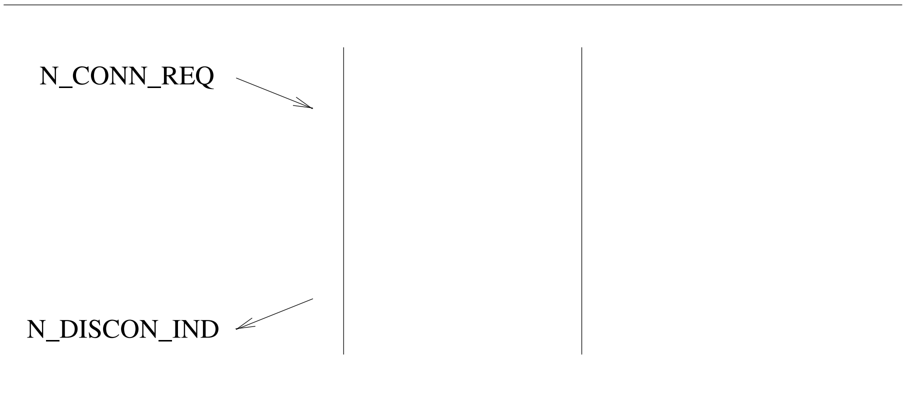

Figure 21. Sequence of Primitives: NS Provider Rejection of an NC Establishment Attempt

3.3 Connectionless Network Services Definition

The CLNS allows for the transfer of the NS user data in one or both directions simultaneously without establishing a network connection. A set of primitives are defined that carry user data and control information between the NS user and NS provider entities. The primitives are modeled as requests initiated by the NS user and indications initiated by the NS provider. Indications may be initiated by the NS provider independently from requests by the NS user.

The connectionless network service consists of one phase.

3.3.1 User Request Primitives

- **N_UNITDATA_REQ** : This primitive requests that the NS provider send the data unit to the specified destination.

3.3.2 Provider Response Primitives

- **N_UNITDATA_IND** : This primitive indicates to the NS user that a data unit has been received from the specified source address.

Figure 22 shows the sequence of primitives for the connectionless mode of data transfer.

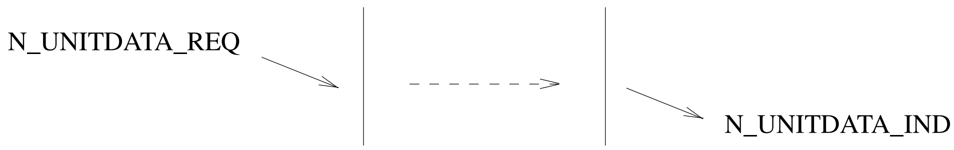

Figure 22. Sequence of Primitives: Connectionless Data Transfer

- **N_UDERROR_IND** : This primitive indicates to the NS user that the data unit with the specified destination address and QOS parameters produced an error. This primitive is specific to CLNS.

Figure 23 shows the sequence of primitives for the CLNS error management primitive.

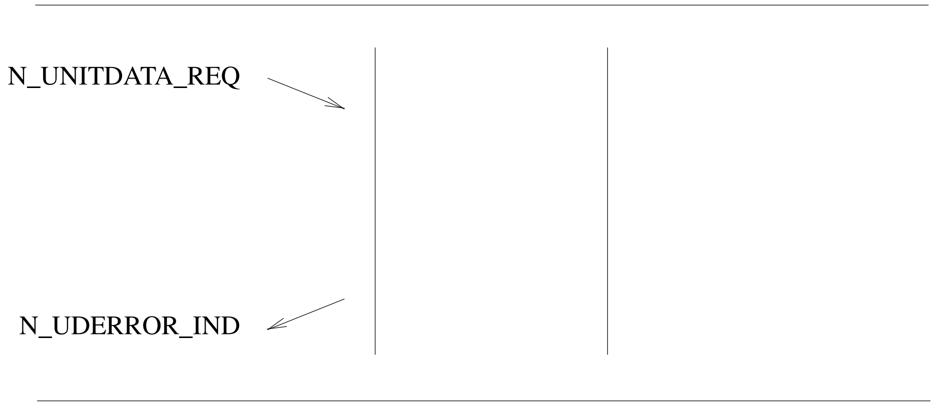

Figure 23. Sequence of Primitives: CLNS Error Indication Service

4. NPI Primitives

This section describes the format and parameters of the NPI primitives (Appendix A shows the mapping of the NPI primitives to the primitives defined in ISO 8348 and CCITT X.213). In addition, it discusses the states the primitive is valid in, the resulting state, and the acknowledgement that the primitive expects. (The state/event tables for these primitives are shown in Appendix B. The precedence tables for the NPI primitives are shown in Appendix C.) Rules for OSI conformance are described in Addendum 1 to this document.

Tables 5, 6, and 7 provide a summary of the NS primitives and their parameters.

SERVICE	PRIMITIVE	PARAMETERS
NC Establishment	N_CONN_REQ	(Called Address, Receipt Confirmation Selection, Expedited Data Selection, QOS Parameter Set, NS User-Data)
	N_CONN_IND	(Called Address, Calling Address, Receipt Confirmation Selection, Expedited Data Selection, QOS Parameter Set, NS User-Data)
	N_CONN_RES	(Responding Address, Receipt Confirmation Selection, Expedited Data Selection, QOS Parameter Set, NS User-Data)
	N_CONN_CON	(Responding Address, Receipt Confirmation Selection, Expedited Data Selection, QOS Parameter Set, NS User-Data)

TABLE 5. NC Establishment Network Service Primitives

SERVICE	PRIMITIVE	PARAMETERS
Normal Data Transfer	N_DATA_REQ	(NS User-Data, Confirmation Request)
	N_DATA_IND	(NS User-Data, Confirmation Request)
	N_UNITDATA_REQ	(Called Address, NS User-Data)
	N_UNITDATA_IND	(Called Address, Calling Address, NS User-Data)
Receipt Confirmation	N_DATACK_REQ	--
	N_DATACK_IND	--
Expedited Data Transfer	N_EXDATA_REQ	(NS User-Data)
	N_EXDATA_IND	(NS User-Data)
Reset	N_RESET_REQ	(Reason)
	N_RESET_IND	(Originator, Reason)
	N_RESET_RES	--
	N_RESET_CON	--

Note: -- No parameters specified with primitive

TABLE 6. Data Transfer Network Service Primitives

SERVICE	PRIMITIVE	PARAMETERS
NC Release	N_DISCON_REQ	(Reason, NS User-Data, Responding Address)
	N_DISCON_IND	(Originator, Reason, NS User-Data, Responding Address)

TABLE 7. NC Release Network Service Primitives

4.1 Management Primitives

These primitives apply both to CONS as well as CLNS.

4.1.1 Network Information Request

N_INFO_REQ

This primitive requests the NS provider to return the values of all supported protocol parameters (see under N_INFO_ACK), and also the current state of the NS provider (as defined in Appendix B). This primitive does not affect the state of the network provider and does not appear in the state tables.

Format

The format of the message is one M_PCPROTO message block and its structure is as follows:

```
typedef struct {
    ulong          PRIM_type;          /*always N_INFO_REQ */
} N_info_req_t;
```

Parameters

PRIM_type: Indicates the primitive type.

Valid States

This primitive is valid in any state where a local acknowledgement is not pending.

New State

The new state remains unchanged.

Acknowledgements

This primitive requires the NS provider to generate one of the following acknowledgements upon receipt of the primitive:

— **Successful**: Acknowledgement of the primitive via the N_INFO_ACK primitive.

— **Non-fatal_errors**: There are no errors associated with the issuance of this primitive.

4.1.2 Network Information Acknowledgement

N_INFO_ACK

This primitive indicates to the NS user any relevant protocol-dependent parameters.[1] It should be initiated in response to the N_INFO_REQ primitive described above.

Format

The format of this message is one M_PCPROTO message block and its structure is as follows:

1. In the future, this primitive will be modified such that it will allow the NPI to accept either sub-network point of attachment addresses or network addresses.

```
typedef struct {
    ulong       PRIM_type;          /*always N_INFO_ACK */
    ulong       NSDU_size;          /*maximum NSDU size */
    ulong       ENSDU_size;         /*maximum ENSDU size */
    ulong       CDATA_size;         /*connect data size */
    ulong       DDATA_size;         /*discon data size */
    ulong       ADDR_size;          /*address size */
    ulong       ADDR_length;        /*address length*/
    ulong       ADDR_offset;        /*address offset*/
    ulong       QOS_length;         /*length of default QOS values*/
    ulong       QOS_offset;         /*offset of default QOS values from the beginning of block*/
    ulong       QOS_range_length;   /*length of range of QOS values*/
    ulong       QOS_range_offset;   /*offset of range of QOS values from the beginning of block*/
    ulong       OPTIONS_flags;      /*bit masking for options supported*/
    ulong       NIDU_size;          /*network interface data unit size*/
    long        SERV_type;          /*service type*/
    ulong       CURRENT_state;      /*current state */
    ulong       PROVIDER_type;      /* type of provider */
    ulong       NODU_size;          /* optimal NSDU size */
    ulong       PROTOID_length;     /*length of bound protocol ids */
    ulong       PROTOID_offset;     /* offset of bound protocol ids */
    ulong       NPI_version;        /* version number of NPI that's supported */
} N_info_ack_t;

/*  Flags to indicate support of NS provider options  */

#define REC_CONF_OPT                0x00000001L
#define EX_DATA_OPT                 0x00000002L
#define DEFAULT_RC_SEL              0x00000004L


/*  Service types supported by the NS provider  */

#define N_CONS 1
#define N_CLNS 2


/* Valid provider types */
#define N_SNICFP                    1
#define N_SUBNET                    2
```

Parameters

The above fields have the following meaning:

PRIM_type: Indicates the primitive type.

NSDU_size: Specifies the maximum size (in octets) of a Network Service Data Unit (NSDU) supported by the NS provider.

ENSDU_size: Specifies the maximum size (in octets) of an Expedited Network Service Data Unit (ENSDU) supported by the NS provider.

CDATA_size: Specifies the maximum number of octets of data that may be associated with connection establishment primitives.

DDATA_size: Specifies the maximum number of octets of data that may be associated with the disconnect primitives.

ADDR_size: Specifies the maximum size (in decimal digits) of a network address.

ADDR_length: Specifies the length in bytes of the network address bound on the STREAM on which the N_INFO_REQ was issued (a network address is bound to a STREAM via a N_BIND_REQ).

ADDR_offset: Specifies the offset of the bound network address from the beginning of the M_PCPROTO message block (this field should be ignored if the ADDR_length field is zero).

QOS_length: in an addendum to this document. In the connection-mode environment, when this primitive is invoked before the NC is established on the stream, the values returned specify the the default values supported by the NS provider. When this primitive is invoked after a NC has been established on the stream, the values returned indicate the negotiated values for the QOS parameters. In the connection-less environment, these values represent the default or the selected QOS parameter values.

In case a QOS parameter is not supported by the NS Provider, a value of QOS_UNKNOWN will be returned. In the case where no QOS parameters are supported by the NS provider, this field will be zero.

QOS_offset: Indicates the offset of the QOS parameters from the beginning of the M_PCPROTO message block.

QOS_range_length: Indicates the length in bytes, of the available range of QOS parameters values supported by the NS provider. These ranges are used by the NS user to select QOS parameter values that are valid with the NS provider. QOS parameter values are selected, or the default values altered via the N_OPTMGMT_REQ primitive. In the connection-mode environment, the values for end-to-end QOS parameters may be specified with the

N_CONN primitives for negotiation.

If the NS provider does not support a certain QOS parameter, its value will be set to QOS_UNKNOWN. In the case where no QOS parameters are supported by the NS provider, the length of this field will be zero.

QOS_range_offset: Indicates the offset of the range of QOS parameter values from the beginning of the M_PCPROTO message block.

OPTIONS_flags: Defines flags that indicate whether the options described below are supported by the NS provider. The possible options are receipt confirmation, expedited data and default selection for use of receipt confirmation.

NIDU_size: This indicates the amount of user data that may be present in a N_DATA primitive. The NIDU_size should not be larger than the NSDU_size specification.

SERV_type: Specifies the service type supported by the NS provider. The possible values can be N_CONS, N_CLNS, (or both as indicated by using N_CONS|N_CLNS).

CURRENT_state: This indicates the current state of the NS provider.

PROVIDER_type: This indicates the type of NS provider. The possible values can be N_SNICFP or N_SUBNET. The value N_SNICFP indicates that the provider is the Subnetwork Independent Convergence Function/Protocol sub-layer of the network layer. The value N_SUBNET indicates that the provider is a subnetwork.

NODU_size: This specifies the optimal NSDU size (in octets) of an NSDU given the current routing information.

PROTOID_length: This specifies the length of the protocol ids that were bound using the N_BIND_REQ.

PROTOID_offset: This specifies the offset of the protocol ids that were bound using the N_BIND_REQ.

NPI_version: This indicates the current version of NPI that is supported.

Flags

REC_CONF_OPT: When set, it indicates that the NS provider supports receipt confirmation.

This flag is used only in the connection-mode environment.

EX_DATA_OPT: When set, it indicates that the NS provider supports expedited data transfer.

This flag is used only in the connection-mode environment.

DEFAULT_RC_SEL: When set, it indicates that the default selection is for the use of receipt confirmation for every N_DATA_REQ primitive (This parameter is applicable only when use of receipt confirmation is successfully negotiated via the N_CONN primitives).

This flag is used only in the connection-mode environment.

N_CONS: When set, it indicates that the NS provider supports connection-mode network services.

N_CLNS: When set, it indicates that the NS provider supports connection-less network services.

Valid States

This primitive is valid in any state in response to a N_INFO_REQ primitive.

New State

The state remains the same.

4.1.3 Bind Protocol Address Request

N_BIND_REQ

This primitive requests that the NS provider bind a NS user entity to a network address and negotiate the number of connect indications allowed to be outstanding by the NS provider for the specified NS user entity being bound.

Format

The format of the message is one M_PROTO message block and its structure is as follows:

```
typedef struct {
    ulong         PRIM_type;            /*always N_BIND_REQ */
    ulong         ADDR_length;          /*length of address */
    ulong         ADDR_offset;          /*offset of address */
    ulong         CONIND_number;        /*req # of conn-indications to be queued */
    ulong         BIND_flags;           /*flags associated with N_BIND_REQ*/
    ulong         PROTOID_length;       /*length of the protocol id*/
    ulong         PROTOID_offset;       /* offset of protocol id */
}  N_bind_req_t;


/* Flags associated with N_BIND_REQ */
#define DEFAULT_LISTENER            0x00000001L
#define TOKEN_REQUEST               0x00000002L
#define DEFAULT_DEST                0x00000004L
```

Parameters

PRIM_type: Is the primitive type.

ADDR_length: Is the length in bytes of the network address to be bound to the stream.

ADDR_offset: Is the offset from the beginning of the M_PROTO block where the network address begins.

CONIND_number: Is the requested number of connect indications allowed to be outstanding by the NS provider for the specified protocol address. (If the number of outstanding connect indications equals CONIND_number, the NS provider need not discard further incoming connect indications, but may choose to queue them internally until the number of outstanding connect indications drops below the CONIND_number.) Only one stream per network address is allowed to have a CONIND_number value greater than zero. This indicates to the network provider that this stream is the listener stream for the NS user. This stream will be used by the NS provider for connect indications for that network address.

If a stream is bound as a listener stream, it will not be able to initiate connect requests. If the NS user attempts to send an N_CONN_REQ primitive down this stream, an N_ERROR_ACK message will be sent to the NS user by the NS provider with an error value fo NACCESS.

This field should be ignored in CLNS.

PROTOID_length: Is the length in bytes of the protocol ids to be bound to the stream.

PROTOID_offset: Is the offset from the beginning of the M_PROTO block where the protocol id begins.

Flags

DEFAULT_LISTENER: When set, this flag indicates that this stream is the "default listener stream". This stream is used to pass connect indications for all incoming calls that contain protocol identifiers that are not bound to any other listener, or when a listener stream with CONIND_number value of greater than zero is not found. Also, the default listener will receive all incoming call indications that contain no user data.

Only one default listener stream is allowed per occurrence of NPI. An attempt to bind a default listener stream when one is already bound should result in an error (of type NBOUND).

The DEFAULT_LISTENER flag is ignored in CLNS.

TOKEN_REQUEST: When set, this flag indicates to the NS provider that the NS user has requested that a "token" be assigned to the stream (to be used in the NC response message), and the token value be returned to the NS user via the N_BIND_ACK primitive.

The token assigned by the NS provider can then be used by the NS user in a subsequent N_CONN_RES primitive to identify the stream on which the NC is to be established.

The TOKEN_REQUEST flag is ignored in CLNS.

DEFAULT_DEST: When set, this flag indicates that this stream is the "default destination stream." This stream will receive all packets destined for the NSAP specified in the bind request. If no

NSAP is indicated in the bind request, then this stream should receive all packets destined to an NSAP which is bound to no other stream.

Only one default destination stream per NSAP is allowed per occurance of NPI. An attempt to bind a default destination stream to an NSAP when one is already bound should result in an error of type NBOUND.

The DEFAULT_DEST flag is ignored in the CONS.

Valid States

This primitive is valid in state NS_UNBND (see Appendix B).

New State

The new state is NE_WACK_BREQ.

Acknowledgements

The NS provider will generate one of the following acknowledgements upon receipt of the N_BIND_REQ primitive:

— **Successful** : Correct acknowledgement of the primitive is indicated via the N_BIND_ACK primitive.

— **Non-fatal errors** : These errors will be indicated via the N_ERROR_ACK primitive. The applicable non-fatal errors are as follows:

NBADADDR: The network address was in an incorrect format or the address contained illegal information. It is not intended to indicate protocol errors.

NBOUND: The NS user attempted to bind a second stream to a network address with the CONIND_number set to a non-zero value, or attempted to bind a second stream with the DEFAULT_LISTENER flag value set to non-zero.

NNOADDR: The NS provider could not allocate an address.

NACCESS: The user did not have proper permissions for the use of the requested address.

NOUTSTATE: The primitive was issued from an invalid state.

NSYSERR: A system error has occurred and the UNIX system error is indicated in the primitive.

NNOPROTOID: Protocol identifier could not be allocated.

4.1.4 Bind Protocol Address Acknowledgement

N_BIND_ACK

This primitive indicates to the NS user that the specified network user entity has been bound to the requested network address and that the specified number of connect indications are allowed to be queued by the NS provider for the specified network address.

Format

The format of the message is one M_PCPROTO message block, and its structure is the following:

```
typedef struct {
    ulong        PRIM_type;          /*always N_BIND_ACK */
    ulong        ADDR_length;        /*address length */
    ulong        ADDR_offset;        /*offset of address */
    ulong        CONIND_number;      /*connection indications */
    ulong        TOKEN_value;        /*NC response token value*/
    ulong        PROTOID_length;     /*length of protocol id */
    ulong        PROTOID_offset;     /*offset from beg. of block */
} N_bind_ack_t;
```

Parameters

PRIM_type: Indicates the primitive type.

ADDR_length: Is the length of the network address that was bound.

ADDR_offset: Is the offset from the beginning of the M_PCPROTO block where the network address begins.

CONIND_number: Is the accepted number of connect indications allowed to be outstanding by the NS provider for the specified network address. If its value is zero, this stream cannot accept N_CONN_IND messages. If its value is greater than zero, then the NS user can accept N_CONN_IND messages up to the value specified in this parameter before having to respond with a N_CONN_RES or a N_DISCON_REQ message.

This field should be ignored for CLNS.

TOKEN_value: Conveys the value of the "token" assigned to this stream that can be used by the NS user in a N_CONN_RES primitive to accept a NC on this stream. It is a non-zero value, and is unique to all streams bound to the NS provider.

This field should be ignored for CLNS.

PROTOID_length: Conveys the length of the protocol ids that were bound.

PROTOID_offset: Conveys the offset of the protocol ids that were bound.

The proper alignment of the address in the M_PCPROTO message block is not guaranteed.

Bind Rules :

The following rules apply to the binding of the specified network address to the stream:

— If the ADDR_length field in the N_BIND_REQ primitive is zero, then the NS provider is to assign a network address to the user.

— The NS provider is to bind the network address as specified in the N_BIND_REQ primitive. If the NS provider cannot bind the specified address, it may assign another network address to the user. It is the network user's responsibility to check the network address returned in the N_BIND_ACK primitive to see if it is the same as the one requested.

The following rules apply to negotiating CONIND_number argument:

— The CONIND_number in the N_BIND_ACK primitive must be less than or equal to the corresponding requested number as indicated in the N_BIND_REQ primitive.

— Only one stream that is bound to the indicated network address may have a negotiated accepted number of maximum connect requests greater than zero. If a N_BIND_REQ primitive specifies a value greater than zero, but another stream has already bound itself to the given network address with a value greater than zero, the NS provider should assign another protocol address to the user.

— If a stream with CONIND_number greater than zero is used to accept a connection, the stream will be found busy during the duration of that connection and no other streams may be bound to that network address with a CONIND_number greater than zero. This will prevent more than one stream bound to the identical network address from accepting connect indications.

— A stream requesting a CONIND_number of zero should always be legal. This indicates to the NS provider that the stream is to be used to request connections only.

— A stream with a negotiated CONIND_number greater than zero may generate connect requests or accept connect indications.

If the above rules result in an error condition, then the NS provider must issue an N_ERROR_ACK primitive to the NS user specifying the error as defined in the description of the N_BIND_REQ primitive.

Valid States

This primitive is in response to a N_BIND_REQ primitive and is valid in the state NS_WACK_BREQ.

New State

The new state is NS_IDLE.

4.1.5 Unbind Protocol Address Request

N_UNBIND_REQ

This primitive requests that the NS provider unbind the NS user entity that was previously bound to the network address.

Format

The format of the message is one M_PROTO block, and its structure is as follows:

```
typedef struct {
    ulong          PRIM_type;          /*always N_UNBIND_REQ */
} N_unbind_req_t;
```

Parameters

PRIM_type: Indicates the primitive type.

Valid States

This primitive is valid in the NS_IDLE state.

New State

The new state is NS_WACK_UREQ.

Acknowledgements

This primitive requires the NS provider to generate the following acknowledgements upon receipt of the primitive:

— **Successful** : Correct acknowledgement of the primitive is indicated via the N_OK_ACK primitive.

— **Unsuccessful (Non-fatal errors)** : These errors will be indicated via the N_ERROR_ACK primitive. The applicable non-fatal errors are as follows:

NOUTSTATE: The primitive was issued from an invalid state.

NSYSERR: A system error has occurred and the UNIX system error is indicated in the primitive.

4.1.6 Network Options Management Request

N_OPTMGMT_REQ

This primitive allows the NS user to manage the QOS parameter values associated with the stream.

Format

The format of the message is one M_PROTO message block, and its structure is as follows:

```
typedef struct {
    ulong       PRIM_type;         /*always N_OPTMGMT_REQ*/
    ulong       QOS_length;        /*length of QOS values */
    ulong       QOS_offset;        /*offset of QOS values */
    ulong       OPTMGMT_flags;     /*default receipt conf. selection*/
} N_optmgmt_req_t;
```

PRIM_type: Indicates the primitive type.

QOS_length: Indicates the length of the default values of the QOS parameters as selected by the NS user. In the connection-mode environment these values will be used in subsequent N_CONN_REQ primitives on the stream that do not specify values for these QOS parameters. In the connection-less environment, these values represent the selected QOS values that would apply to each unitdata transmission. If the NS user cannot determine the value of a QOS parameter, its value should be set to QOS_UNKNOWN. If the NS user does not specify any QOS parameter values, the length of this field should be set to zero.

QOS_offset: Indicates the offset of the QOS parameters from the beginning of the M_PROTO message block.

Flags

DEFAULT_RC_SEL: When set, it indicates to the NS provider that the NS user's default selection is for the use of receipt confirmation with every N_DATA_REQ message (applicable only when its use is successfully negotiated via the N_CONN primitives). This default indication is used only when the M_PROTO message block is not present in the N_DATA_REQ primitive.

This flag should be ignored in the connection-less environment.

Valid States

This primitive is valid in the NS_IDLE state.

New State

The new state is NS_WACK_OPTREQ.

Acknowledgements

The N_OPTMGMT_REQ primitive requires the NS provider to generate one of the following acknowledgements upon receipt of the primitive:

— **Successful** : Acknowledgement is via the N_OK_ACK primitive. At successful completion, the resulting state is NS_IDLE.

— **Non-fatal errors** : These errors are indicated in the N_ERROR_ACK primitive. The resulting state remains unchanged. The applicable non-fatal errors are defined as follows:

NOUTSTATE:	The primitive was issued from an invalid state.
NBADQOSPARAM:	The QOS parameter values specified are outside the range supported by the NS provider.
NBADQOSTYPE:	The QOS structure type is not supported by the NS provider.
NSYSERR:	A system error has occurred and the UNIX system error is indicated in the primitive.

4.1.7 Error Acknowledgement

N_ERROR_ACK

This primitive indicates to the NS user that a non-fatal error has occurred in the last network-user-originated primitive. This may only be initiated as an acknowledgement for those primitives that require one. It also indicates to the user that no action was taken on the primitive that caused the error.

Format

The format of the message is one M_PCPROTO message block, and its structure is as follows:

```
typedef struct {
    ulong       PRIM_type;          /*always N_ERROR_ACK */
    ulong       ERROR_prim;         /*primitive in error */
    ulong       NPI_error;          /*NPI error code */
    ulong       UNIX_error;         /*UNIX system error code */
} N_error_ack_t;
```

Parameters

PRIM_type:	Identifies the primitive type
ERROR_prim:	Identifies the primitive type that caused the error.
NPI_error:	Contains the Network Provider Interface error code.
UNIX_error:	Contains the UNIX system error code. This may only be non-zero if the NPI_error is equal to NSYSERR.

Valid Error Codes

The following error codes are allowed to be returned:

NBADADDR:	The network address as specified in the primitive was in an incorrect format, or the address contained illegal information.
NBADOPT:	The options values as specified in the primitive were in an incorrect format, or they contained illegal information.
NBADQOSPARAM:	The QOS values specified are outside the range supported by the NS provider. illegal.
NBADQOSTYPE:	The QOS structure type is not supported by the NS provider.
NBADTOKEN:	Token used is not associated with an open stream.
NNOADDR:	The NS provider could not allocate an address.
NACCESS:	The user did not have proper permissions.
NOUTSTATE:	The primitive was issued from an invalid state.

NBADSEQ:	The sequence number specified in the primitive was incorrect or illegal.
NBADFLAG:	The flags specified in the primitive were incorrect or illegal.
NBADDATA:	The amount of user data specified was outside the range supported by the NS provider.
NSYSERR:	A system error has occurred and the UNIX system error is indicated in the primitive.
NNOTSUPPORT:	Specified primitive type is not known to the NS provider.

Valid States

This primitive is valid in all states that have a pending acknowledgement or confirmation.

New State

The new state is the same as the one from which the acknowledged request or response was issued.

4.1.8 Successful Receipt Acknowledgement

N_OK_ACK

This primitive indicates to the NS user that the previous network- user-originated primitive was received successfully by the network provider. It does not indicate to the NS user any network protocol action taken due to the issuance of the last primitive. The N_OK_ACK primitive may only be initiated as an acknowledgement for those user-originated primitives that have no other means of confirmation.

Format

The format of the message is one M_PCPROTO message block, and its structure is as follows:

```
typedef struct {
    ulong        PRIM_type;          /*always N_OK_ACK */
    ulong        CORRECT_prim;       /*primitive being acknowledged */
} N_ok_ack_t;
```

Parameters

— PRIM_type : Identifies the primitive.

— CORRECT_prim: Identifies the successfully received primitive type.

Valid States

This primitive is issued in states NS_WACK_UREQ, NS_WACK_OPTREQ, NS_WACK_RRES, NS_WACK_CRES, NS_WACK_DREQ6, NS_WACK_DREQ7, NS_WACK_DREQ9, NS_WACK_DREQ10, and NS_WACK_DREQ11, in response to N_UNBIND_REQ, N_RESET_RES, N_CONN_RES, and N_DISCON_REQ primitives.

New State

The resulting state depends on the current state (see Appendix B, Tables B-7 and B-8.)

4.2 CONS: Primitive Format and Rules

This section describes the format of the CONS primitives and the rules associated with these primitives. The default values of the QOS parameters associated with a NC may be selected via the N_OPTMGMT_REQ primitive.

4.2.1 Connection Establishment Phase

The following network service primitives pertain to the establishment of an NC, provided the NS users exist, and are known to the NS provider.

4.2.1.1 Network Connection Request

N_CONN_REQ

This primitive requests that the NS provider make a network connection to the specified destination.

Format

The format of the message is one M_PROTO message block followed by one or more M_DATA blocks for the NS user data transfer. The specification of the NS user data is optional. The NS user can send any integral number of octets of data within the range supported by the NS provider (see N_INFO_ACK). If the user does not specify QOS parameter values, the default values (specified via N_OPTMGMT_REQ) are used by the NS provider.

The structure of the M_PROTO message block is as follows:

```
typedef struct {
    ulong       PRIM_type;          /*always N_CONN_REQ */
    ulong       DEST_length;        /*destination address length*/
    ulong       DEST_offset;        /*destination address offset*/
    ulong       CONN_flags;         /*bit masking for options flags  */
    ulong       QOS_length;         /*QOS parameters' length*/
    ulong       QOS_offset;         /*QOS parameters' offset*/
} N_conn_req_t;

/*  Flags to indicate if options are requested  */

#define REC_CONF_OPT                0x00000001L
#define EX_DATA_OPT                 0x00000002L
```

Parameters

PRIM_type: Indicates the primitive type.

DEST_length: Indicates the length of the destination address parameter that conveys an address identifying the NS user to which the NC is to be established. This field will accommodate variable length addresses within a range supported by the NS provider.

DEST_offset: Is the offset of the destination address from the beginning of the M_PROTO message block.

QOS_length: Indicates the length of the QOS parameters values that apply to the NC being requested. If the NS user cannot determine the value of a QOS parameter, its value should be set to QOS_UNKNOWN. If the NS user does not specify any QOS parameter values, the length of this field should be set to zero.

QOS_offset: Indicates the offset of the QOS parameters from the beginning of the M_PROTO message block.

Flags

REC_CONF_OPT: The receipt confirmation selection parameter indicates the use/availability of the receipt confirmation service on the NC. The receipt confirmation service must be supported by the NS provider to be used on the NC.

EX_DATA_OPT: Indicates the use of the expedited data transfer service on the NC. The expedited data transfer service must be provided by the NS provider for it to be used on the NC.

Valid States

This primitive is valid in state NS_IDLE.

New State

The new state is NS_WCON_CREQ.

Acknowledgements

The following acknowledgements are valid for this primitive:

— **Successful NC Establishment**: This is indicated via the N_CONN_CON primitive. This results in the data transfer state.

— **Unsuccessful NC Establishment**: This is indicated via the N_DISCON_IND primitive. For example, a connection may be rejected because either the called NS user cannot be reached, or the NS provider and/or the called NS user did not agree with the specified QOS. This results in the idle state.

— **Non-fatal errors**: These are indicated via the N_ERROR_ACK primitive. The applicable non-fatal errors are defined as follows:

NACCESS: The user did not have proper permissions for the use of the requested address or options.

NBADQOSPARAM: The QOS parameter values specified are outside the range supported by the NS provider.

NBADQOSTYPE:	The QOS structure type is not supported by the NS provider.
NBADADDR:	The network address was in an incorrect format or contained illegal information. It is not intended to indicate NC errors, such as an unreachable destination. These errors types are indicated via the N_DISCON_IND primitive.
NBADOPT:	The options were in an incorrect format, or they contained illegal information.
NOUTSTATE:	The primitive was issued from an invalid state.
NBADDATA:	The amount of user data specified was outside the range supported by the NS provider.
NSYSERR:	A system error has occurred and the UNIX system error is indicated in the primitive.

4.2.1.2 Network Connection Indication

N_CONN_IND

This primitive indicates to the destination NS user that a network connect request has been made by the user at the specified source address.

Format

The format of this message is one M_PROTO message block followed by one or more M_DATA blocks for NS user data. The specification of NS user data is optional. The NS user can send any integral number of octets of data within the range supported by the NS provider. The NS user data will only be present if the corresponding N_CONN_REQ had NS user data parameter specified, and their data will be identical.

The structure of the M_PROTO message block is as follows:

```
typedef struct {
    ulong      PRIM_type;        /*always N_CONN_IND */
    ulong      DEST_length;      /*destination address length*/
    ulong      DEST_offset;      /*destination address offset*/
    ulong      SRC_length;       /*source address length*/
    ulong      SRC_offset;       /*source address offset*/
    ulong      SEQ_number;       /*sequence number*/
    ulong      CONN_flags;       /*bit masking for options flags*/
    ulong      QOS_length;       /*QOS parameters' length*/
    ulong      QOS_offset;       /*QOS parameters' offset*/
} N_conn_ind_t;
```

Parameters

PRIM_type: Indicates the primitive type.

DEST_length: Indicates the length of the destination address parameter that conveys an address identifying the NS user to which the NC is to be established.

DEST_offset: Is the offset of the destination address from the beginning of the M_PROTO message block.

SRC_length: The source address parameter conveys the network address of the NS user from which the NC has been requested. The semantics of the value in the N_CONN_IND primitive is identical to the value associated with the stream on which the N_CONN_REQ was issued.

SRC_offset: Is the offset of the destination address from the beginning of the M_PROTO message block.

SEQ_number: Identifies the sequence number that can be used by the NS user to associate this message with the N_CONN_RES or

N_DISCON_REQ primitive that is to follow. This value must be unique among the outstanding N_CONN_IND messages.

The use of this field allows the NS user to issue the N_CONN_RES or the N_DISCON_REQ messages in any order.

QOS_length: Indicates the length of the QOS parameters values that are negotiated during NC establishment. If the destination NS user does not agree to the range of QOS values specified by the source NS user in the N_CONN_REQ primitive, it will reject the NC establishment by invoking a N_DISCON_REQ primitive (the originator parameter in the N_DISCON_REQ primitive will indicate NS user initiated release). If the NS user does not support or cannot determine the value of a QOS parameter, its value will be set to QOS_UNKNOWN. If the NS user does not specify any QOS parameter values, the length of this field should be set to zero.

QOS_offset: Indicates the offset of the QOS parameters from the beginning of the M_PROTO message block.

Flags

REC_CONF_OPT: The receipt confirmation selection parameter indicates the use/availability of the receipt confirmation service on the NC. The receipt confirmation service must be provided in the network service to be used on the NC.

EX_DATA_OPT: The expedited data selection parameter indicates the use/ availability of the expedited data transfer service on the NC. The expedited data transfer service must be provided by the NS provider for it to be used on the NC.

Valid States

This primitive is valid in the states NS_IDLE and NS_WRES_CIND.

New State

In both cases the resulting state is NS_WRES_CIND (the number of connect indications waiting for user response is incremented by one).

4.2.1.3 Network Connection Response

N_CONN_RES

This primitive allows the destination NS user to request that the network provider accept a previous connect request.

Format

The format of this message is one M_PROTO message block followed by one or more M_DATA blocks (for NS user data). The specification of the NS user data is optional. The NS user can send any integral number of octets of data within the range supported by the NS provider.

The structure of the M_PROTO block is as follows:

```
typedef struct {
    ulong       PRIM_type;          /*always N_CONN_RES */
    ulong       TOKEN_value;        /*NC response token value*/
    ulong       RES_length;         /*responding address length*/
    ulong       RES_offset;         /*responding address offset*/
    ulong       SEQ_number;         /*sequence number*/
    ulong       CONN_flags;         /*bit masking for options flags*/
    ulong       QOS_length;         /*QOS parameters' length*/
    ulong       QOS_offset;         /*QOS parameters' offset*/
} N_conn_res_t;
```

Parameters

PRIM_type: Indicates the primitive type.

TOKEN_value: Is used to identify the stream that the NS user wants to establish the NC on. (Its value is determined by the NS user by issuing a N_BIND_REQ primitive with the TOKEN_REQUEST flag set. The token value is returned in the N_BIND_ACK). The value of this field should be non-zero when the NS user wants to establish the NC on a stream other than the stream on which the N_CONN_IND arrived. If the NS user wants to establish a NC on the same stream that the N_CONN_IND arrived on, then the value of this field should be zero.

RES_length: Indicates the length of the responding address parameter that conveys the network address of the NS user to which the NC has been established. Under certain circumstances, such as call redirection, generic addressing, etc., the value of this parameter may be different from the destination address parameter specification in the corresponding N_CONN_REQ.

RES_offset: Indicates the offset of the responding address from the beginning of the M_PROTO message block.

SEQ_number: Indicates the sequence number of the N_CONN_RES message. It is used by the NS provider to associate the N_CONN_RES message with an outstanding N_CONN_IND message. An invalid sequence number should result in error with the message type NBADSEQ.

QOS_length: Indicates the length of the QOS parameters values that are negotiated during NC establishment. If the NS user does not agree to the QOS values, it will reject the NC establishment by invoking a N_DISCON_REQ primitive (the originator parameter in the N_DISCON_REQ primitive will indicate NS user invoked release). If the NS user cannot determine the value of a QOS parameter, its value should be set to QOS_UNKNOWN. If the NS user does not specify any QOS parameter values, the length of this field should be set to zero.

QOS_offset: Indicates the offset of the QOS parameters from the beginning of the M_PROTO message block.

Flags

REC_CONF_OPT: The receipt confirmation selection parameter indicates the use/availability of the receipt confirmation service on the NC. The receipt confirmation service must be provided in the network service to be used on the NC.

EX_DATA_OPT: The expedited data selection parameter indicates the use/ availability of the expedited data transfer service on the NC. The expedited data transfer service must be provided by the NS provider for it to be used on the NC.

Valid States

This primitive is valid in state NS_WRES_CIND.

New State

The new state is NS_WACK_CRES.

Acknowledgements

The NS provider should generate one of the following acknowledgements upon receipt of this primitive:

— **Successful**: Successful completion is indicated via the N_OK_ACK primitive.

— **Unsuccessful (Non-fatal errors)**: Errors are indicated via the N_ERROR_ACK primitive. The applicable non-fatal errors are defined as follows:

NBADOPT:	The options were in an incorrect format, or they contained illegal information.
NBADQOSPARAM:	The QOS parameter values specified are outside the range supported by the NS provider.
NBADQOSTYPE:	The QOS structure type is not supported by the NS provider.
NBADTOKEN:	The token specified is not associated with an open stream.
NACCESS:	The user did not have proper permissions for the use of the options of the options or response id.
NOUTSTATE:	The primitive was issued from an invalid state.
NBADDATA:	The amount of user data specified was outside the range supported by the NS provider.
NBADSEQ:	The sequence number specified in the primitive was incorrect or illegal.
NSYSERR:	A system error has occurred and the UNIX system error is indicated in the primitive.

4.2.1.4 Network Connection Confirm

N_CONN_CON

This primitive indicates to the source NS user that the network connect request has been confirmed on the specified responding address.

Format

The format of this message is one M_PROTO message block followed by one or more M_DATA blocks (for NS user data). The specification of the NS user data is optional. The NS user can send any integral number of octets of NS user data within a range supported by the NS provider (see N_INFO_ACK). The NS user data will only be present if the corresponding N_CONN_RES had NS user data specified with it, and their data will always be identical.

The structure of the M_PROTO block is as follows:

```
typedef struct {
    ulong       PRIM_type;       /*always N_CONN_CON*/
    ulong       RES_length;      /*responding address length*/
    ulong       RES_offset;      /*responding address offset*/
    ulong       CONN_flags;      /*bit masking for options flags*/
    ulong       QOS_length;      /*QOS parameters' length*/
    ulong       QOS_offset;      /*QOS parameters' offset*/
} N_conn_con_t;
```

Parameters

PRIM_type: Indicates the primitive type.

RES_length: Indicates the length of the responding address parameter that conveys the network address of the NS user entity to which the NC has been established. The semantics of the values in the N_CONN_CON is identical to the values in N_CONN_RES. Under certain circumstances, such as call redirection, generic addressing, etc., the value of this parameter may be different from the destination address parameter specification in the corresponding N_CONN_REQ.

RES_offset: Indicates the offset of the responding address from the beginning of the M_PROTO message block.

QOS_length: Indicates the length of the QOS parameters values selected by the responding NS user. If the NS provider does not support or cannot determine the selected value of a QOS parameter, its value will be set to QOS_UNKNOWN. If the NS provider does not specify any QOS parameter values, the length of this field should be set to zero.

QOS_offset: Indicates the offset of the QOS parameters from the beginning of the M_PROTO message block.

Flags

REC_CONF_OPT: The receipt confirmation selection parameter indicates the use/availability of the receipt confirmation service on the NC. The receipt confirmation service must be provided in the network service to be used on the NC.

EX_DATA_OPT: The expedited data selection parameter indicates the use/ availability of the expedited data transfer service on the NC. The expedited data transfer service must be provided by the NS provider for it to be used on the NC.

Valid States

This primitive is valid in state NS_WCON_CREQ.

New State

The new state is NS_DATA_XFER.

4.2.2 Normal Data Transfer Phase

The data transfer service primitives provide for an exchange of NS user data known as NSDUs, in either direction or in both directions simultaneously on a NC. The network service preserves both the sequence and the boundaries of the NSDUs (when the NS provider supports NSDUs).

4.2.2.1 Normal Data Transfer Request

N_DATA_REQ

This user-originated primitive indicates to the NS provider that this message contains NS user data. It allows the transfer of NS_user_data between NS users, without modification by the NS provider.

The NS user must send any integral number of octets of data greater than zero. In a case where the size of the NSDU exceeds the NIDU (as specified by the size of the NIDU_size parameter of the N_INFO_ACK primitive), the NSDU may be broken up into more than one NIDU. When an NSDU is broken up into more than one NIDU, the N_MORE_DATA_FLAG will be set on each NIDU except the last one. The RC_flag may only be set on the last NIDU.

Format

The format of the message is one or more M_DATA blocks. Use of a M_PROTO message block is optional. The M_PROTO message block is used for two reasons:

i. to indicate that the NSDU is broken into more than one NIDUs, and that the data carried in the following M_DATA message block constitutes one NIDU;

ii. to indicate whether receipt confirmation is desired for the NSDU.

Guidelines for use of M_PROTO:

The following guidelines must be followed with respect to the use of the M_PROTO message block:

1. *The M_PROTO message block need not be present when the NSDU size is less than or equal to the NIDU size and one of the following is true:*

   — receipt confirmation has been negotiated for non-use (via the N_CONN primitives); or

   — receipt confirmation has been successfully negotiated for use or non-use and the default selection as specified via the N_OPTMGMT primitive is to be used.

2. *The M_PROTO message block must be present when:*

   — the NSDU size is greater than the NIDU size;

   — receipt confirmation has been successfully negotiated for use and the default selection as specified via N_OPTMGMT_REQ primitive needs to be overridden.

The structure of the M_PROTO message block, if present, is as follows:

```
typedef struct {
    ulong          PRIM_type;          /*always N_DATA_REQ*/
    ulong          DATA_xfer_flags;    /*bit masking for data xfer flags*/
} N_data_req_t;


/*  Data Transfer Flags  */

#define N_MORE_DATA_FLAG          0x00000001L
#define N_RC_FLAG                 0x00000002L
```

Parameters

PRIM_type: Indicates the primitive type.

Flags

N_MORE_DATA_FLAG: When set, the MORE_DATA_FLAG indicates that the next N_DATA_REQ message (NIDU) is also part of this NSDU.

N_RC_FLAG: By setting this flag on the N_DATA_REQ, the originating NS user can request confirmation of receipt of the N_DATA primitive. The receipt is provided by the N_DATACK primitives. The parameter may only be present if use of Receipt Confirmation was agreed by both NS users and the NS provider during NC establishment.

Valid States

This primitive is valid in the NS_DATA_XFER state.

New State

The resulting state remains the same (NS_DATA_XFER).

Acknowledgements

This primitive does not require any acknowledgements, although it may generate a fatal error. This is indicated to the NS user via a M_ERROR STREAMS message type (specifying an errno value of EPROTO) which results in the failure of all system calls on that stream. The applicable errors are defined as follows:

EPROTO: This indicates one of the following unrecoverable protocol conditions:

— The network interface was found to be in an incorrect state.

— The amount of NS user data associated with the primitive is outside the range supported by the NS provider (as specified by the NIDU_size parameter of N_INFO_ACK primitive).

— The options requested are either not supported by the NS provider or its use not specified with the N_CONN_REQ primitive.

— The M_PROTO message block was not followed by one or more M_DATA message blocks.

— The amount of NS user data associated with the current NSDU is outside the range supported by the NS provider (as specified by the NSDU_size parameter if the N_INFO_ACK primitive.)

— The N_RC_FLAG and N_MORE_DATA_FLAG were both set in the primitive, or the flags field contained an unknown value.

NOTE: If the interface is in the NS_IDLE or NS_WRES_RIND states when the provider receives the N_DATA_REQ primitive, then the NS provider should discard the request without generating a fatal error.

4.2.2.2 Normal Data Transfer Indication

N_DATA_IND

This network-provider-originated primitive indicates to the NS user that this message contains NS user data. As in the N_DATA_REQ primitive, the NSDU can be segmented into more than one NIDUs. The NIDUs are associated with the NSDU by using the MORE_DATA_FLAG. The RC_FLAG is allowed to be set only on the last NIDU.

Format

The format of the message is one or more M_DATA message blocks. The value of the NS user data field is always the same as that supplied in the corresponding N_DATA_REQ primitive at the peer service access point.

Use of M_PROTO message blocks is optional (see guidelines under N_DATA_REQ).

The structure of the M_PROTO message block, if present, is as follows:

```
typedef struct {
    ulong       PRIM_type;          /*always N_DATA_IND*/
    ulong       DATA_xfer_flags;    /*bit masking for data xfer flags*/
} N_data_ind_t;


/*  Data Transfer Flags  */

#define N_MORE_DATA_FLAG        0x00000001L
#define N_RC_FLAG               0x00000002L
```

Parameters

PRIM_type: Indicates the primitive type.

Flags

MORE_DATA_FLAG: When set, indicates that the next N_DATA_IND message (NIDU) is part of this NSDU.

RC_FLAG: The value of the parameter may indicate either that confirmation is requested or that it is not requested. The parameter is allowed to be set only if use of Receipt Confirmation was agreed to between both the NS users and the NS provider during NC establishment. The value of this parameter is always identical to that supplied in the corresponding N_DATA_REQ primitive.

Valid States

This primitive is valid in state NS_DATA_XFER.

New State

The resulting state remains the same (NS_DATA_XFER).

4.2.3 Receipt Confirmation Service Primitives

The receipt confirmation service is requested by the confirmation request parameter on the N_DATA_REQ primitive. For each and every NSDU with the confirmation request parameter set, the receiving NS user should return an N_DATACK_REQ primitive. Such acknowledgements should be issued in the same sequence as the corresponding N_DATA_IND primitives are received, and are to be conveyed by the NS provider in such a way so as to preserve them distinct from any previous or subsequent acknowledgements. The NS user may thus correlate them with the original requests by counting. When an NSDU has been segmented into more than one NIDUs, only the last NIDU is allowed to request receipt confirmation.

N_DATACK_REQ primitives will not be subject to the flow control affecting N_DATA_REQ primitives at the same NC endpoint. N_DATACK_IND primitives will not be subject to the flow control affecting N_DATA_IND primitives at the same NC endpoint.

The use of the receipt confirmation service must be agreed to by the two NS users of the NC and the NS provider during the NC establishment by using the RC_selection parameter on the N_CONN primitives.

4.2.3.1 Data Acknowledgement Request

N_DATACK_REQ

This is a user-originated primitive that requests that the network provider acknowledge the N_DATA_IND that had previously been received with the receipt confirmation parameter set.

Format

The format of the message is one M_PROTO message block and its structure is as follows:

```
typedef struct {
   ulong          PRIM_type;          /*always N_DATACK_REQ */
} N_datack_req_t;
```

Parameters

PRIM_type: Indicates the primitive type.

Valid States

This primitive is valid in state NS_DATA_XFER.

New State

The resulting state remains the same (NS_DATA_XFER).

Acknowledgements

This primitive does not require any acknowledgements, although it may generate a fatal (unrecoverable) error. This is indicated via an M_ERROR STREAMS message type (issued to the NS user specifying the errno value of EPROTO), which results in the failure of all system calls on that stream. The allowable errors are as follows:

EPROTO: This indicates the following unrecoverable protocol condition:

— The network interface was found to be in an incorrect state.

NOTE: If the interface is in the NS_IDLE state when the provider receives the N_DATACK_REQ primitive, then the NS provider should discard the request without generating a fatal error.

If the NS provider had no knowledge of a previous N_DATA_IND with the receipt confirmation flag set, then the NS provider should just ignore the request without generating a fatal error.

4.2.3.2 Data Acknowledgement Indication

N_DATACK_IND

This is a NS provider originated primitive that indicates to the network service user that the remote network service user has acknowledged the data that had previously been sent with the receipt confirmation set.

Format

The format of the message is one M_PROTO message block and its structure is as follows:

```
typedef struct {
    ulong        PRIM_type;         /*always N_DATACK_IND */
} N_datack_ind_t;
```

Parameters

PRIM_type: Indicates the primitive type.

Valid States

This primitive is valid in state NS_DATA_XFER.

New State

The resulting state remains the same (NS_DATA_XFER).

4.2.4 Expedited Data Transfer Service

The expedited data transfer service provides a further means of information exchange on an NC in both directions simultaneously. The transfer of expedited network service data unit (ENSDU) is subject to separate flow control from that applying to NS user data (However, a separate STREAMS message type for expedited data is not available with UNIX® System V Release 3.1. Until a new STREAMS message type is provided, expedited data will be implemented via queue manipulation). The NS provider should guarantee that an expedited-NSDU will not be delivered after any subsequently issued NSDU or expedited-NSDU on that NC. The relationship between normal and expedited data is shown in Table 2. Expedited data can still be delivered when the receiving NS user is not accepting normal data (however this cannot be guaranteed if there are blockages occurring in the lower layers).

The expedited data transfer service is a NS provider option, and its use must be agreed by the two NS users of the NC and the NS provider during NC establishment by using the EX_DATA_OPT parameter on the N_CONN primitives.

4.2.4.1 Expedited Data Transfer Request

N_EXDATA_REQ

This is a NS user originated primitive and is used to indicate to the network provider that the message block contains an ENSDU.

Format

The format of the message is one M_PROTO message block, followed by one or more M_DATA blocks. The NS user must send an integral number of octets of data within the range supported by the NS provider (see N_INFO_ACK). The structure of the M_PROTO message block is as follows:

```
typedef struct {
    ulong        PRIM_type;        /*always N_EXDATA_REQ */
} N_exdata_req_t;
```

Parameters

PRIM_type: Indicates the primitive type.

Valid States

This primitive is valid in state NS_DATA_XFER.

New State

The resulting state remains the same (NS_DATA_XFER).

Acknowledgements

This primitive does not require any acknowledgements, although it may generate a fatal (unrecoverable) error. This is indicated via an M_ERROR STREAMS message type (issued to the NS user with the errno value of EPROTO), which results in the failure of *all system calls on that stream. The applicable errors are as follows:*

EPROTO: This indicates one of the following unrecoverable protocol conditions:

— The network interface was found to be in an incorrect state.

— The amount of NS user data associated with the primitive defines an expedited network service data unit of a size that is outside the range supported by the NS provider.

— Expedited data transfer is either not supported by the NS provider or not requested with the N_CONN_REQ primitive.

NOTE: If the interface is in the NS_IDLE or NS_WRES_RIND states when the provider receives the N_EXDATA_REQ primitive, then the NS provider should discard the request without generating a fatal error.

4.2.4.2 Expedited Data Transfer Indication

N_EXDATA_IND

This is a NS provider originated primitive and is used to indicate to the NS user that this message contains an ENSDU.

Format

The format of the message is one M_PROTO message block, followed by one or more M_DATA blocks. The value of the data in the M_DATA blocks is identical to that supplied with the corresponding N_EXDATA_REQ primitive. The structure of the M_PROTO message block is as follows:

```
typedef struct {
    ulong           PRIM_type;              /*always N_EXDATA_IND */
} N_exdata_ind_t;
```

Parameters

PRIM_type: Indicates the primitive type.

Valid States

This primitive is valid in state NS_DATA_XFER.

New State

The resulting state remains the same (NS_DATA_XFER).

4.2.5 Reset Service

The reset service can be used by the NS user to resynchronize the use of the NC; or by the NS provider to report detected loss of data unrecoverable within the network service. All loss of data which does not involve loss of the NC is reported in this way. Invocation of the reset service will unblock the flow of NSDUs and ENSDUs in case of congestion of the NC; *it will cause the NS provider to discard NSDUs, ENSDUs, or confirmations of receipt associated with the NC (See Table 1), and to notify any NS user or users that did not invoke reset that a reset has occurred.* The service will be completed in finite time, irrespective of the acceptance of the NSDUs, ENSDUs, and confirmations of receipt by the NS users.

4.2.5.1 Reset Request

N_RESET_REQ

This user-originated primitive requests that the NS provider reset the network connection.

Format

The format of the message is one M_PROTO message block, and its structure is as follows:

```
typedef struct {
    ulong        PRIM_type;           /*always N_RESET_REQ */
    ulong        RESET_reason;        /*reason for reset */
} N_reset_req_t;
```

Parameters

PRIM_type: Indicates the primitive type.

RESET_reason: Gives information indicating the cause of the reset.

Valid States

This primitive is valid in the NS_DATA_XFER state.

New State

The resulting state is NS_WACK_RREQ.

Acknowledgements

Successful: This primitive does not require an immediate acknowledgement, although when the resynchronization completes successfully, a N_RESET_CON primitive is issued to the NS user that issued the N_RESET_REQ.

Unsuccessful: A non-fatal error is acknowledged via the N_ERROR_ACK primitive. *In this case the resulting state remains unchanged.*

The following non-fatal error codes are valid:

NOUTSTATE: The primitive was issued from an invalid state.

NSYSERR: A system error has occurred and the UNIX® system error is indicated with the N_ERROR_ACK primitive.

NOTE: If the interface is in the NS_IDLE state when the provider receives the N_RESET_REQ primitive, then the NS provider should discard the message without generating an error.

4.2.5.2 Reset Indication

N_RESET_IND

This network-provider-originated primitive indicates to the NS user that the network connection has been reset.

Format

The format of the message is one M_PROTO message block, and its structure is as follows:

```
typedef struct {
    ulong        PRIM_type;        /*always N_RESET_IND */
    ulong        RESET_orig;       /*reset originator */
    ulong        RESET_reason;     /*reason for reset */
} N_reset_ind_t;
```

Parameters

PRIM_type:	Indicates the primitive type.
RESET_orig:	This parameter indicates the source of the reset.
RESET_reason:	Gives information indicating the cause of the reset.

Valid States

This primitive is valid in the NS_DATA_XFER state.

New State

The new state is NS_WRES_RIND.

4.2.5.3 Reset Response

N_RESET_RES

This user-originated primitive indicates that the NS user has accepted a reset request.

Format

The format of the message is one M_PROTO message block and its structure is the following:

```
typedef struct {
    ulong          PRIM_type;          /*always N_RESET_RES */
} N_reset_res_t;
```

Parameters

PRIM_type: Indicates the primitive type.

Valid States

This primitive is valid in state NS_WRES_RIND.

New State

The new state is NS_WACK_RRES.

Acknowledgements:

Successful: The successful completion of this primitive is indicated via the N_OK_ACK primitive. This results in the data transfer state.

Unsuccessful: An unsuccessful completion of this primitive is indicated by the N_ERROR_ACK primitive. The resulting state remains the same.

The following non-fatal error-codes are valid:

NOUTSTATE: The primitive was issued from an invalid state.

NSYSERR: A system error has occurred and the UNIX system error is indicated in the N_ERROR_ACK primitive.

NOTE: If the interface is in the NS_IDLE state when the provider receives the N_RESET_RES primitive, then the NS provider should discard the message without generating an error.

4.2.5.4 Reset Confirmation

N_RESET_CON

This NS provider-originated primitive indicates to the network user that initiated the reset, that the reset request has been confirmed. The NS provider is allowed to issue the N_RESET_CON primitive to the NS user that initiated the reset even before receiving a N_RESET_RES.

Format

The format of the message is one M_PROTO message block and its structure is the following:

```
typedef struct {
    ulong           PRIM_type;              /*always N_RESET_CON */
} N_reset_con_t;
```

Parameters

PRIM_type: Indicates the primitive type.

Valid States

This primitive is valid in state NS_WCON_RREQ.

New State

The resulting state is NS_DATA_XFER.

4.2.6 Network Connection Release Phase

The NC release service primitives are used to release a NC. The release may be performed by:

— either or both of the NS users to release an established NC;

— the NS provider to release an established NC (all failures to maintain an NC are indicated in this manner);

— the destination NS user to reject an N_CONN_IND;

— by the NS provider to indicate its inability to establish a requested NC.

An NC release is permitted at any time regardless of the current phase of the NC. Once an NC release procedure has been invoked, the NC will be released; a request for release cannot be rejected. The network service does not guarantee delivery of any data once the NC release phase is entered (see Table 1).

4.2.6.1 Disconnect Request

N_DISCON_REQ

This user-originated primitive requests that the NS provider deny a request for a network connection, or disconnect an existing connection.

Format

The format of the message is one M_PROTO message block, followed by one or more M_DATA message blocks (for NS user data). The NS user data may be lost if the NS provider initiates release before the N_DISCON_IND is delivered. Therefore, the NS user data parameter is present only if the originator parameter (as discussed in N_DISCON_IND definition) indicates that the release was originated by an NS user. The NS user may send any integral number of octets of data within a range supported by the NS provider (see N_INFO_ACK).

The structure of the M_PROTO message block is as follows:

```
typedef struct {
    ulong        PRIM_type;          /*always N_DISCON_REQ */
    ulong        DISCON_reason;      /*reason */
    ulong        RES_length;         /*responding address length*/
    ulong        RES_offset;         /*responding address offset*/
    ulong        SEQ_number;         /*sequence number*/
} N_discon_req_t;
```

Parameters

PRIM_type: Indicates the primitive type.

DISCON_reason: Gives information about the cause of the release.

RES_length: Indicates the length of the address of the responding address parameter. The responding address parameter is an optional parameter, and is present in the primitive only in the case where the primitive is used to indicate rejection of an NC establishment attempt by an NS user. The responding address parameter conveys the network address of the NS user entity from which the N_DISCON_REQ was issued and under certain circumstances (e.g. call redirection, generic addressing, etc.) may be different from the "destination address" in the corresponding N_CONN_REQ primitive.

RES_offset: Is the offset from the beginning of the M_PROTO message block where the network address begins.

SEQ_number: When non-zero, it identifies the sequence number of the N_CONN_IND message being rejected. This number is used by the NS provider to associate the N_DISCON_REQ with an unacknowledged N_CONN_IND that is to be rejected. If the N_DISCON_REQ is rejecting a NC that is already established (or rejecting a N_CONN_REQ that the NS user had previously sent and has not yet been confirmed), then this field should have a value of 0.

Valid States

This primitive is valid in states NS_WCON_CREQ, NS_WRES_CIND, NS_DATA_XFER, NS_WCON_RREQ, NS_WRES_RIND.

New State

The new state depends on the original state (see Appendix B, Table B-8).

Acknowledgements:

The NS provider should generate one of the following acknowledgements upon receipt of this primitive:

— **Successful**: Successful completion is indicated via the N_OK_ACK primitive.

— **Unsuccessful (Non-fatal errors)**: Errors are indicated via the N_ERROR_ACK primitive. The applicable non-fatal errors are as follows:

NOUTSTATE: The primitive was issued from an invalid state.

NBADDATA: The amount of user data specified was outside the range supported by the NS provider.

NSYSERR: A system error has occurred and the UNIX system error is indicated in the primitive.

NBADSEQ: The specified sequence number referred to an invalid N_CONN_IND message,or the N_DISCON_REQ is rejecting an NC that is already established (or rejecting an N_CONN_REQ that the NS user had previously sent and has not yet been confirmed) and the value of the sequence number is not 0.

4.2.6.2 Disconnect Indication

N_DISCON_IND

This network-provider originated primitive indicates to the NS user that either a request for connection has been denied or an existing connection has been disconnected.

Format

The format of the message is one M_PROTO message block, followed by one or more M_DATA blocks. The value of the NS user data parameter is identical to the value in the corresponding N_DISCON_REQ primitive. The NS user data parameter is present only if the originator parameter indicates that the release was initiated by the NS user.

The struct of the M_PROTO message block is as follows:

```
typedef struct {
    ulong        PRIM_type;          /*always N_DISCON_IND */
    ulong        DISCON_orig;        /*originator */
    ulong        DISCON_reason;      /*reason */
    ulong        RES_length;         /*responding address length */
    ulong        RES_offset;         /*responding address offset */
    ulong        SEQ_number;         /*sequence number*/
} N_discon_ind_t;
```

Parameters

PRIM_type: Indicates the primitive type.

DISCON_orig: Indicates the source of the NC release.

DISCON_reason: Gives information about the cause of the release.

RES_length: Indicates the length of the address of the responding address parameter. The responding address parameter is an optional parameter, and is present in the primitive only in the case where the primitive is used to indicate rejection of an NC establishment attempt by an NS user. When not present, the value of this parameter is zero. When present, the value of the disconnect address parameter is identical to that supplied with the corresponding N_DISCON_REQ primitive.

RES_offset: Is the offset from the beginning of the M_PROTO message block where the network address begins.

SEQ_number: When its value is non-zero, it identifies the sequence number associated with the N_CONN_IND that is being aborted.

The value of this parameter must be zero when:

a. indicating the rejection of a previously issued N_CONN_REQ primitive; or

b. indicating the release of a NC that is already successfully established.

When this field is non-zero and its value is the same as the sequence number assigned to an unacknowledged N_CONN_IND, it indicates that the NS provider is canceling the unacknowledged N_CONN_IND.

Valid States

The valid states are NS_WCON_CREQ, NS_WRES_CIND, NS_DATA_XFER, NS_WCON_RREQ, and NS_WRES_RIND.

New State

The new state is NS_IDLE (except when number of outstanding connect indications is greater than 1, in which case the resulting state is NS_WRES_CIND).

4.3 CLNS: Primitive Format and Rules

This section describes the format of the CLNS primitives and the rules associated with these primitives. The values of the QOS parameters associated with each unitdata transmission are selected via the N_OPTMGMT_REQ primitive.

4.3.1 Unitdata Request

N_UNITDATA_REQ

This primitive requests that the NS provider send the specified datagram to the specified destination.

Format

The format of the message is one M_PROTO message block followed by one or more M_DATA message blocks. The structure of the M_PROTO is as follows:

```
typedef struct {
    ulong     PRIM_type;             /*always N_UNITDATA_REQ*/
    ulong     DEST_length;           /*destination address length*/
    ulong     DEST_offset;           /*destination address offset*/
    ulong     RESERVED_field[2];     /*reserved field for DLPI compatibility*/
} N_unitdata_req_t;
```

Parameters

PRIM_type: Indicates the primitive type.

DEST_length: Indicates the length of the destination address.

DEST_offset: Indicates the offset of the destination address from the beginning of the M_PROTO message block.

RESERVED_field: This is a reserved field (for compatibility with DLPI) whose value must be set to zero for both entries of the array.

Valid States

This primitive is valid in state NS_IDLE.

New State

The resulting state remains unchanged.

Acknowledgements

Successful: There is no acknowledgement for the successful completion of this primitive.

Non-Fatal Error: If a non-fatal error occurs, it is the responsibility of the NS provider to report it via the N_UDERROR_IND primitive. The following non-fatal error codes are allowed:

NBADADDR: The network address as specified in the primitive was in an incorrect format, or the address contained illegal information.

NBADDATA: The amount of user data specified was outside the range supported by the NS provider.

NOUTSTATE: The primitive was issued from an invalid state.

Fatal Error: Fatal errors are indicated via an M_ERROR STREAMS message type (issued to the NS user with the errno value of EPROTO), which results in the failure of all UNIX system calls on the stream. The fatal errors are as follows:

EPROTO: This indicates one of the following unrecoverable protocol conditions:

— The network service interface was found to be in an incorrect state.

— The amount of NS user data associated with the primitive defines a network service data unit larger than that allowed by the NS provider.

4.3.2 Unitdata Indication

N_UNITDATA_IND

This primitive indicates to the NS user that a datagram has been received from the specified source address.

Format

The format of the message is one M_PROTO message block followed by one or more M_DATA blocks containing at least one byte of data. The format of the M_PROTO is as follows:

```
typedef struct {
    ulong        PRIM_type;        /* always N_UNITDATA_IND */
    ulong        DEST_length;      /* destination address length */
    ulong        DEST_offset;      /* destination address offset */
    ulong        SRC_length;       /* source address length */
    ulong        SRC_offset;       /* source address offset */
    ulong        ERROR_type;       /* specifies the reason for the error*/
} N_unitdata_ind_t;
```

Parameters

PRIM_type: Indicates the primitive type.

DEST_length: Indicates the length of the destination address. The address is the same as in the corresponding N_UNITDATA_REQ primitive.

DEST_offset: Indicates the offset of the destination address from the beginning of the M_PROTO message block.

SRC_length: Indicates the length of the source network address. This address is the same as the value associated with the stream on which the N_UNITDATA_REQ was issued.

SRC_offset: Indicates the offset of the source address from the beginning of the M_PROTO message block.

ERROR_type: Specifies the reason for the error. The possible values are:

N_UD_CONGESTION: This packet experienced congestion during its delivery.

Valid States

This primitive is valid in state NS_IDLE.

New State

The resulting state remains unchanged.

4.3.3 Unitdata Error Indication

N_UDERROR_IND

This primitive indicates to the NS user that a datagram with the specified destination address and QOS parameters has resulted in an error condition.

Format

The format of the message is one M_PROTO message block, and its structure is as follows:

```
typedef struct {
    ulong       PRIM_type;          /*always N_UDERROR_IND */
    ulong       DEST_length;        /*destination address length */
    ulong       DEST_offset;        /*destination address offset */
    ulong       RESERVED_field;     /*reserved field for DLPI compatibility*/
    ulong       ERROR_type;         /*error type */
} N_uderror_ind_t;
```

Parameters

PRIM_type:	Indicates the primitive type.
DEST_length:	Indicates the length of the destination address. The address is the same as in the corresponding N_UNITDATA_REQ primitive.
DEST_offset:	Indicates the offset of the destination address from the beginning of the M_PROTO message block.
RESERVED_field:	This field is reserved whose value must be set to zero.
ERROR_type:	Specifies the reason for the error.

Valid States

This primitive is valid in state NS_IDLE.

New State

The resulting state remains unchanged.

5. Diagnostics Requirements

Two error handling facilities should be provided to the network service user: one to handle non-fatal errors, and the other to handle fatal errors.

5.1 Non-Fatal Error Handling Facility

These are errors that do not change the state of the network service interface as seen by the network service user, and provide the user the option of reissuing the network service primitive with the corrected options specification. The non-fatal error handling is provided only to those primitives that require acknowledgements, and uses the N_ERROR_ACK to report these errors. These errors retain the state of the network service interface the same as it was before the network provider received the primitive that was in error. Syntax errors and rule violations are reported via the non-fatal error handling facility.

5.2 Fatal Error Handling Facility

These errors are issued by the NS provider when it detects errors that are not correctable by the network service user, or if it is unable to report a correctable error to the network service user. Fatal errors are indicated via the STREAMS message type M_ERROR with the UNIX system error EPROTO. The M_ERROR STREAMS message type will result in the failure of all the UNIX system calls on the stream. The network service user can recover from a fatal error by having all the processes close the files associated with the stream, and then reopening them for processing.

6. References

1. CCITT X.213, (Geneva, 1986), "Network Service Definition for Open Systems Interconnection (OSI) for CCITT Applications", (Grey Book)
2. ISO 8348 - "Information Processing Systems - Data Communications - Network Service Definition", 4/15/87
3. ISO 8348/AD1 - "Information Processing Systems - Data Communications - Network Service Definition - Addendum 1: Connectionless Mode Transmission", 4/15/87
4. ISO 8473 - "Information Processing Systems - Data Communications Protocol for Providing the Connectionless Mode Network Service", SC6 N4542
5. ISO 8208 - "Information Processing Systems - X.25 Packet Level Protocol for Data Terminal Equipment", 9/15/87
6. ISO 8878 - "Information Processing Systems - Data Communications - Use of X.25 to Provide the OSI Connection-Mode Network Service", 9/1/87
7. System V Interface Definition, Issue 2 - Volume 3
8. CCITT X.210, (Geneva, 1984), "Open Systems Interconnection (OSI) Layer Service Definition Conventions", (Red Book)

7. Addendum for OSI Conformance

This section describes the formats and rules that are specific to OSI. The addendum must be used along with the generic NPI as defined in the main document when implementing a NS provider that will be configured with the OSI Transport Layer.

7.1 Quality of Service: Model & Description

The "Quality of Service" characteristics apply to both CONS as well as CLNS.

7.1.1 QOS Overview

QOS (Quality of Service) is described in terms of QOS parameters. There are two types of QOS parameters:

1. Those that are "negotiated" on a per-connection basis during NC establishment. (CLNS does not support end-to-end QOS parameter negotiation).
2. Those that are not negotiated and their values are selected/determined by local management methods.

Table 8 summarizes the supported parameters both for connection-mode and connectionless network service. For more details on the definition of the QOS parameters, refer to CCITT X.213 [1] and ISO 8348 [2].

PARAMETER	SERVICE MODE	NEGOTIATION
NC Establishment Delay	CONS	Local
NC Establishment Failure Probability	CONS	Local
Throughput	CONS	End-to_End
Transit Delay	CONS, CLNS	End-to-End (for CONS) Local (for CLNS)
Residual Error Rate	CONS, CLNS	Local
NC Resilience	CONS	Local
Transfer Failure Probability	CONS	Local
NC Release Delay	CONS	Local
NC Release Failure Probability	CONS	Local
Protection	CONS, CLNS	End-to-End (for CONS) Local (for CLNS)
Priority	CONS, CLNS	End-to-End (for CONS) Local (for CLNS)
Maximum Acceptable Cost	CONS, CLNS	Local

TABLE 8. Supported QOS Parameters

7.1.2 QOS Parameter Formats

This section describes the formats of the QOS parameters for CONS and/or CLNS services. The requested QOS parameter values apply to complete NSDUs.

7.1.2.1 NC Establishment Delay

This parameter applies to CONS only. It is defined as the maximum acceptable delay between a N_CONN_REQ and the corresponding N_CONN_CON primitive. NC establishment delay is measured in milliseconds.

Format:

```
long    nc_estab_delay;         /*maximum NC establishment delay*/
```

nc_estab_delay: Is the maximum acceptable delay value for NC establishment.

7.1.2.2 NC Establishment Failure Probability

This parameter applies to CONS only. NC Establishment Failure Probability is the percent ratio (rounded to the nearest integer) of total NC establishment failures to total NC establishment attempts in a measurement sample. A measurement sample consists of 100 NC establishment attempts.

NC establishment failure occurs due to NS provider behavior such as misconnection, NC refusal, and excessive delay. NC establishment attempts that fail due to NS user behavior such as error, NC refusal, or excessive delay are excluded in calculating NC establishment failure probability.

Format:

```
long    nc_estab_fail_prob;     /*maximum NC estab failure probability*/
```

nc_estab_fail_prob: Is the maximum acceptable percent value (rounded to the nearest integer) for the NC establishment failure probability.

7.1.2.3 Throughput

This parameter applies to CONS only, is specified separately for each direction of transfer, and has end-to-end significance. Throughput is defined in terms of at least two successfully transferred NSDUs presented continuously to the NS provider at the maximum rate the NS provider can continuously sustain, and unconstrained by flow control applied by the receiving NS user. Given a sequence of "n" NSDUs (where is greater than or equal to two; suggested value is 100), throughput is defined to be the smaller of:

a. the number of NS user data octets contained in the last "n-1" NSDUs divided by the time between the first and the last N_DATA_REQs in the sequence; and

b. the number of NS user data octets contained in the last "n-1" NSDUs divided by the time between the first and the last N_DATA_INDs in the sequence.

Throughput should be measured and specified in bits per second.

Format:

```
typedef struct {
    long        thru_targ_value;
    long        thru_min_value;
} thru_values_t;
```

thru_targ_value: Specifies the requested QOS value for throughput for data transfer between the two NS users.

thru_min_value: Specifies the requested lowest acceptable QOS value for throughput between the two NS users.

7.1.2.4 Transit Delay

This parameter applies to CONS as well as CLNS. Transit Delay is the elapsed time between a N_DATA_REQ and the corresponding N_DATA_IND (calculated on successfully transferred NSDUs only). The pair of values specified for an NC applies to both directions of transfer. The specified values are averages (based on 100 samples using a NSDU size of 128 bytes).

Transit Delay should be measured in milliseconds.

Format:

```
typedef struct {
    long        td_targ_value;
    long        td_max_value;
} td_values_t;
```

td_targ_value: Specifies the desired QOS value for transit delay between the two NS users.

td_max_value: Specifies the maximum QOS value that the source NS user will agree for transit delay between the two NS users.

7.1.2.5 Residual Error Rate

This parameter applies to both CONS as well as CLNS. Residual Error Rate is the percent ratio (rounded to the nearest integer) of total incorrect, lost, and duplicate NSDUs to total NSDUs transferred across the NS boundary during a measurement period. The measurement period will be 3600 seconds.

Format:

```
    long    residual_error_rate;        /*maximum acceptable residual error rate*/
```

residual_error_rate: Specifies the maximum acceptable percent value (rounded to the nearest integer) of the residual error rate.

7.1.2.6 NC Resilience

This parameter applies to CONS only. NC Resilience specifies the percent probability (rounded to the nearest integer) of a NS provider invoked NC release or a NS provider invoked reset during a specified time interval on an established NC. The time interval will be 3600 seconds.

Format:

```
long    nc_resilience;          /*maximum acceptable nc resilience*/
```

nc_resilience: Specifies the maximum acceptable value for NC resilience.

7.1.2.7 Transfer Failure Probability

This parameter applies to CONS only. It is the percent ratio (rounded to the nearest integer) of total transfer failures to total transfer samples observed during a performance measurement. A transfer sample is a discrete observation of NS provider performance in transferring NSDUs between specified sending and receiving NS user. A transfer sample will last for the duration of the NC. A transfer failure is a transfer sample in which the observed performance is worse than the specified minimum acceptable level. A transfer failure is identified by comparing the measured values for the supported performance parameters with specified transfer failure thresholds. The three supported performance parameters are throughput, transit delay, and residual error rate.

Format:

```
long        xfer_fail_prob;     /*maximum xfer failure prob*/
```

xfer_fail_prob: Specifies the maximum acceptable percent value (rounded to the nearest integer) for transfer failure probability.

7.1.2.8 NC Release Delay

This parameter applies to CONS only. NC Release Delay is defined as the maximum acceptable delay between a NS user invoked N_DISCON_REQ and the successful release of the NC at the peer NS user. NC Release Delay is specified independently for each NS user. It does not apply in cases where NC release is invoked by the NS provider. NC release delay should be measured in milliseconds.

Format:

```
long    nc_rel_delay;           /*maximum nc release delay*/
```

nc_rel_delay: Is the maximum acceptable value for NC release delay.

7.1.2.9 NC Release Failure Probability

This parameter applies to CONS only. It is the percent ratio (rounded to the nearest integer) of total NC release requests resulting in release failure to total NC release requests included in a measurement sample. A measurement sample consists of a 100 NC release requests. This parameter is specified independently for each NS user.

A release failure is defined to occur for a particular NS user, if that user does not receive a N_DISCON_IND within a specified maximum NC release delay of the NS user issuing the N_DISCON_REQ (given that the former NS user has not issued a N_DISCON_REQ).

Format:

```
long    nc_rel_fail_prob;        /*maximum nc rel fail probability*/
```

nc_rel_fail_prob: Is the maximum acceptable percent value (rounded to the nearest integer) of NC release failure probability.

7.1.2.10 Protection

This parameter applies to both CONS and CLNS.

It specifies the extent to which the NS provider attempts to prevent unauthorized monitoring or manipulation of NS user originated information.

```
/* Types of protection */

#define N_NO_PROT                 0x00000000L   /*no protection*/
#define N_PASSIVE_PROT            0x00000001L   /*protection against passive monitoring*/
#define N_ACTIVE_PROT             0x00000002L   /*protection against active monitoring*/
#define N_ACTIVE_PASSIVE_PROT     0x00000003L   /*maximum protection*/
```

Four protection options are provided:

1. No protection features;
2. Protection against passive monitoring;
3. Protection against modification, replay, addition, or deletion
4. Both 2 and 3.

Format:

```
typedef struct {
    long        protect_targ_value;   /*target protection*/
    long        protect_min_value;    /*minimum protection*/
} protection_values_t;
```

protect_targ_value: Specifies the target protection of the NS user originated information.

protect_min_value: Specifies the lowest quality acceptable of protection of the NS user originated information.

7.1.2.11 Priority

This parameter applies to both CONS and CLNS.

It specifies the target priority of:

a. an NSDU in relation to any other NSDUs (for CLNS);

b. a NC (for CONS).

The number of priority levels is limited to 15 (where level 1 is the highest priority and level 15 is the lowest priority).

Format:

```
typedef struct {
    long          priority_targ_value;    /*target priority*/
    long          priority_min_value;     /*minimum priority*/
} priority_values_t;
```

priority_targ_value: Specifies the target NC priority level.

priority_min_value: Specifies the lowest quality acceptable of the NC priority level.

7.1.2.12 Maximum Acceptable Cost

This parameter applies to both CONS and CLNS. It specifies the maximum acceptable cost in local currency (composed of communications and end-system resource costs), or indicates to the NS provider that it should choose the least expensive means available to it.

Format

```
      long    max_accept_cost;          /*acceptable cost maximum*/


/* Choose least expensive means */

#define N_LEAST_EXPENSIVE       0x00000000L    /*choose least expensive means*/
```

max_accept_cost: Specifies the maximum acceptable cost in local currency.

7.1.3 QOS Data Structures

The quality of services parameters are organized into six different structures for simplicity:

N_QOS_CO_RANGE1: QOS range requested for connection-mode service as used with the N_CONN_REQ and N_CONN_IND primitives.

N_QOS_CO_SEL1: QOS values selected for the connection-mode service as used with the N_CONN_RES and N_CONN_CON primitives.

N_QOS_CL_RANGE1: Range of QOS values for connectionless-mode service as specified with the *QOS_range_length* and *QOS_range_offset* parameters of the N_INFO_ACK primitive.

N_QOS_CL_SEL1: QOS values supported/selected for connectionless-mode service as specified with the *QOS_length* and *QOS_offset* parameters of the N_INFO_ACK and the N_OPTMGMT_REQ primitives.

N_QOS_CO_OPT_RANGE1: Range of QOS values for connection-mode service as specified with the *QOS_range_length* and *QOS_range_offset* parameters of the N_INFO_ACK primitive.

N_QOS_CO_OPT_SEL1: Default QOS values supported/selected for connection-mode service as specified with the *QOS_length* and *QOS_offset* parameters of the N_INFO_ACK and the N_OPTMGMT_REQ primitives.

7.1.3.1 Structure N_QOS_CO_RANGE1

Structure N_qos_co_range1 defines the QOS parameters that are transferred between the source and destination NS users for a NC. The format of this structure is as follows:

```
typedef struct {
    ulong          n_qos_type;              /*always N_QOS_CO_RANGE*/
    thru_values_t src_throughput_range;   /*source throughput range*/
    thru_values_t dest_throughput_range; /*dest throughput range*/
    td_values_t transit_delay_range;        /*transit delay range*/
    protection_values_t protection_range; /*protection range*/
    priority_values_t priority_range;        /*priority target*/
} N_qos_co_range1_t;
```

This structure should be used in the *QOS_length* and *QOS_offset* fields of the following NPI primitives:

- N_CONN_REQ
- N_CONN_IND

7.1.3.2 Structure N_QOS_CO_SEL1

Structure N_qos_co_sel1 defines the QOS parameters that are transferred between the destination and source NS users for a NC. The format of this structure is as follows:

```
typedef struct {
    ulong          n_qos_type;            /*always N_QOS_CO_SEL*/
    long           src_throughput_sel;    /*source throughput selected*/
    long           dest_throughput_sel;   /*destination throughput selected*/
    long           transit_delay_sel;     /*transit delay selected*/
    long           protection_sel;        /*NC protection selected*/
    long           priority_sel;          /*NC priority selected*/
} N_qos_co_sel1_t;
```

This structure should be used in the *QOS_length* and *QOS_offset* fields of the following NPI primitives:

- N_CONN_RES
- N_CONN_CON

7.1.3.3 Structure N_QOS_CL_RANGE1

Structure N_qos_cl_range1 defines the range of QOS parameter values that are supported by the NS provider. The format of the structure is as follows:

```
typedef struct {
    ulong          n_qos_type;            /*always N_QOS_CL_RANGE*/
    td_values_t transit_delay_max;        /*maximum transit delay*/
    long           residual_error_rate;   /*residual error rate*/
    protection_values_t protection_range; /*target protection*/
    priority_values_t priority_range;     /*target priority*/
    long           max_accept_cost;       /*maximum acceptable cost*/
} N_qos_cl_range1_t;
```

This structure should be used in the:

- *QOS_range_length* and *QOS_range_offset* fields of the N_INFO_ACK primitive;

7.1.3.4 Structure N_QOS_CL_SEL1

Structure N_qos_cl_sel1 defines the QOS parameters values that will apply to each unitdata transmission between the CLNS users. The format of the structure is as follows:

```
typedef struct {
    ulong          n_qos_type;           /*always N_QOS_CL_sel*/
    long           transit_delay_max;    /*maximum transit delay*/
    long           residual_error_rate;  /*residual error rate*/
    long           protection_sel;       /*protection selected*/
    long           priority_sel;         /*priority selected*/
    long           max_accept_cost;      /*maximum acceptable cost*/
} N_qos_cl_sel1_t;
```

This structure should be used in the:

- *QOS_length* and *QOS_offset* fields of the N_INFO_ACK primitive;
- *QOS_length* and *QOS_offset* fields of the N_OPTMGMT_REQ primitive.

7.1.3.5 Structure N_QOS_CO_OPT_RANGE1

Structure N_qos_opt_range1 defines the range of the default QOS parameter values that are supported by the NS provider. This allows the NS user to select values within the range supported by the NS provider. The format of the structure is as follows:

```
typedef struct {
    ulong          n_qos_type;           /*always N_QOS_CO_OPT_RANGE*/
    thru_values_t src_throughput;        /*source throughput values*/
    thru_values_t dest_throughput;       /*dest throughput values*/
    td_values_t transit_delay;           /*transit delay values*/
    long           nc_estab_delay;       /*NC establishment delay*/
    long           nc_estab_fail_prob;   /*NC estab failure probability*/
    long           residual_error_rate;  /*residual error rate*/
    long           xfer_fail_prob;       /*transfer failure probability*/
    long           nc_resilience;        /*NC resilience*/
    long           nc_rel_delay;         /*NC release delay*/
    long           nc_rel_fail_prob;     /*NC release fail probability*/
    protection_values_t protection_range; /*protection range*/
    priority_values_t priority_range;    /*priority range*/
    long           max_accept_cost;      /*maximum acceptable cost*/
} N_qos_co_opt_range1_t;
```

This structure should be used in the:

- *QOS_range_length* and *QOS_range_offset* fields of the N_INFO_ACK primitive;

7.1.3.6 Structure N_QOS_CO_OPT_SEL1

Structure N_qos_opt_sel1 defines the selected QOS parameter values. The format of the structure is as follows: as follows:

```
typedef struct {
    ulong           n_qos_type;             /*always N_QOS_CO_OPT_SEL*/
    thru_values_t src_throughput;          /*source throughput values*/
    thru_values_t dest_throughput;         /*dest throughput values*/
    td_values_t transit_delay;             /*transit delay values*/
    long            nc_estab_delay;         /*NC establishment delay*/
    long            nc_estab_fail_prob;     /*NC estab failure probability*/
    long            residual_error_rate;    /*residual error rate*/
    long            xfer_fail_prob;         /*transfer failure probability*/
    long            nc_resilience;          /*NC resilience*/
    long            nc_rel_delay;           /*NC release delay*/
    long            nc_rel_fail_prob;       /*NC release failure probability*/
    long            protection_sel;         /*protection selected*/
    long            priority_sel;           /*priority selected*/
    long            max_accept_cost;        /*maximum acceptable cost*/
} N_qos_co_opt_sel1_t;
```

This structure should be used in the:

- *QOS_length* and *QOS_offset* fields of the N_INFO_ACK primitive;
- *QOS_length* and *QOS_offset* fields of the N_OPTMGMT_REQ primitive.

7.2 NPI Primitives: Rules for OSI Conformance

The following are the rules that apply to the NPI primitives for OSI compatibility.

7.2.1 Local Management Primitives

7.2.1.1 N_INFO_ACK

Parameters

NSDU_size: A value greater than zero specifies the maximum size of a Network Service Data Unit (NSDU); a value of 0 specifies that the transfer of normal data is not supported by the NS provider, and a value of -1 specifies that there is no limit on the size of a NSDU.

ENSDU_size: A value between 1 and 32 inclusive specifies the maximum size of an Expedited Network Service Data Unit (ENSDU); a value of 0 specifies that the transfer of expedited data is not supported by the NS provider.

CDATA_size: A value between 1 and 128 inclusive specifies the maximum number of octets of data that may be associated with connection establishment primitives. A value of 0 specifies that the NS provider does not allow data to be sent with connection establishment primitives.

When used in an OSI conformant environment, CDATA_size shall always equal 128.

DDATA_size: A value between 1 and 128 inclusive specifies the maximum number of octets of data that may be associated with the disconnect primitives; a value of 0 specifies that the NS provider does not allow data to be sent with the disconnect primitives.

When used in an OSI conformant environment, DDATA_size shall always equal 128.

ADDR_size: A value between 1 and 40 indicates the maximum size of a network address in decimal digits.

When used in an OSI conformant environment, ADDR_size shall always equal 40 in order to accomodate a full NSAP address.

QOS_length: Indicates the length in bytes of the default/negotiated/selected values of the QOS parameters. The applicable QOS parameters are defined in the following structures:

i. N_QOS_CO_OPT_SEL1 for CONS; and

ii. N_QOS_CL_SEL1 for CLNS.

In the connection-mode environment, when this primitive is invoked before the NC is established on the stream, the values returned specify the the default values supported by the NS provider. When this primitive is invoked after a NC has been established on the stream, the values returned indicate the negotiated values for the QOS parameters. In the connectionless environment, these values represent the default or the selected QOS parameter values.

In case a QOS parameter is not supported by the NS Provider, a value of QOS_UNKNOWN will be returned. In the case where no QOS parameters are supported by the NS provider, the length of this field will be zero.

QOS_range_length: Indicates the length in bytes, of the available range of QOS parameters values supported by the NS provider. These ranges are used by the NS user to select QOS parameter values that are valid with the NS provider.

The applicable QOS parameters are defined in the following structures:

i. N_QOS_CO_OPT_RANGE1 for CONS; and

ii. N_QOS_CL_RANGE1 for CLNS.

QOS parameter values are selected, or the default values altered via the N_OPTMGMT_REQ primitive. In the connection-mode environment, the values for end-to-end QOS parameters may be specified with the N_CONN primitives for negotiation. If the NS provider does not support a certain QOS parameter, its value will be set to QOS_UNKNOWN. In the case where no QOS parameters are supported by the NS provider, the length of this field will be zero.

NIDU_size: This indicates the amount of user data that may be present in a N_DATA primitive. The NIDU_size should not be larger than the NSDU_size specification.

SERV_type: Specifies the service type supported by the NS provider. The possible values can be N_CONS, N_CLNS, (or both by using N_CONS|N_CLNS).

If the SERV_type is N_CLNS, the following rules will apply:

— The ENSDU_size, CDATA_size, DDATA_size, and DEFAULT_rc_sel fields are not used and their values should be set to 0;

— The NSDU_size should be the same as the NIDU_size.

NODU_size: The NODU_size specifies the optimal NSDU size in octets of an NSDU given the current routing information.

PROTOID_length: The length of the protocol identifiers to be bound.

PROTOID_offset: The offset of the protocol identifiers to be bound, from the beginning of the block.

7.2.1.2 N_OPTMGMT_REQ

Parameters

QOS_length: Indicates the length of the default values of the QOS parameters as selected by the NS user. In the connection-mode environment these values will be used in subsequent N_CONN_REQ primitives on the stream that do not specify values for these QOS parameters. In the connection-less environment, these values represent the selected QOS values that would apply to each unitdata transmission. The applicable QOS parameters are defined in the following structures:

i. N_QOS_CO_OPT_SEL1 for CONS; and

ii. N_QOS_CL_SEL1 for CLNS.

If the NS user cannot determine the value of a QOS parameter, its value should be set to QOS_UNKNOWN. If the NS user does not specify any QOS parameter values, the length of this field should be set to zero.

7.2.2 CONS: Connection Establishment Phase

Rules for QOS Parameter Negotiation

The negotiation for NC throughput and NC transit-delay QOS parameters are conducted as follows:

a. in the N_CONN_REQ primitive, the source NS user specifies two values for each negotiable QOS parameter:

   1. a "target" which is the QOS value desired; and

   2. a "lowest acceptable" QOS value to which the source NS user will agree;

   The value of each of these parameters must be within the limit of the allowable values defined for the network service. "Default" values for these parameters are supported by the NS provider. The default values may be selected by the NS user via the N_OPTMGMT_REQ primitive.

b. if the NS provider agrees to provide a value of QOS which is in the range between the "target" and the "lowest acceptable" QOS values, inclusive, of the

N_CONN_REQ, then the NS provider specifies two parameters in the N_CONN_IND issued to the destination NS user:

1. an "available" value which is the QOS value the NS provider is willing to provide; and
2. a "lowest acceptable" QOS value which is identical to the "lowest acceptable" value specified in the N_CONN_REQ; (if the NS provider does not agree to provide QOS in the given range, then the NC establishment request is rejected);

c. if the destination NS user agrees to a QOS value which is in the range between the "available" and the "lowest acceptable" QOS values, inclusive, of the N_CONN_IND, then the destination NS user specifies a single parameter, "selected" in the N_CONN_RES; this parameter is the QOS value the destination NS user agrees to; (if the destination NS user does not agree to a QOS in the given range, then the NC establishment request is rejected);

d. the NS provider adopts the QOS value for the NC which was specified by the destination NS user and supplies this as a single parameter, "selected", in the N_CONN_CON primitive.

- **The negotiation for the NC protection parameter is conducted as follows:**

a. In the N_CONN_REQ primitive, the calling NS user specifies values for the "Target" and "Lowest Quality Acceptable" subparameters; permitted value assignments are:

Case1: both the "Target" and "Lowest Quality Acceptable" are "unspecified";

Case2: values other than "unspecified" are specified for both "Target" and "Lowest Quality Acceptable";

Case3: a value other than "unspecified" is specified for "Target" and the "Lowest Quality Acceptable" is "unspecified".

NOTE: In case where "Target" is "unspecified", the "Lowest Quality Acceptable" must also be "unspecified".

b. If the NS provider does not support a choice of NC protection levels, the value of the "Target" parameter is conveyed by the NS provider and passed to the called NS user unchanged as the "Available" subparameter in the N_CONN_IND primitive;

c. If the NS provider does support a choice of NC protection levels, then:

1. In Case1, the NS provider determines the QOS value to be offered on the NC and specifies it in the "Available" subparameter in the N_CONN_IND primitive;

2. In Case2 and Case3, if the NS provider does not agree to provide a QOS in the requested range, then the NC establishment attempt is rejected as described in clause 13.5 of ISO 8348. If the NS provider does agree to provide a QOS in the requested range, then in the N_CONN_IND primitive, the "Available" subparameter specifies the highest QOS value within the range which the NS provider is willing to provide.

d. The value of the "Lowest Quality Acceptable" subparameter in the N_CONN_IND primitive is identical to that in the N_CONN_REQ primitive;

e. If the value of the "Available" subparameter of the N_CONN_IND primitive is "unspecified" then:

1. if the called NS user does not agree to accept establishment of a NC with this unspecified quality, the NS user rejects the NC establishment attempt as described in clause 13.4 of ISO 8348;

2. if the called NS user does agree, then the NS user specifies the value "unspecified" in the "Selected" subparameter of the N_CONN_RES primitive.

f. If the value of the "Available" subparameter in the N_CONN_IND primitive is not "unspecified" then:

1. if the called NS user does not agree to a QOS in the range identified by the "Available" and "Lowest Quality Acceptable" subparameters of the N_CONN_IND primitive, then the NS user rejects the NC establishment attempt as described in clause 13.4 of ISO 8348;

2. if the called NS user does agree to a QOS in the identified range, then the NS user specifies the agreed value in the "Selected" subparameter of the N_CONN_RES primitive.

g. In the N_CONN_CON primitive, the "Selected" subparameter has a value identical to that of "Selected" in the N_CONN_RES primitive.

- **The negotiation of the NC priority parameter is conducted as follows:**

a. In the N_CONN_REQ primitive, the calling NS user specifies values for the "Target" and "Lowest Quality Acceptable" subparameters; permitted value assignments are:

Case1: both the "Target" and "Lowest Quality Acceptable" are "unspecified";

Case2: values other than "unspecified" are specified for both "Target" and "Lowest Quality Acceptable";

Case3: a value other than "unspecified" is specified for "Target" and the "Lowest Quality Acceptable" is "unspecified".

NOTE: In case where "Target" is "unspecified", the "Lowest Quality

Acceptable" must also be "unspecified".

b. If the NS provider does not support a choice of NC priority levels, the value of the "Target" parameter is conveyed by the NS provider and passed to the called NS user unchanged as the "Available" subparameter in the N_CONN_IND primitive;

c. If the NS provider does support a choice of NC priority levels, then:

    1. In Case1, the NS provider determines the QOS value to be offered on the NC and specifies it in the "Available" subparameter in the N_CONN_IND primitive;

    2. In Case2 and Case3, if the NS provider does not agree to provide a QOS in the requested range, then the NC establishment attempt is rejected as described in clause 13.5 of ISO 8348. If the NS provider does agree to provide a QOS in the requested range, then in the N_CONN_IND primitive, the "Available" subparameter specifies the highest QOS value within the range which the NS provider is willing to provide.

d. The value of the "Lowest Quality Acceptable" subparameter in the N_CONN_IND primitive is identical to that in the N_CONN_REQ primitive;

e. If the value of the "Available" subparameter of the N_CONN_IND primitive is "unspecified" then:

    1. if the called NS user does not agree to accept establishment of a NC with this unspecified quality, the NS user rejects the NC establishment attempt as described in clause 13.4 of ISO 8348;

    2. if the called NS user does agree, then the NS user specifies the value "unspecified" in the "Selected" subparameter of the N_CONN_RES primitive.

f. If the value of the "Available" subparameter in the N_CONN_IND primitive is not "unspecified" then:

    1. if the called NS user does not agree to a QOS in the range identified by the "Available" and "Lowest Quality Acceptable" subparameters of the N_CONN_IND primitive, then the NS user rejects the NC establishment attempt as described in clause 13.4 of ISO 8348;

    2. if the called NS user does agree to a QOS in the identified range, then the NS user specifies the agreed value in the "Selected" subparameter of the N_CONN_RES primitive.

g. In the N_CONN_CON primitive, the "Selected" subparameter has a value identical to that of "Selected" in the N_CONN_RES primitive.

Rules for QOS Parameter Selection

When a NS user/provider cannot determine the value of a QOS field, it should return a value of QOS_UNKNOWN.

```
#define QOS_UNKNOWN            -1
```

Rules for Receipt Confirmation Selection

- The receipt confirmation selection parameter values on the various primitives are related such that:
  1. on the N_CONN_REQ, either of the defined values may occur (namely, "use of receipt confirmation", or "no use of receipt confirmation").
  2. on the N_CONN_IND, the value is either equal to the value on the request primitive, or is "no use of receipt confirmation".
  3. on the N_CONN_RES, the value is either equal to the value on the indication primitive or is "no use of receipt confirmation".
  4. on the N_CONN_CON, the value is equal to the value on the response primitive.
- Since the NS users and the NS provider must agree to the use of receipt confirmation selection, there are four possible cases of negotiation of receipt confirmation on an NC:
  1. if the source NS user does not request it -- it is not used;
  2. if the source NS user requests it but the NS provider does not provide it -- it is not used;
  3. if the source NS user requests it and the NS provider agrees to provide it, but the destination NS user does not agree to its use -- it is not used;
  4. if the source NS user requests it, the NS provider agrees to provide it, and the destination NS user agrees to its use -- it can be used.

Rules for Expedited Data Selection

- The expedited data selection parameter values on the various primitives are related such that:
  1. on the N_CONN_REQ, either of the defined values may occur, (namely "use of expedited data" or "no use of expedited data");
  2. on the N_CONN_IND, the value is either equal to the value on the request primitive, or is "no use of expedited data";
  3. on the N_CONN_RES, the value is either equal to the value on the indication primitive, or is "no use of expedited data";

4. on the N_CONN_CON, the value is equal to the value on the response primitive.

- Since the NS users and the NS provider must agree to the use of expedited data selection, there are four possible cases of negotiation of expedited data on an NC:
  1. if the source NS user does not request it -- it is not used;
  2. if the source NS user requests it but the NS provider does not provide it -- it is not used;
  3. if the source NS user requests it and the NS provider agrees to provide it, but the destination NS user does not agree to its use -- it is not used;
  4. if the source NS user requests it, the NS provider agrees to provide it, and the destination NS user agrees to its use -- it can be used.

7.2.2.1 N_CONN_REQ

Parameters

QOS_length: Indicates the length of the QOS parameters values that apply to the NC being requested.

The applicable QOS parameters are defined in the following structure:

i. N_QOS_CO_RANGE1

If the NS user cannot determine the value of a QOS parameter, its value should be set to QOS_UNKNOWN. If the NS user does not specify any QOS parameter values, the length of this field should be set to zero.

Flags

REC_CONF_OPT: The receipt confirmation selection parameter indicates whether receipt confirmation service is desired by the calling NS user on the NC. The receipt confirmation service must be provided in the network service to be used on the NC. When set, it indicates "use of receipt confirmation", and when not set it indicates "no use of receipt confirmation".

EX_DATA_OPT: The expedited data selection parameter indicates whether the expedited data service is desired by the calling NS user on the NC. The expedited data transfer service must be provided by the NS provider for it to be used on the NC. When set, it indicates "use of expedited data", and when not set it indicates "no use of expedited data".

7.2.2.2 N_CONN_IND

Parameters

QOS_length: Indicates the length of the QOS parameters values that are negotiated during NC establishment.

The applicable QOS parameters are defined in the following structure:

i. N_QOS_CO_RANGE1

If the NS provider does not support or cannot determine the value of a QOS parameter, its value will be set to QOS_UNKNOWN. If the NS provider does not specify any QOS parameter values, the length of this field should be set to zero.

QOS_offset: Indicates the offset of the QOS parameters from the beginning of the M_PROTO message block.

Flags

REC_CONF_OPT: The receipt confirmation selection parameter indicates whether the receipt confirmation service is available on the NC and the calling NS user desires its use. The receipt confirmation service must be provided in the network service to be used on the NC. When set, it indicates "use of receipt confirmation", and when not set, it indicates "no use of receipt confirmation". The value on the N_CONN_IND is either equal to the value on the request primitive or is "no use of receipt confirmation".

EX_DATA_OPT: The expedited data selection parameter indicates whether the expedited data transfer service is available on the NC and the calling NS user desires its use. The expedited data transfer service must be provided by the NS provider for it to be used on the NC. When set, it indicates "use of expedited data" or "no use of expedited data". The value on the N_CONN_IND is either equal to the value on the request primitive or is "no use of expedited data".

7.2.2.3 N_CONN_RES

Parameters

QOS_length: Indicates the length of the QOS parameters values that are negotiated during NC establishment. The applicable QOS parameters are defined in the following structure:

i. N_QOS_CO_SEL1

If the NS user does not agree to the QOS values, it will reject the NC establishment by invoking a N_DISCON_REQ primitive (the originator parameter in the N_DISCON_REQ primitive will indicate NS user invoked release). If the NS user cannot determine the value of a QOS parameter, its value should be set to QOS_UNKNOWN. If the NS user does not specify any QOS parameter values, the length of this field should be set to zero.

Flags

REC_CONF_OPT: The receipt confirmation selection parameter indicates whether the receipt confirmation service can be used on the NC. The receipt confirmation service must be provided in the network service to be used on the NC. When set, it indicates "use of receipt confirmation", and when not set it indicates "no use of receipt confirmation". The value on the N_CONN_RES is either equal to the value on the indication primitive or is "no use of receipt confirmation".

EX_DATA_OPT: The expedited data selection parameter indicates whether the expedited data transfer service can be used on the NC. The expedited data transfer service must be provided by the NS provider for it to be used on the NC. When set, it indicates "use of expedited data", and when not set, it indicates "no use of expedited data". The value on the N_CONN_RES is either equal to the value on the indication primitive or is "no use of expedited data".

7.2.2.4 N_CONN_CON

Parameters

QOS_length: Indicates the length of the QOS parameters values selected by the responding NS user. The applicable QOS parameters are defined in the following structure:

i. N_QOS_CO_SEL1

If the NS provider does not support or cannot determine the selected value of a QOS parameter, its value will be set to QOS_UNKNOWN. If the NS provider does not specify any QOS parameter values, the length of this field should be set to zero.

Flags

REC_CONF_OPT:	The receipt confirmation selection parameter indicates whether the receipt confirmation service can be used on the NC. The receipt confirmation service must be provided in the network service to be used on the NC. When set, it indicates "use of receipt confirmation", and when not set it indicates "no use of receipt confirmation". The value on the N_CONN_CON is equal to the value on the response primitive.
EX_DATA_OPT:	The expedited data selection parameter indicates whether the expedited data transfer service can be used on the NC. The expedited data transfer service must be provided by the NS provider for it to be used on the NC. When set, it indicates "use of expedited data", and when not set, it indicates "no use of expedited data". The value on the N_CONN_CON is equal to the value on the response primitive.

7.2.3 CONS: Reset Service

7.2.3.1 N_RESET_REQ

Parameters

RESET_reason: Gives information indicating the cause of the reset.

Rules governing the value of the RESET_reason parameter

For an N_RESET_REQ, the reason shall always indicate N_USER_RESYNC.

7.2.3.2 N_RESET_IND

Parameters

RESET_orig: This parameter indicates the source of the reset.

Reset Originator

N_PROVIDER:	NS provider originated reset
N_USER:	NS user originated reset
N_UNDEFINED:	reset originator undefined

RESET_reason: Gives information indicating the cause of the reset.

Rules governing the value of the RESET_reason parameter

The value conveyed in this parameter will be as follows:

i. when the originator parameter indicates an NS provider invoked reset; the parameter is one of:

N_CONGESTION:	reset due to congestion;
N_RESET_UNSPECIFIED:	reset-reason unspecified.

ii. when the originator parameter indicates an NS user invoked reset, the value is:

N_USER_RESYNC:	user resynchronization.

iii. when the originator parameter has the value "undefined", then the value of the reason parameter is:

N_REASON_UNDEFINED:	reset reason undefined

7.2.4 CONS: NC Release Phase

7.2.4.1 N_DISCON_REQ

Parameters:

DISCON_reason: Gives information about the cause of the release.

Rules governing the value of the DISCON_reason parameter

The value conveyed in the parameter will be as follows:

N_DISC_NORMAL:	"disconnection-normal condition"
N_DISC_ABNORMAL:	"disconnection-abnormal condition"
N_REJ_P	"connection rejection-permanent condition"
N_REJ_T:	"connection rejection-transient condition"
N_REJ_QOS_UNAVAIL_P:	"connection rejection-QOS not available/permanent condition"
N_REJ_QOS_UNAVAIL_T:	"connection rejection-QOS not available/transient condition"
N_REJ_INCOMPAT_INFO:	"connection rejection-incompatible information in NS user data"
N_REJ_UNSPECIFIED:	"connection rejection-reason unspecified"

7.2.4.2 N_DISCON_IND

Parameters

DISCON_orig: Indicates the source of the NC release. Its value are as follows:

N_PROVIDER:	NS provider originated disconnect
N_USER:	NS user originated disconnect

	N_UNDEFINED: disconnect originator undefined
	The value "undefined" is not permitted when an N_DISCON_IND is issued by an NS user or the NS provider in order to reject an NC establishment attempt.
DISCON_reason:	Gives information about the cause of the release.

Rules governing the value of the DISCON_reason parameter

The value conveyed in the parameter will be as follows:

a. When the originator parameter indicates an NS provider invoked release, the value is one of:

N_DISC_P:	"disconnection-permanent condition"
N_DISC_T:	"disconnection-transient condition"
N_REJ_NSAP_UNKNOWN:	"connection rejection-NSAP address unknown (permanent condition)"
N_REJ_NSAP_UNREACH_P:	"connection rejection-NSAP unreachable (permanent condition)"
N_REJ_NSAP_UNREACH_T:	"connection rejection-NSAP unreachable (transient condition)"
N_REJ_QOS_UNAVAIL_P:	"connection rejection-QOS not available/permanent condition"
N_REJ_QOS_UNAVAIL_T:	"connection rejection-QOS not available/transient condition"
N_REJ_UNSPECIFIED:	"connection rejection-reason unspecified"

b. When the originator parameter indicates an NS user invoked release, the value is one of:

N_DISC_NORMAL:	"disconnection-normal condition"
N_DISC_ABNORMAL:	"disconnection-abnormal condition"
N_REJ_P	"connection rejection-permanent condition"
N_REJ_T:	"connection rejection-transient condition"
N_REJ_QOS_UNAVAIL_P:	"connection rejection-QOS not available/permanent condition"
N_REJ_QOS_UNAVAIL_T:	"connection rejection-QOS not available/transient condition"
N_REJ_INCOMPAT_INFO:	"connection rejection-incompatible information in NS user data"

N_REJ_UNSPECIFIED: "connection rejection-reason unspecified"

c. When the originator parameter value is undefined, then the value of the reason parameter shall be:

N_REASON_UNDEFINED: disconnect reason undefined

7.2.5 CLNS

7.2.5.1 N_UDERROR_IND

Parameters

ERROR_type: Specifies the reason for the error. The possible values are:

N_UD_UNDEFINED:	*no reason specified;*
N_UD_TD_EXCEEDED:	*transit delay exceeded;*
N_UD_CONGESTION:	*NS provider congestion;*
N_UD_QOS_UNAVAIL:	*other requested QOS/service characteristic unavailable;*
N_UD_LIFE_EXCEEDED:	*NSDU lifetime exceeded;*
N_UD_ROUTE_UNAVAIL:	*suitable route unavailable.*

Appendix A. Mapping NPI Primitives to ISO 8348 and CCITT X.213

Table A-1 shows a mapping of the NPI primitives to the OSI network service definition primitives.

NETWORK PRIMITIVE	STREAM MESSAGE TYPE	OSI NETWORK PRIMITIVE
N_CONN_REQ	M_PROTO	N-CONNECT request
N_CONN_IND	M_PROTO	N-CONNECT indication
N_CONN_RES	M_PROTO	N-CONNECT response
N_CONN_CON	M_PROTO	N-CONNECT confirm
N_DATA_REQ	M_PROTO	N-DATA request
N_DATA_IND	M_PROTO	N-DATA indication
N_EXDATA_REQ	M_PROTO	N-EXPEDITED-DATA request
N_EXDATA_IND	M_PROTO	N-EXPEDITED-DATA indication
N_DATACK_REQ	M_PROTO	N-DATA-ACKNOWLEDGE request
N_DATACK_IND	M_PROTO	N-DATA-ACKNOWLEDGE indication
N_RESET_REQ	M_PROTO	N-RESET request
N_RESET_IND	M_PROTO	N-RESET indication
N_RESET_RES	M_PROTO	N-RESET response
N_RESET_CON	M_PROTO	N-RESET confirm
N_DISCON_REQ	M_PROTO	N_DISCONNECT request
N_DISCON_IND	M_PROTO	N-DISCONNECT indication
N_UNITDATA_REQ	M_PROTO	N-UNITDATA request
N_UNITDATA_IND	M_PROTO	N-UNITDATA indication
N_BIND_REQ	M_PROTO	‡
N_BIND_ACK	M_PCPROTO	‡
N_UNBIND_REQ	M_PROTO	‡
N_OK_ACK	M_PCPROTO	‡
N_ERROR_ACK	M_PCPROTO	‡
N_INFO_REQ	M_PCPROTO	‡
N_INFO_ACK	M_PCPROTO	‡
N_UDERROR_IND	M_PROTO	‡
N_OPTMGMT_REQ	M_PROTO	‡

‡ — Local management issue, not defined in ISO 8348 and CCITTX.213

Table A-1. Mapping NPI Primitives to OSI NS

Appendix B. State/Event Tables

This appendix contains tables showing the network-user's view of the possible states that the NPI may enter due to an event, and the possible events that may occur on the interface.

The N_INFO_REQ, N_INFO_ACK, N_TOKEN_REQ, and N_TOKEN_ACK primitives are excluded from the state transition table because they can be issued from several states, and secon- dly, they do not cause a state transition to occur. However, the N_INFO_REQ and the N_TOKEN_REQ primitives may not be issued by the NS user when a local acknowledgement to a previously issued primitive is pending.

STATE	ABBREVIATION	DESCRIPTION	SERVICE TYPE
sta_0	unbnd	unbound	CONS, CLNS
sta_1	w_ack, b_req	awaiting acknowledgement of N_BIND_REQ	CONS, CLNS
sta_2	w_ack, u_req	awaiting acknowledgement of N_UNBIND_REQ	CONS, CLNS
sta_3	idle	idle-no connection	CONS, CLNS
sta_4	w_ack, op_req	awaiting acknowledgement of N_OPTMGMT_REQ	CONS, CLNS
sta_5	w_ack, r_res	awaiting acknowledgement of N_RESET_RES	CONS
sta_6	w_con, c_req	awaiting confirmation of N_CONN_REQ	CONS
sta_7	w_res, c_ind	awaiting response of N_CONN_IND	CONS
sta_8	w_ack, c_res	awaiting acknowledgement of N_CONN_RES	CONS
sta_9	data_t	data transfer	CONS
sta_10	w_con, r_req	awaiting confirmation of N_RESET_REQ	CONS
sta_11	w_res, r_ind	awaiting response of N_RESET_IND	CONS
sta_12	w_ack, dreq6	awaiting acknowledgement of N_DISCON_REQ	CONS
sta_13	w_ack, dreq7	awaiting acknowledgement of N_DISCON_REQ	CONS
sta_14	w_ack, dreq9	awaiting acknowledgement of N_DISCON_REQ	CONS
sta_15	w_ack, dreq10	awaiting acknowledgement of N_DISCON_REQ	CONS
sta_16	w_ack, dreq11	awaiting acknowledgement of N_DISCON_REQ	CONS

Table B-1. Kernel Level NPI States

Tables B-2 and B-3 describe the variables and outputs used in the state tables.

VARIABLE	DESCRIPTION
token	is a value contained in the N_CONN_RES primitive that is used to identify the stream on which the NC is to be established. When its value is zero, it indicates that the NC is to be established on the stream on which the N_CONN_IND arrived. When its value is non-zero, it identifies another stream on which the NS provider is to accept the NC.
outcnt	counter for the number of outstanding connection indications not responded to by the network user entity.

Table B-2. State Table Variables

OUTPUT	DESCRIPTION
[1]	outcnt = 0
[2]	outcnt = outcnt + 1
[3]	outcnt = outcnt - 1
[4]	pass connection to stream as indicated by the token in the N_CONN_RES primitive.

Table B-3. State Table Outputs

Table B-4 shows outgoing events that are initiated by the network-user entity. These events are either requests to the network provider or responses to an event of the network provider.

EVENT	DESCRIPTION	SERVICE TYPE
info_req	information request	CONS, CLNS
bind_req	bind request	CONS, CLNS
unbind_req	unbind request	CONS, CLNS
optmgmt_req	options mgmt request	CONS, CLNS
conn_req	connection request	CONS
conn_res	connection response	CONS
discon_req	disconnect request	CONS
data_req	data request	CONS
exdata_req	expedited data request	CONS
datack_req	data ack request	CONS
reset_req	reset request	CONS
reset_res	reset response	CONS
unitdata_req	unitdata request	CLNS

Table B-4. Kernel Level NPI Outgoing Events

Table B-5 shows incoming events that are initiated by the network provider. These events are either confirmations of a request, or are indications to the NS user entity that an event has occurred.

EVENT	DESCRIPTION	SERVICE TYPE
info_ack	information acknowledgement	CONS, CLNS
bind_ack	bind acknowledgement	CONS, CLNS
error_ack	error acknowledgement	CONS, CLNS
ok_ack1	ok acknowledgement outcnt == 0	CONS, CLNS
ok_ack2	ok acknowledgement outcnt == 1, token == 0	CONS, CLNS
ok_ack3	ok acknowledgement outcnt == 1, token! == 0	CONS, CLNS
ok_ack4	ok acknowledgement outcnt>1, token! == 0	CONS, CLNS
conn_ind	connection indication	CONS
conn_conf	connection confirm	CONS
discon_ind1	disconnect indication outcnt == 0	CONS
discon_ind2	disconnect indication outcnt == 1	CONS
discon_ind3	disconnect indication outcnt > 1	CONS
data_ind	data indication	CONS
exdata_ind	expedited data indication	CONS
datack_ind	data ack indication	CONS
reset_ind	reset indication	CONS
reset_conf	reset confirm	CONS
pass_conn	pass connection	CONS
unitdata_ind	unitdata indication	CLNS
uderror_ind	unitdata error indication	CLNS

Table B-5. Kernel Level NPI Incoming Events

Tables B-6 and B-7 describe the possible events the NPI may enter given a current state and event. The contents of each box represent the next state given the current state (column) and the current incoming or outgoing event (row). An empty box represents a state/event combination that is invalid. Along with the next state, each box may include an action. The network provider must take specific actions in the order specified in the state table.

STATE EVENT	sta_3
unitdata_req	sta_3
unitdata_ind	sta_3
uderror_ind	sta_3

Table B-6. Data Transfer State Table for CLNS

STATE EVENT	sta_0	sta_1	sta_2	sta_3	sta_4
bind_req	sta_1				
unbind_req				sta_2	
optmgmt_req				sta_4	
bind_ack		sta_3 [1]			
error_ack		sta_0	sta_3		sta_3
ok_ack1			sta_0		sta_3

Table B-7. Initialization State Table for CONS

STATE EVENT	3	5	6	7	8	9	10	11	12	13	14	15	16
conn_req	6												
conn_ind	7 [2]			7 [2]									
conn_res				8									
conn_con			9										
discon_req			12	13		14	15	16					
data_req						9							
exdata_req						9							
reset_req						10							
reset_res								5					
data_ind						9							
exdata_ind						9							
datack_req						9							
datack_ind						9							
reset_ind						11							
reset_con							9						
discon_ind1			3			3	3	3					
discon_ind2				3 [3]									
discon_ind3				7 [3]									
error_ack		11	3		7		9		6	7	9	10	11
ok_ack1		9							3		3	3	3
ok_ack2					9 [3]					3 [3]			
ok_ack3					3 [3,4]					3 [3]			
ok_ack4					7 [3,4]					7 [3]			
pass_conn	9												

NOTE: The column headings and box entries are state (sta_) numbers as described in Table B-1. Refer to Table B-3 for the interpretation of the bracketed numbers.

Table B-8. State Table for CONS for Connection/Release/Data Transfer States

Appendix C. Primitive Precedence Tables

Tables C-1 and C-2 describe the precedence of the NPI primitives for both the *stream* write and read queues. In both these tables, primitive Y is already on the queue and primitive X is about to be put on the queue. The *stream* write queue contains network user initiated primitives and the *stream* read queue contains network provider initiated primitives. The column headings are a shorthand notation for the row headings.

PRIM X PRIM Y on queue	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13
R1 n_conn_req			4										
R2 n_conn_res			3	1									
R3 n_discon_req	1												
R4 n_data_req			5	1	2						3		2
R5 n_exdata_req			5	1	1						3		2
R6 n_bind_req													
R7 n_unbind_req													
R8 n_info_req													
R9 n_unitdata_req									1				
R10 n_optmgmt_req													
R11 n_reset_req			3										
R12 n_reset_res			3	1							1		
R13 n_datack_req			5	1	2						3		1

Blank *not applicable - queue should be empty.*

1 *X has no precedence over Y.*

2 *X has precedence over Y.*

3 *X has precedence over Y and Y must be removed.*

4 *X has precedence over Y and both X and Y must be removed.*

5 *X may have precedence over Y (choice of user). It it does then it is the same as 3.*

Table C-1. ***STREAM*** **Write Queue Precedence Table**

PRIM X PRIM Y on queue	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14
I1 n_conn_ind			4			2								
I2 n_conn_con			3	1	1	2								
I3 n_discon_ind	1					2		2	2					
I4 n_data_ind			5	1	2	2			1			3		2
I5 n_exdata_ind			5	1	1				1			3		2
I6 n_info_ack	1	1	1	1						1	1	1	1	
I7 n_bind_ack	1													
I8 n_error_ack	1	1	1	1	1							1	1	1
I9 n_ok_ack	1	1	1	1	1							1	1	1
I10 n_unitdata_ind						2		2		1	2			
I11 n_uderror_ind						2		1		1	1			
I12 n_reset_ind			3			2								
I13 n_reset_con			3			2								
I14 n_datack_ind			5		2							1		

Blank *not applicable - queue should be empty.*

1 *X has no precedence over Y.*

2 *X has precedence over Y.*

3 *X has precedence over Y and Y must be removed.*

4 *X has precedence over Y and both X and Y must be removed.*

5 *X may have precedence over Y (choice of user). If it does then it is the same as 3.*

Table C-2. ***STREAM*** **Read Queue Precedence Table**

Appendix D. NPI Header File Listing

This appendix contains a listing of the NPI header file needed by implementations.

```
/*
 * npi.h header for the Network Provider Interface (OSI Conforming)
 */

#define N_CURRENT_VERSION       0x02            /* current version of NPI */
#define N_VERSION_2             0x02            /* version of npi, December
                                                 * 16, 1991 */
/*
 * Primitives that are initiated by the network user.
 */

#define N_CONN_REQ       0                      /* NC request */
#define N_CONN_RES       1                      /* Accept previous
                                                 * connection indication */
#define N_DISCON_REQ     2                      /* NC disconnection request */
#define N_DATA_REQ       3                      /* Connection-Mode data
                                                 * transfer request */
#define N_EXDATA_REQ     4                      /* Expedited data request */
#define N_INFO_REQ       5                      /* Information Request */
#define N_BIND_REQ       6                      /* Bind a NS user to network
                                                 * address */
#define N_UNBIND_REQ     7                      /* Unbind NS user from
                                                 * network address */
#define N_UNITDATA_REQ   8                      /* Connection-less data send
                                                 * request */
#define N_OPTMGMT_REQ    9                      /* Options Management
                                                 * request */
/*
 * Primitives that are initiated by the network provider.
 */
#define N_CONN_IND      11                      /* Incoming connection
                                                 * indication */
#define N_CONN_CON      12                      /* Connection established */
#define N_DISCON_IND    13                      /* NC disconnected */
#define N_DATA_IND      14                      /* Incoming connection-mode
                                                 * data indication */
#define N_EXDATA_IND    15                      /* Incoming expedited data
                                                 * indication */
#define N_INFO_ACK      16                      /* Information
                                                 * Acknowledgement */
#define N_BIND_ACK      17                      /* NS User bound to network
                                                 * address */
#define N_ERROR_ACK     18                      /* Error Acknowledgement */
#define N_OK_ACK        19                      /* Success Acknowledgement */
#define N_UNITDATA_IND  20                      /* Connection-less data
                                                 * receive indication */
#define N_UDERROR_IND   21                      /* UNITDATA Error Indication */
/*
```

```
 * Additional NPI Primitivies
 */

#define N_DATACK_REQ    23                        /* Data acknowledgement
                                                   * request */
#define N_DATACK_IND    24                        /* Data acknowledgement
                                                   * indication */
#define N_RESET_REQ     25                        /* NC reset request */
#define N_RESET_IND     26                        /* Incoming NC reset request
                                                   * indication */
#define N_RESET_RES     27                        /* Reset processing accepted */
#define N_RESET_CON     28                        /* Reset processing complete */




/*
 * The following are the events that drive the state machine
 */
/* Initialization events */
#define NE_BIND_REQ     0                         /* bind request */
#define NE_UNBIND_REQ   1                         /* unbind request */
#define NE_OPTMGMT_REQ  2                         /* manage options request */
#define NE_BIND_ACK     3                         /* bind acknowledgement */
#define NE_ERROR_ACK    5                         /* error acknowledgement */
#define NE_OK_ACK1      6                         /* ok ack, outcnt == 0 */
#define NE_OK_ACK2      7                         /* ok ack, outcnt == 1, q ==
                                                   * rq */
#define NE_OK_ACK3      8                         /* ok ack, outcnt == 1, q!
                                                   * == rq */
#define NE_OK_ACK4      9                         /* ok ack, outcnt > 1 */

/* Connection-Mode events */
#define NE_CONN_REQ     10                        /* connect request */
#define NE_CONN_RES     11                        /* connect response */
#define NE_DISCON_REQ   12                        /* disconnect request */
#define NE_DATA_REQ     13                        /* data request */
#define NE_EXDATA_REQ   14                        /* expedited data request */
#define NE_CONN_IND     16                        /* connect indication */
#define NE_CONN_CON     17                        /* connect confirm */
#define NE_DATA_IND     18                        /* data indication */
#define NE_EXDATA_IND   19                        /* expedited data indication */
#define NE_DISCON_IND1  21                        /* disconnect indication,
                                                   * outcnt == 0 */
#define NE_DISCON_IND2  22                        /* disconnect indication,
                                                   * outcnt == 1 */
#define NE_DISCON_IND3  23                        /* disconnect indication,
                                                   * outcnt > 1 */
#define NE_PASS_CON     24                        /* pass connection */
#define NE_RESET_REQ    28                        /* reset request */
#define NE_RESET_RES    29                        /* reset response */
#define NE_DATACK_REQ   30                        /* data acknowledgement
```

```
                                                 * request */
#define NE_DATACK_IND   31                      /* data acknowledgement
                                                 * indication */
#define NE_RESET_IND    32                      /* reset indication */
#define NE_RESET_CON    33                      /* reset confirm */

/* Connection-less events */
#define NE_UNITDATA_REQ 25                      /* unitdata request */
#define NE_UNITDATA_IND 26                      /* unitdata indication */
#define NE_UDERROR_IND  27                      /* unitdata error indication */

#define NE_NOEVENTS     36                      /* no events */


/*
 * NPI interface states
 */
#define NS_UNBND        0                       /* NS user not bound to
                                                 * network address */
#define NS_WACK_BREQ    1                       /* Awaiting acknowledgement
                                                 * of N_BIND_REQ */
#define NS_WACK_UREQ    2                       /* Pending acknowledgement
                                                 * for N_UNBIND_REQ */
#define NS_IDLE         3                       /* Idle, no connection */
#define NS_WACK_OPTREQ  4                       /* Pending acknowledgement
                                                 * of N_OPTMGMT_REQ */
#define NS_WACK_RRES    5                       /* Pending acknowledgement
                                                 * of N_RESET_RES */
#define NS_WCON_CREQ    6                       /* Pending confirmation of
                                                 * N_CONN_REQ */
#define NS_WRES_CIND    7                       /* Pending response of
                                                 * N_CONN_REQ */
#define NS_WACK_CRES    8                       /* Pending acknowledgement
                                                 * of N_CONN_RES */
#define NS_DATA_XFER    9                       /* Connection-mode data
                                                 * transfer */
#define NS_WCON_RREQ    10                      /* Pending confirmation of
                                                 * N_RESET_REQ */
#define NS_WRES_RIND    11                      /* Pending response of
                                                 * N_RESET_IND */
#define NS_WACK_DREQ6   12                      /* Waiting ack of
                                                 * N_DISCON_REQ */
#define NS_WACK_DREQ7   13                      /* Waiting ack of
                                                 * N_DISCON_REQ */
#define NS_WACK_DREQ9   14                      /* Waiting ack of
                                                 * N_DISCON_REQ */
#define NS_WACK_DREQ10  15                      /* Waiting ack of
                                                 * N_DISCON_REQ */
#define NS_WACK_DREQ11  16                      /* Waiting ack of
                                                 * N_DISCON_REQ */

#define NS_NOSTATES     18                      /* No states */
```

```
/*
 * N_ERROR_ACK error return code values
 */
#define NBADADDR         1                        /* Incorrect address
                                                   * format/illegal address
                                                   * information */
#define NBADOPT          2                        /* Options in incorrect
                                                   * format or contain illegal
                                                   * information */
#define NACCESS          3                        /* User did not have proper
                                                   * permissions */
#define NNOADDR          5                        /* NS Provider could not
                                                   * allocate address */
#define NOUTSTATE        6                        /* Primitive was issues in
                                                   * wrong sequence */
#define NBADSEQ          7                        /* Sequence number in
                                                   * primitive was
                                                   * incorrect/illegal */
#define NSYSERR          8                        /* UNIX system error
                                                   * occurred */
#define NBADDATA        10                        /* User data spec. outside
                                                   * range supported by NS
                                                   * provider */
#define NBADFLAG        16                        /* Flags specified in
                                                   * primitive were
                                                   * illegal/incorrect */
#define NNOTSUPPORT     18                        /* Primitive type not
                                                   * supported by the NS
                                                   * provider */
#define NBOUND          19                        /* Illegal second attempt to
                                                   * bind listener or default
                                                   * listener */
#define NBADQOSPARAM    20                        /* QOS values specified are
                                                   * outside the range
                                                   * supported by the NS
                                                   * provider */
#define NBADQOSTYPE     21                        /* QOS structure type
                                                   * specified is not
                                                   * supported by the NS
                                                   * provider */
#define NBADTOKEN       22                        /* Token used is not
                                                   * associated with an open
                                                   * stream */
#define NNOPROTOID      23                        /* Protocol id could not be
                                                   * allocated */


/*
 * N_UDERROR_IND reason codes
 */
#define N_UD_UNDEFINED          10                /* no reason specified */
#define N_UD_TD_EXCEEDED        11                /* Transit delay exceeded */
#define N_UD_CONGESTION         12                /* NS Provider congestion */
```

```
#define N_UD_QOS_UNAVAIL        13              /* Requested QOS/service
                                                 * characteristic
                                                 * unavailable */
#define N_UD_LIFE_EXCEEDED      14              /* NSDU Lifetime exceeded */
#define N_UD_ROUTE_UNAVAIL      15              /* Suitable route
                                                 * unavailable */
#define N_UD_SEG_REQUIRED       16              /* Segmentation reqd where
                                                 * none permitted */


/*
 * NPI Originator for Resets and Disconnects
 */
#define N_PROVIDER      0x0100                  /* provider originated
                                                 * reset/disconnect */
#define N_USER          0x0101                  /* user originated
                                                 * reset/disconnect */
#define N_UNDEFINED     0x0102                  /* reset/disconnect
                                                 * originator undefined */


/*
 * NPI Disconnect & Reset reasons when the originator is the N_UNDEFINED
 */
#define N_REASON_UNDEFINED      0x0200


/*
 * NPI Disconnect reasons when the originator is the N_PROVIDER
 */
#define N_DISC_P                0x0300          /* Disconnection-permanen t
                                                 * condition */
#define N_DISC_T                0x0301          /* Disconnection-transien t
                                                 * condition */
#define N_REJ_NSAP_UNKNOWN      0x0302          /* Connection rejection-NSAP
                                                 * address unknown
                                                 * (permanent condition) */
#define N_REJ_NSAP_UNREACH_P    0x0303          /* Connection rejection-NSAP
                                                 * unreachable (permanent
                                                 * condition) */
#define N_REJ_NSAP_UNREACH_T    0x0304          /* Connection rejection-NSAP
                                                 * unreachable (transient
                                                 * condition) */


/*
 * NPI Disconnect reasons when the originator is the N_USER
 */
#define N_DISC_NORMAL           0x0400          /* Disconnection-normal
                                                 * condition */
#define N_DISC_ABNORMAL         0x0401          /* Disconnection-abnormal
                                                 * condition */
```

```
#define N_REJ_P                 0x0402          /* Connection
                                                 * rejection-permanent
                                                 * condition */
#define N_REJ_T                 0x0403          /* Connection
                                                 * rejection-transient
                                                 * condition */
#define N_REJ_INCOMPAT_INFO     0x0406          /* Connection
                                                 * rejection-incompatible
                                                 * information in
                                                 * NS-user-data */


/*
 * NPI Disconnect reasons when the originator is the N_USER or N_PROVIDER
 */
#define N_REJ_QOS_UNAVAIL_P     0x0305          /* Connection rejection-QOS
                                                 * unavailable (permanent
                                                 * condition) */
#define N_REJ_QOS_UNAVAIL_T     0x0306          /* Connection rejection-QOS
                                                 * unavailable (transient
                                                 * condition) */
#define N_REJ_UNSPECIFIED       0x0307          /* Connection
                                                 * rejection-reason
                                                 * unspecified */


/*
 * NPI Reset reasons when originator is N_PROVIDER
 */
#define N_CONGESTION            0x0500          /* Reset due to congestion */
#define N_RESET_UNSPECIFIED     0x0501          /* Reset-reason
                                                 * "unspecified" */


/*
 * NPI Reset reasons when originator is N_USER
 */
#define N_USER_RESYNC           0x0600          /* Reset due to user
                                                 * resynchronization */


/*
 * CONN_flags definition; (used in N_conn_req, N_conn_ind, N_conn_res, and
 * N_conn_con primitives)
 *
 * Flags to indicate support of network provider options; (used with the
 * OPTIONS_flags field of N_info_ack primitive)
 */

#define REC_CONF_OPT            0x00000001L     /* Receipt Confirmation
                                                 * Selection and Support */
#define EX_DATA_OPT             0x00000002L     /* Expedited Data Selection
                                                 * and Support */
```

```
/* This flag is used with the OPTIONS_flags field of N_info_ack as well as */
/* the OPTMGMT_flags field of the N_optmgmt_req primitive */

#define DEFAULT_RC_SEL  0x00000003L             /* Indicates if default
                                                 * receipt confirmation is
                                                 * selected */



/*
 * BIND_flags; (used with N_bind_req primitive)
 */

#define DEFAULT_LISTENER  0x00000001L           /* indicates if this stream
                                                 * is the default listener */
#define TOKEN_REQUEST     0x00000002L           /* indicates if "token"
                                                 * should be assigned to
                                                 * thestream */
#define DEFAULT_DEST      0x00000004L           /* indicates if default
                                                 * dest. stream */




/*
 * QOS Parameter Definitions
 */



/*
 * Throughput
 *
 * This parameter is specified for both directions.
 */

typedef struct {
        long            thru_targ_value;        /* target throughput values */
        long            thru_min_value;         /* minimum acceptable
                                                 * throughput value */
}               thru_values_t;


/*
 * Transit Delay
 */

typedef struct {
        long            td_targ_value;          /* target transit delay */
        long            td_max_value;           /* maximum acceptable
                                                 * transit delay */
}               td_values_t;
```

```
/*
 * Protection Values
 */

typedef struct {
        long            protect_targ_value;     /* target protection value */
        long            protect_min_value;      /* minimum or available
                                                 * protection */
}               protection_values_t;



/*
 * Priority Values
 */

typedef struct {
        long            priority_targ_value;    /* target priority */
        long            priority_min_value;     /* minimum acceptable
                                                 * priority */
}               priority_values_t;



/*
 * Types of protection specifications
 */
#define    N_NO_PROT                    0x00000000L     /* no protection */
#define    N_PASSIVE_PROT               0x00000001L     /* protection against
                                                         * passive monitoring */
#define    N_ACTIVE_PROT                0x00000002L     /* protection against
                                                         * active monitoring */
#define    N_ACTIVE_PASSIVE_PROT        0x00000003L     /* protection against
                                                         * active and passive
                                                         * monitoring */


/*
 * Cost Selection
 */

#define    N_LEAST_EXPENSIVE    0x00000000L      /* choose least expensive
                                                  * means */



/*
 * QOS STRUCTURE TYPES AND DEFINED VALUES
 */


#define N_QOS_CO_RANGE1         0x0101
#define N_QOS_CO_SEL1           0x0102
```

```
#define N_QOS_CL_RANGE1         0x0103
#define N_QOS_CL_SEL1           0x0104
#define N_QOS_CO_OPT_RANGE1     0x0105
#define N_QOS_CO_OPT_SEL1       0x0106


/*
 * When a NS user/provider cannot determine the value of a QOS field, it
 * should return a value of QOS_UNKNOWN.
 */

#define QOS_UNKNOWN             -1



/*
 * QOS range for CONS.  (Used with N_CONN_REQ and N_CONN_IND.)
 */
typedef struct {
        ulong           n_qos_type;              /* always N_QOS_CO_RANGE */
        thru_values_t   src_throughput_range;    /* source throughput range */
        thru_values_t   dest_throughput_range;   /* destination throughput
                                                  * range */
        td_values_t     transit_delay_range;     /* transit delay range */
        protection_values_t protection_range;    /* protection range */
        priority_values_t priority_range;        /* priority range */
}               N_qos_co_range_t;


/*
 * QOS selected for CONS. (Used with N_CONN_RES and N_CONN_CON.)
 */

typedef struct {
        ulong           n_qos_type;              /* always N_QOS_CO_SEL */
        long            src_throughput_sel;      /* source throughput
                                                  * selected */
        long            dest_throughput_sel;     /* destination throughput
                                                  * selected */
        long            transit_delay_sel;       /* transit delay selected */
        long            protection_sel;          /* NC protection selected */
        long            priority_sel;            /* NC priority selected */
}               N_qos_co_sel_t;


/*
 * QOS range for CLNS options management. (Used with N_INFO_ACK.)
 */
typedef struct {
        ulong           n_qos_type;              /* always N_QOS_CL_RANGE */
        td_values_t     transit_delay_max;       /* maximum transit delay */
        ulong           residual_error_rate;     /* residual error rate */
        protection_values_t protection_range;    /* protection range */
```

```
        priority_values_t priority_range;        /* priority range */
        long            max_accept_cost;         /* maximum acceptable cost */
}               N_qos_cl_range_t;


/*
 * QOS selection for CLNS options management. (Used with N_OPTMGMT_REQ and
 * N_INFO_ACK.)
 */
typedef struct {
        ulong           n_qos_type;              /* always N_QOS_CL_sel */
        long            transit_delay_max;       /* maximum transit delay */
        ulong           residual_error_rate;     /* residual error rate */
        long            protection_sel;          /* protection selected */
        long            priority_sel;            /* priority selected */
        long            max_accept_cost;         /* maximum acceptable cost */
}               N_qos_cl_sel_t;


/*
 * QOS range for CONS options management. (Used with N_OPTMGMT_REQ.)
 */
typedef struct {
        ulong           n_qos_type;              /* always N_QOS_CO_OPT_RANGE */
        thru_values_t   src_throughput;          /* source throughput values */
        thru_values_t   dest_throughput;         /* dest throughput values */
        td_values_t     transit_delay_t;         /* transit delay values */
        long            nc_estab_delay;          /* NC establishment delay */
        ulong           nc_estab_fail_prob;      /* NC estab failure
                                                  * probability */
        ulong           residual_error_rate;     /* residual error rate */
        ulong           xfer_fail_prob;          /* transfer failure
                                                  * probability */
        ulong           nc_resilience;           /* NC resilience */
        long            nc_rel_delay;            /* NC release delay */
        ulong           nc_rel_fail_prob;        /* NC release failure
                                                  * probability */
        protection_values_t protection_range;    /* protection range */
        priority_values_t priority_range;        /* priority range */
        long            max_accept_cost;         /* maximum acceptable cost */
}               N_qos_co_opt_range_t;


/*
 * QOS values selected for CONS options management. (Used with N_OPTMGMT_REQ
 * and N_INFO_ACK.)
 */
typedef struct {
        ulong           n_qos_type;              /* always N_QOS_CO_OPT_SEL */
        thru_values_t   src_throughput;          /* source throughput values */
        thru_values_t   dest_throughput;         /* dest throughput values */
        td_values_t     transit_delay_t;         /* transit delay values */
        long            nc_estab_delay;          /* NC establishment delay */
```

```
        ulong           nc_estab_fail_prob;       /* NC estab failure
                                                   * probability */
        ulong           residual_error_rate;      /* residual error rate */
        ulong           xfer_fail_prob;           /* transfer failure
                                                   * probability */
        ulong           nc_resilience;            /* NC resilience */
        long            nc_rel_delay;             /* NC release delay */
        ulong           nc_rel_fail_prob;         /* NC release failure
                                                   * probability */
        long            protection_sel;           /* protection selected */
        long            priority_sel;             /* priority selected */
        long            max_accept_cost;          /* maximum acceptable cost */
}               N_qos_co_opt_sel_t;



/*
 * NPI Primitive Definitions
 */



/*
 * Local management service primitives
 */


/*
 * Information request
 */

typedef struct {
        ulong           PRIM_type;                /* always N_INFO_REQ */
}               N_info_req_t;


/*
 * Information acknowledgement
 */

typedef struct {
        ulong           PRIM_type;                /* always N_INFO_ACK */
        ulong           NSDU_size;                /* maximum NSDU size */
        ulong           ENSDU_size;               /* maximum ENSDU size */
        ulong           CDATA_size;               /* connect data size */
        ulong           DDATA_size;               /* discon data size */
        ulong           ADDR_size;                /* address size */
        ulong           ADDR_length;              /* address length */
        ulong           ADDR_offset;              /* address offset */
        ulong           QOS_length;               /* QOS values length */
        ulong           QOS_offset;               /* QOS values offset */
        ulong           QOS_range_length;         /* length of QOS values'
                                                   * range */
```

```
        ulong           QOS_range_offset;         /* offset of QOS values'
                                                   * range */
        ulong           OPTIONS_flags;            /* bit masking for options
                                                   * supported */
        ulong           NIDU_size;                /* network i/f data unit
                                                   * size */
        long            SERV_type;                /* service type */
        ulong           CURRENT_state;            /* current state */
        ulong           PROVIDER_type;            /* type of NS provider */
        ulong           NODU_size;                /* optimal NSDU size */
        ulong           PROTOID_length;           /* length of bound protocol
                                                   * ids */
        ulong           PROTOID_offset;           /* offset of bound protocol
                                                   * ids */
        ulong           NPI_version;              /* version # of npi that is
                                                   * supported */
}               N_info_ack_t;


/*
 * Service types supported by NS provider
 */
#define N_CONS  1                                 /* Connection-mode network
                                                   * service supported */
#define N_CLNS  2                                 /* Connection-less network
                                                   * service supported */



/*
 * Valid provider types
 */
#define N_SNICFP 1
#define N_SUBNET 2
/*
 * Bind request
 */

typedef struct {
        ulong           PRIM_type;                /* always N_BIND_REQ */
        ulong           ADDR_length;              /* length of address */
        ulong           ADDR_offset;              /* offset of address */
        ulong           CONIND_number;            /* requested # of connect- */
        /* indications to be queued */
        ulong           BIND_flags;               /* bind flags */
        ulong           PROTOID_length;           /* length of bound protocol
                                                   * ids */
        ulong           PROTOID_offset;           /* offset of bound protocol
                                                   * ids */
}               N_bind_req_t;


/*
 * Bind acknowledgement
```

```
 */

typedef struct {
        ulong           PRIM_type;              /* always N_BIND_ACK */
        ulong           ADDR_length;            /* address length */
        ulong           ADDR_offset;            /* offset of address */
        ulong           CONIND_number;          /* connection indications */
        ulong           TOKEN_value;            /* value of "token" assigned
                                                 * to stream */
        ulong           PROTOID_length;         /* length of bound protocol
                                                 * ids */
        ulong           PROTOID_offset;         /* offset of bound protocol
                                                 * ids */
}               N_bind_ack_t;


/*
 * Unbind request
 */

typedef struct {
        ulong           PRIM_type;              /* always N_UNBIND_REQ */
}               N_unbind_req_t;


/*
 * Options management request
 */

typedef struct {
        ulong           PRIM_type;              /* always N_OPTMGMT_REQ */
        ulong           QOS_length;             /* length of QOS parameter
                                                 * values */
        ulong           QOS_offset;             /* offset of QOS parameter
                                                 * values */
        ulong           OPTMGMT_flags;          /* options management flags */
}               N_optmgmt_req_t;



/*
 * Error acknowledgement for CONS services
 */

typedef struct {
        ulong           PRIM_type;              /* always N_ERROR_ACK */
        ulong           ERROR_prim;             /* primitive in error */
        ulong           NPI_error;              /* NPI error code */
        ulong           UNIX_error;             /* UNIX error code */
}               N_error_ack_t;
```

```
/*
 * Successful completion acknowledgement
 */

typedef struct {
        ulong           PRIM_type;              /* always N_OK_ACK */
        ulong           CORRECT_prim;           /* primitive being
                                                 * acknowledged */
}               N_ok_ack_t;



/*
 * CONS PRIMITIVES
 */


/*
 * Network connection request
 */

typedef struct {
        ulong           PRIM_type;              /* always N_CONN_REQ */
        ulong           DEST_length;            /* destination address
                                                 * length */
        ulong           DEST_offset;            /* destination address
                                                 * offset */
        ulong           CONN_flags;             /* bit masking for options
                                                 * flags  */
        ulong           QOS_length;             /* length of QOS parameter
                                                 * values */
        ulong           QOS_offset;             /* offset of QOS parameter
                                                 * values */
}               N_conn_req_t;


/*
 * Connection indication
 */

typedef struct {
        ulong           PRIM_type;              /* always N_CONN_IND */
        ulong           DEST_length;            /* destination address
                                                 * length */
        ulong           DEST_offset;            /* destination address
                                                 * offset */
        ulong           SRC_length;             /* source address length */
        ulong           SRC_offset;             /* source address offset */
        ulong           SEQ_number;             /* sequence number */
        ulong           CONN_flags;             /* bit masking for options
                                                 * flags */
        ulong           QOS_length;             /* length of QOS parameter
                                                 * values */
```

```
        ulong           QOS_offset;             /* offset of QOS parameter
                                                 * values */
}               N_conn_ind_t;



/*
 * Connection response
 */

typedef struct {
        ulong           PRIM_type;              /* always N_CONN_RES */
        ulong           TOKEN_value;            /* NC response token value */
        ulong           RES_length;             /* responding address length */
        ulong           RES_offset;             /* responding address offset */
        ulong           SEQ_number;             /* sequence number */
        ulong           CONN_flags;             /* bit masking for options
                                                 * flags */
        ulong           QOS_length;             /* length of QOS parameter
                                                 * values */
        ulong           QOS_offset;             /* offset of QOS parameter
                                                 * values */
}               N_conn_res_t;



/*
 * Connection confirmation
 */

typedef struct {
        ulong           PRIM_type;              /* always N_CONN_CON */
        ulong           RES_length;             /* responding address length */
        ulong           RES_offset;             /* responding address offset */
        ulong           CONN_flags;             /* bit masking for options
                                                 * flags */
        ulong           QOS_length;             /* length of QOS parameter
                                                 * values */
        ulong           QOS_offset;             /* offset of QOS parameter
                                                 * values */
}               N_conn_con_t;




/*
 * Connection mode data transfer request
 */

typedef struct {
        ulong           PRIM_type;              /* always N_DATA_REQ */
        ulong           DATA_xfer_flags;        /* data transfer flags */
}               N_data_req_t;
```

```
/*
 * NPI MORE_DATA_FLAG for segmenting NSDU into more than 1 NIDUs
 */
#define N_MORE_DATA_FLAG        0x00000001L      /* Indicates that the next
                                                  * NIDU is part of this NSDU */
/*
 * NPI Receipt confirmation request set flag
 */
#define N_RC_FLAG       0x00000002L              /* Indicates if receipt
                                                  * confirmation is required */


/*
 * Incoming data indication for an NC
 */

typedef struct {
        ulong           PRIM_type;               /* always N_DATA_IND */
        ulong           DATA_xfer_flags;         /* data transfer flags */
}               N_data_ind_t;


/*
 * Data acknowledgement request
 */

typedef struct {
        ulong           PRIM_type;               /* always N_DATACK_REQ */
}               N_datack_req_t;


/*
 * Data acknowledgement indication
 */

typedef struct {
        ulong           PRIM_type;               /* always N_DATACK_IND */
}               N_datack_ind_t;
/*
 * Expedited data transfer request
 */

typedef struct {
        ulong           PRIM_type;               /* always N_EXDATA_REQ */
}               N_exdata_req_t;


/*
 * Expedited data transfer indication
 */

typedef struct {
        ulong           PRIM_type;               /* always N_EXDATA_IND */
```

```
}               N_exdata_ind_t;


/*
 * NC reset request
 */

typedef struct {
        ulong           PRIM_type;                  /* always N_RESET_REQ */
        ulong           RESET_reason;               /* reason for reset */
}               N_reset_req_t;
/*
 * NC reset indication
 */

typedef struct {
        ulong           PRIM_type;                  /* always N_RESET_IND */
        ulong           RESET_orig;                 /* reset originator */
        ulong           RESET_reason;               /* reason for reset */
}               N_reset_ind_t;
/*
 * NC reset response
 */
typedef struct {
        ulong           PRIM_type;                  /* always N_RESET_RES */
}               N_reset_res_t;


/*
 * NC reset confirmed
 */

typedef struct {
        ulong           PRIM_type;                  /* always N_RESET_CON */
}               N_reset_con_t;


/*
 * NC disconnection request
 */

typedef struct {
        ulong           PRIM_type;                  /* always N_DISCON_REQ */
        ulong           DISCON_reason;              /* reason */
        ulong           RES_length;                 /* responding address length */
        ulong           RES_offset;                 /* responding address offset */
        ulong           SEQ_number;                 /* sequence number */
}               N_discon_req_t;


/*
 * NC disconnection indication
 */
```

```
typedef struct {
        ulong           PRIM_type;              /* always N_DISCON_IND */
        ulong           DISCON_orig;            /* originator */
        ulong           DISCON_reason;          /* reason */
        ulong           RES_length;             /* address length */
        ulong           RES_offset;             /* address offset */
        ulong           SEQ_number;             /* sequence number */
}               N_discon_ind_t;



/*
 * CLNS PRIMITIVES
 */
/*
 * Unitdata transfer request
 */

typedef struct {
        ulong           PRIM_type;              /* always N_UNITDATA_REQ */
        ulong           DEST_length;            /* destination address
                                                 * length */
        ulong           DEST_offset;            /* destination address
                                                 * offset */
        ulong           RESERVED_field[2];      /* reserved field for DLPI
                                                 * compatibility */
}               N_unitdata_req_t;


/*
 * Unitdata transfer indication
 */

typedef struct {
        ulong           PRIM_type;              /* always N_UNITDATA_IND */
        ulong           SRC_length;             /* source address length */
        ulong           SRC_offset;             /* source address offset */
        ulong           DEST_length;            /* source address length */
        ulong           DEST_offset;            /* source address offset */
        ulong           ERROR_type;             /* reserved field for DLPI
                                                 * compatibility */
}               N_unitdata_ind_t;



/*
 * Unitdata error indication for CLNS services
 */

typedef struct {
        ulong           PRIM_type;              /* always N_UDERROR_IND */
        ulong           DEST_length;            /* destination address
                                                 * length */
```

```
        ulong           DEST_offset;            /* destination address
                                                 * offset */
        ulong           RESERVED_field;         /* reserved field for DLPI
                                                 * compatibility */
        ulong           ERROR_type;             /* error type */
}               N_uderror_ind_t;



/*
 * The following represents a union of all the NPI primitives
 */

union N_primitives {
        ulong           type;
        N_info_req_t    info_req;               /* information request */
        N_info_ack_t    info_ack;               /* information
                                                 * acknowledgement */
        N_bind_req_t    bind_req;               /* bind request */
        N_bind_ack_t    bind_ack;               /* bind acknowledgement */
        N_unbind_req_t  unbind_req;             /* unbind request */
        N_optmgmt_req_t optmgmt_req;            /* options management
                                                 * request */
        N_error_ack_t   error_ack;              /* error acknowledgement */
        N_uderror_ind_t uderror_ind;            /* unitdata error indication */
        N_ok_ack_t      ok_ack;                 /* ok acknowledgement */
        N_conn_req_t    conn_req;               /* connect request */
        N_conn_ind_t    conn_ind;               /* connect indication */
        N_conn_res_t    conn_res;               /* connect response */
        N_conn_con_t    conn_con;               /* connect confirm */
        N_data_req_t    data_req;               /* data request */
        N_data_ind_t    data_ind;               /* data indication */
        N_datack_req_t  datack_req;             /* data acknowledgement
                                                 * request */
        N_datack_ind_t  datack_ind;             /* data acknowledgement
                                                 * indication */
        N_exdata_req_t  exdata_req;             /* expedited data request */
        N_exdata_ind_t  exdata_ind;             /* expedited data indication */
        N_reset_req_t   reset_req;              /* reset request */
        N_reset_ind_t   reset_ind;              /* reset indication */
        N_reset_res_t   reset_res;              /* reset response */
        N_reset_con_t   reset_con;              /* reset confirm */
        N_discon_req_t  discon_req;             /* disconnect request */
        N_discon_ind_t  discon_ind;             /* disconnect indication */
        N_unitdata_req_t unitdata_req;          /* unitdata request */
        N_unitdata_ind_t unitdata_ind;          /* unitdata indication */
};
```

PLEASE DISCARD THIS PAGE!!!

Table of Contents