

NPI

User Manual

February 2003

Protocols: X.25, SNA, and HDLC/SDLC

Copyright © GCOM, Inc.
All rights reserved.

© 1993-2003 GCOM, Inc. All rights reserved.

Non-proprietary—Provided that this notice of copyright is included, this document may be copied in its entirety without alteration. Permission to publish excerpts should be obtained from GCOM, Inc.

GCOM reserves the right to revise this publication and to make changes in content without obligation on the part of GCOM to provide notification of such revision or change. The information in this document is believed to be accurate and complete on the date printed on the title page. No responsibility is assumed for errors that may exist in this document.

Rsystem is a registered trademark of GCOM, Inc. UNIX is a registered trademark of UNIX Systems Laboratories, Inc. in the U.S. and other countries. SCO is a trademark of the Santa Cruz Operation, Inc. IBM PC, IBM PC/AT, OS/2 and PC DOS are registered trademarks of International Business Machines Corporation. All other brand product names mentioned herein are the trademarks or registered trademarks of their respective owners.

Any provision of this product and its manual to the U.S. Government is with “Restricted Rights”: Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013 of the DoD FAR Supplement.

This manual was written, formatted and produced by technical writers Debra J. Schweiger and Scott D. Smith using Microsoft Word 97 and FrameMaker 6.0 on a Windows Millennium platform with the help of subject matter specialist Dave Grothe.

This manual was printed in the U.S.A.

FOR FURTHER INFORMATION

If you want more information about GCOM products, contact us at:

GCOM, Inc.
1800 Woodfield
Savoy, IL 61874
(217) 351-4241
FAX: (217) 351-4240
e-mail: support@gcom.com
homepage: <http://gcom.com>

CONTENTS

| | |
|-----------|--|
| 7 | <i>Purpose of This Guide</i> |
| 7 | Knowledge Requirements |
| 8 | Organization of This Guide |
| 8 | Conventions Used in This Guide |
| 8 | <i>Special Notices</i> |
| 9 | <i>Text Conventions</i> |
| 11 | <i>Section 1 - Overview of the Network Provider Interface (NPI)</i> |
| 12 | 1.1 Feature List |
| 13 | 1.2 NSAPs |
| 14 | 1.3 Outbound Connections |
| 14 | 1.4 Inbound Connections |
| 15 | <i>Section 2 - Using NPI</i> |
| 16 | 2.1 Establishing a Connection |
| 18 | <i>N_BIND_REQ for Outbound Connections</i> |
| 19 | <i>N_BIND_ACK for Outbound Connections</i> |
| 20 | <i>N_CONN_REQ</i> |
| 21 | <i>N_CONN_CON</i> |
| 22 | <i>N_BIND_REQ for Inbound Connections</i> |
| 24 | <i>N_BIND_ACK for Inbound Connections</i> |
| 25 | <i>N_CONN_IND for Inbound Connections</i> |
| 27 | <i>N_CONN_RES for Inbound Connections</i> |
| 29 | 2.2 Disconnecting |
| 30 | <i>N_DISCON_REQ</i> |
| 31 | <i>N_DISCON_IND</i> |
| 33 | 2.3 Resetting a Connection |
| 33 | <i>N_RESET_REQ</i> |
| 35 | <i>N_RESET_IND</i> |
| 36 | 2.4 Data Transfer |
| 36 | <i>M_DATA vs. M_PROTO</i> |
| 37 | <i>Normal Data</i> |
| 38 | <i>Expedited Data</i> |
| 40 | <i>Receipt Confirmation</i> |
| 40 | <i>Packetizing of Data Messages</i> |
| 42 | 2.5 Option Management |
| 42 | <i>N_INFO_REQ</i> |

| | |
|-----------|--|
| 44 | <i>N_OPTMGMT_REQ</i> |
| 45 | <i>Section 3 - GCOM Extensions to NPI</i> |
| 45 | 3.1 Extensions to the Data Primitives |
| 47 | 3.2 Extensions to Bind Request |
| 48 | <i>N_EXT_BIND_REQ</i> |
| 49 | <i>N_EXT2_BIND_REQ</i> |
| 50 | 3.3 Extensions to Connection Primitives |
| 51 | <i>N_EXT_CONN_REQ</i> |
| 52 | <i>N_EXT_CONN_RES</i> |
| 52 | <i>N_EXT_CONN_IND</i> |
| 53 | <i>N_EXT2_CONN_IND</i> |
| 55 | <i>N_EXT_CONN_CON</i> |
| 56 | <i>N_EXT2_CONN_CON</i> |
| 57 | 3.4 Flow Control Primitives |
| 59 | 3.5 Draining Data |
| 61 | <i>Index</i> |

PREFACE

Purpose of This Guide

Knowledge Requirements

This document is intended for those who wish to use GCOM's implementation of the Network Provider Interface (NPI) for UNIX system 5.4. This implementation is based upon the AT&T document *STREAMS-Based Network Provider Interface Specification - Version 1.3* by R. J. Lewis and P. Bajpay at AT&T Bell Laboratories. The user is presumed to be somewhat familiar with the AT&T document.

GCOM's implementation of NPI incorporates the Rsystem X.25 packet level as the protocol processing software with an NPI provider above it to interface X.25 to STREAMS and a DLPI user below it to interface to a STREAMS based data link layer in a lower STREAMS module. The GCOM package for NPI consists of one STREAMS driver which incorporates the NPI interface code, the X.25 packet level and the DLPI user code.

Prior to using NPI streams for user oriented functions, such as establishing connections, the NPI module must be configured. The configuration must specify parameters for the X.25 packet level, characteristics of the DLPI user module and addressing information for the NPI module. The configuration is accomplished by the use of a control stream. The format of the configuration messages is documented in the *GCOM Configuration Manual*.

In a manner similar to X.25, the same NPI Provider serves as an interface to the user for GCOM's SNA protocol stack and GCOM's Bisync protocol stack.

Organization of This Guide

Table 1 shows the organization of this manual and tells you where to find specific information.

Table 1 Location of Important Information

| <i>For information about:</i> | <i>Look at:</i> |
|--|-----------------|
| Overview of the Network Provider Interface (NPI) | Section 1 |
| Using the NPI | Section 2 |
| GCOM extensions to NPI | Section 3 |

Conventions Used in This Guide

This section discusses conventions used throughout this guide.

Special Notices

A special format indicates notes, cautions and warnings. The purpose of these notices is defined as follows:



Note: *Notes call attention to important features or instructions.*



Caution: Cautions contain directions that you must follow to avoid immediate system damage or loss of data.



Warning! Warnings contain directions that you must follow for your personal safety. Follow these instructions carefully.

Text Conventions

The use of italics, boldface and other text conventions are explained as follows:

Terminology

The following terms appear in **boldface**: directories and file names. An example is the **hstpar.h** include file.

Boldface names within angle brackets refer to the global copy of the file. For instance, **<intsx25.h>** refers to **/rsys/include/intsx25.h**.

The following terms appear in *italics*: variables (parameters), fields, structures, glossary terms, routines (functions, programs, utilities and applications), flags, commands and scripts. Examples include the *count* variable, *Command Type* field, *rteparam* structure, *target* term, *rsys_read()* routine, *avail* flag, *Add Route* command and *gcomunld* script.

“Enter” vs. “Type”

When the word “enter” is used in this guide, it means type something and then press the Return key. Do not press Return when an instruction simply says “type.”

Screen Display

This `typeface` is used to represent displays that appear on a terminal screen. Commands entered at the prompt use the same typeface only in boldface. For example:

```
C:> cd gcom
% cd gcom
# cd gcom
```

Each of these commands instructs you to enter “cd gcom” at the system prompt.

Section 1

Overview of the Network Provider Interface (NPI)

The Network Provider Interface (NPI) is an interface protocol between a STREAMS driver and a user. The user can be another STREAMS driver or pushable module, or it can be a program running at user level and using the *getmsg* and *putmsg* user functions to interface to STREAMS.

NPI exists at the network layer of the OSI protocol stack. It is patterned after the CCITT X.213 protocol, which is a network layer interface protocol.

NPI supposes that there are a number of addressable entities with which one can establish a connection. It assumes that a user of NPI can also be an addressable entity, and thus the target of an incoming network connection. The *Bind* messages are used to associate network addresses with local users, and the *Connect* messages are used to establish connections between local and remote users.

1.1 Feature List

This version of the NPI provider and its associated X.25 packet level protocol module have the following features:

- STREAMS multiplexor with NPI provider on top and DLPI user on bottom
- Support for all NPI primitives except N_OPTMGMT_REQ
- Network management primitives, documented separately, for configuration, statistics and alarms
- Flexible wild card network addressing for binds
- Support for multiple network lines
- Cost-based routing to multiple network lines configurable by network management
- Call retry for the case of multiple routes to a given destination
- Multiple outstanding receipt confirmation data messages handled efficiently
- Debugging features that can be enabled by network management to help troubleshoot STREAMS protocol interaction problems
- X.25 facilities for calls specifiable on a per line basis by network management
- X.25 protocol module fully 1988 compliant
- X.25 can be configured separately for each line by network management
- X.25 can act as DTE or DCE
- Support for all X.2 facilities
- Packet traces available through network management
- Compatible with GCOM's DLPI provider, which provides X.25 LAPB, SDLC and MLP support

1.2 NSAPs

The AT&T document is a bit vague concerning the format of a Network Service Access Point (NSAP) address. However, since GCOM's NPI implementation is oriented towards the X.25 packet level, we have defined an NSAP address format that is consistent with X.25 addresses and X.121, the international numbering plan for addresses.

All NPI protocol messages that contain NSAPs consist of a header portion, which includes a count field for the size of the NSAP (in bytes according to the AT&T document), and a variable length field of bytes containing the NSAP farther down in the message buffer.

The string of bytes that constitute the NSAP has one of the following formats:

- Form 1 A string of decimal digits—The high order 4 bits of each byte is the left hand digit, and the low order 4 bits is the right hand digit. For example, the digit string 123456 would be encoded in consecutive bytes as 0x12 0x34 0x56 (byte count of 3).
- Form 2 A string of ASCII characters representing decimal digits and wildcard characters—In this case the first byte contains the value 0xF0, and the following bytes consist of ASCII characters. To encode the digit string 123456 using this scheme, the sequence of bytes would be 0xF0 0x31 0x32 0x33 0x34 0x35 0x36 (byte count of 7).

Understanding Form 1 and Form 2

In form 1, there must be an even number of digits in the NSAP. This is because the count is in *bytes* but the address itself is in 4-bit *digits*.

For form 2, the address can be an even or odd number of digits and pattern-matching wildcard characters can be included. The 0xF0 that introduces form 2 is dropped from the address.

The wildcard characters are “?”, which matches any single digit and “*”, which matches zero or more consecutive digits. Wildcard addresses are not useful for specifying remote addresses since these addresses must be in CCITT X.121 form. However, they are useful for specifying local addresses in the N_BIND_REQ.

For example, an address of 0xF0 *01 would match any address that ends with the digits 01. This could be a subaddress associated with a particular network service.

Form 1 seems to be the form specified by AT&T. Form 2 is a GCOM extension provided to get around the odd-number of digits restriction and to offer flexibility in binding addresses for listening purposes.

1.3 Outbound Connections

Outbound connections are originated by the NPI user. The user issues an N_CONN_REQ to NPI, and NPI, in turn, issues an X.25 call request packet to the network. The user receives an N_CONN_CON if the connection succeeds and an N_DISCON_IND if it fails. If the N_CONN_REQ was malformed, the user receives an N_ERROR_ACK.

1.4 Inbound Connections

Inbound connections originate with the NPI user offering to “listen” on a particular address. This is done with a special form of the N_BIND_REQ message. An X.25 incoming call then matches up with the user’s “listen” through its address. The incoming call is converted to an N_CONN_IND message and sent to the user on the listening stream. The user can choose to accept the call directly on the listening stream or transfer it to another stream. In the latter case, the user opens another stream to NPI, issues an N_BIND_REQ on the new stream and uses the N_CONN_RES message, sent on the listening stream, to transfer the call to the new stream.

Section 2

Using NPI

This section of the manual explains the steps that a user must take in order to use the NPI driver. It discusses connection establishment, disconnecting, data transfers and resets.

2.0 NPI Files

The following files in the /dev directory are used in conjunction with the NPI driver.

`/dev/npip` This file represents the control stream of the NPI multiplexing driver. It is used by `Gcom_monitor` for configuration and for monitoring event indications from the driver. Other GCOM utilities use this file to extract trace table and statistical data from the driver. The formats of messages exchanged between the NPI driver and these utilities is GCOM proprietary information and is not documented externally. User programs should never open this file.

`/dev/npip_clone` This file represents a clone open to the NPI driver. Each open of this file returns a separate data stream over which the NPI protocol is used. User programs may access these protocol objects via `getmsg/putmsg`. STREAMS drivers may access this message stream via `I_PUSH` or `I_LINK`.

`/dev/npip1_clone,`

`/dev/npip2_clone, etc` Some operating systems, such as Linux, restrict the number of minor devices per major device to 256. In these cases, GCOM provides more major devices that are controlled by the NPI driver so as to increase the number of available NPI data streams. If an open of `/dev/npip_clone` returns an error with `errno` set to `EBUSY`, you can then try opening these extra devices for additional streams.

Non-Linux systems should not have this problem since the limit to the number of minor devices is large enough not to pose practical problems.

2.1 Establishing a Connection

Connections can be established through NPI as either outbound or inbound connections.

Outbound Connections

For outbound connections, follow these steps:

1. **Open a stream to NPI.**
2. **Issue an N_BIND_REQ to NPI—This associates a local address with the connection.**
3. **Receive an N_BIND_ACK from NPI, signaling success of the bind.**
4. **Issue an N_CONN_REQ to NPI—This carries the remote address of the connection.**
5. **Receive an N_CONN_CON from NPI, indicating that the connection has been established.**

Inbound Connections

For inbound connections, follow these steps:

1. **Open a stream to NPI.**
2. **Issue an N_BIND_REQ to NPI—This associates a local address with the connection and specifies the number of outstanding connect indications allowed for the stream.**
3. **Receive an N_BIND_ACK from NPI, signalling success of the bind.**
4. **Receive an N_CONN_IND on the open stream.**
5. **Open a second stream to NPI.**
6. **Issue an N_BIND_REQ to NPI on the new stream—This bind will request a token to be used to transfer the call from the listening (original) stream to the new stream.**
7. **Receive an N_BIND_ACK from NPI with the token.**
8. **Issue an N_CONN_RES message to NPI by way of the listening stream—The token specifies the new stream to which the call is transferred.**
9. **Receive an N_OK_ACK on the new stream.**

N_BIND_REQ for Outbound Connections

The N_BIND_REQ sent to NPI has the following form. We will note anything special about the fields of this message here. Consult the AT&T NPI document for the full definition of the fields of the message.

```
typedef struct
{
    ulong PRIM_type;
    ulong ADDR_length;
    ulong ADDR_offset;
    ulong CONIND_number;
    ulong BIND_flags;
} N_bind_req_t;
```

The fields are used as follows:

| | |
|----------------------|--|
| <i>PRIM_type</i> | Set to N_BIND_REQ. |
| <i>ADDR_length</i> | Length of address—The length is given in bytes. The address itself is encoded as described in “1.2 NSAPs” on page 13. <i>Note:</i> If the “digits” form of the NSAP is used, the <i>ADDR_length</i> field will be equal to half the number of digits in the address. For example, an address field consisting of the bytes 0x12 0x24 0x56 (6 digits) will have an <i>ADDR_length</i> equal to 3 (the number of bytes). An address field consisting of 0xF0 0x31 0x32 0x33 0x34 0x35 0x36 (6 digits with escape) will have an <i>ADDR_length</i> equal to 7. Addresses with an odd number of digits must use the escape form. |
| <i>ADDR_offset</i> | This is the byte offset from the beginning of the M_PROTO message block to the first byte of the address. |
| <i>CONIND_number</i> | For outgoing connections, this field is always equal to zero. |
| <i>BIND_flags</i> | For outgoing connections, this field is always equal to zero. Refer to GCOM extensions for additional values of this field. |

N_BIND_ACK for Outbound Connections

The form of the N_BIND_ACK sent to the NPI user for outgoing connections is as follows:

```
typedef struct
{
    ulong PRIM_type;
    ulong ADDR_length;
    ulong ADDR_offset;
    ulong CONIND_number;
    ulong TOKEN_value;
} N_bind_ack_t;
```

The fields are used as follows:

| | |
|----------------------|---|
| <i>PRIM_type</i> | Set to N_BIND_ACK. |
| <i>ADDR_length</i> | Length of address—This is the same format as in N_BIND_REQ. |
| <i>ADDR_offset</i> | This is the byte offset from the beginning of the M_PROTO message block to the first byte of the address. |
| <i>CONIND_number</i> | For outgoing connections, this field is always equal to zero. |
| <i>TOKEN_value</i> | For outgoing connections, this field is always equal to zero. |

N_CONN_REQ

The N_CONN_REQ sent to NPI has the following form. We will note anything special about the fields of this message here. Consult the AT&T NPI document for the full definition of the fields of the message. See “N_EXT_CONN_REQ” on page 51 for GCOM extensions.

```
typedef struct
{
    ulong PRIM_type;
    ulong DEST_length;
    ulong DEST_offset;
    ulong CONN_flags;
    ulong QOS_length;
    ulong QOS_offset;
} N_conn_req_t;
```

The fields are used as follows:

| | |
|--------------------|--|
| <i>PRIM_type</i> | Set to N_CONN_REQ. |
| <i>DEST_length</i> | Length of address. This is the same format as in N_BIND_REQ. Wildcard addresses will result in connection failure. |
| <i>DEST_offset</i> | This is the byte offset from the beginning of the M_PROTO message block to the first byte of the address. |
| <i>CONN_flags</i> | All flags are valid. |
| <i>QOS_length</i> | Specifies length of QOS parameters. |
| <i>QOS_offset</i> | Offset to QOS parameters from start of M_PROTO message block. |

The N_CONN_REQ message results in the building and sending of an X.25 call request packet. The called DTE address comes from the *DEST_addr* in the N_CONN_REQ; the calling DTE address comes from the address in the N_BIND_REQ; the X.25 facilities come from defaults supplied by network management and can change depending on the particular access line to which the call is routed; and the call user data comes from any data attached to the N_CONN_REQ message.

If an M_DATA block is attached to the M_PROTO by way of the *b_cont* field in the *mblk_t* header of the M_PROTO, the data in the M_DATA will be sent with the X.25 call request packet in the call user *data* field.

The *QOS* field can be present in the message but its contents do not affect the call request packet submitted to X.25.

NPI compares the *DEST_addr* to address patterns set up by network management for each X.25 line. It makes a list of X.25 lines whose addresses match and orders the list in increasing “cost”. It then tries the call on each line until one of them succeeds or all of them fail. The call that is sent to each individual line can have different facilities in it due to default facilities being different for different X.25 lines.

If the call succeeds, the user is sent an N_CONN_CON message; if it fails, the user is sent an N_DISCON_IND message; and, if the N_CONN_REQ itself was malformed, the user is sent an N_ERROR_ACK message.

N_CONN_CON

The N_CONN_CON sent to the NPI user has the following form. We will note anything special about the fields of this message here. Consult the AT&T NPI document for the full definition of the fields of the message. See “N_EXT_CONN_CON” on page 55 for GCOM extensions.

```
typedef struct
{
    ulong PRIM_type;
    ulong RES_length;
    ulong RES_offset;
    ulong CONN_flags;
    ulong QOS_length;
    ulong QOS_offset;
} N_conn_con_t;
```

The fields are used as follows:

| | |
|-------------------|---|
| <i>PRIM_type</i> | Set to N_CONN_CON. |
| <i>RES_length</i> | Length of address. This is the same format as in N_BIND_REQ. |
| <i>RES_offset</i> | This is the byte offset from the beginning of the M_PROTO message block to the first byte of the address. |
| <i>CONN_flags</i> | All flags are valid. |
| <i>QOS_length</i> | Specifies length of QOS parameters. This will |

always be zero.

QOS_offset Offset to QOS parameters from start of M_PROTO message block. This will always be zero.

NPI constructs this message from a received X.25 call connected packet. The RES_addr comes from the called DTE address of the call connected packet. This address will be absent if there was no called DTE address present in the packet.

The *QOS* field will never be present in the N_CONN_CON message.

If the call connected packet had call user data in it then that data will be forwarded to the user in an M_DATA block attached to the M_PROTO by way of the *b_cont* field of the *mblk_t* header of the M_PROTO.

N_BIND_REQ for Inbound Connections

The N_BIND_REQ sent to NPI has the following form. We will note anything special about the fields of this message here. Consult the AT&T NPI document for the full definition of the fields of the message.

```
typedef struct
{
    ulong PRIM_type;
    ulong ADDR_length;
    ulong ADDR_offset;
    ulong CONIND_number;
    ulong BIND_flags;
} N_bind_req_t;
```

The fields are used as follows:

PRIM_type Set to N_BIND_REQ.

ADDR_length Length of address—This is the same form as specified above for outgoing connections. The wildcard form of the address is especially useful for binding a listener stream. For instance, the wildcard address of “*01” could be used to listen for any call coming in on sub-address “01”.

ADDR_offset The byte offset from the beginning of the M_PROTO message block to the first byte of the address

CONIND_number For incoming connections this field can contain a

non-zero number—This is the number of outstanding N_CONN_IND messages that NPI can send to the user before receiving an N_CONN_RES for one of them. NPI will attempt to allocate enough memory to satisfy the user's request. If the memory allocation fails, it will attempt to find a suitable number and return that reduced number in the N_BIND_ACK. If this stream is going to be used as a transfer stream, then this field should be set to zero.

BIND_flags

The DEFAULT_LISTENER flag can be set to specify that this stream is to receive all incoming calls for which no other bound stream can be found.

Note: Only one default listener stream is allowed in NPI. The TOKEN_REQUEST flag can be set if this stream is being used as a transfer stream. This allows NPI to return a “token” value in the N_BIND_ACK, which you can then use in your N_CONN_RES to transfer the call from the listener stream to the new stream. GCOM defines additional values for this field in a set of extensions, documented below.

N_BIND_ACK for Inbound Connections

The form of the N_BIND_ACK sent to the NPI user for incoming connections is as follows:

```
typedef struct
{
    ulong PRIM_type;
    ulong ADDR_length;
    ulong ADDR_offset;
    ulong CONIND_number;
    ulong TOKEN_value;
} N_bind_ack_t;
```

The fields are used as follows:

| | |
|----------------------|--|
| <i>PRIM_type</i> | Set to N_BIND_ACK. |
| <i>ADDR_length</i> | Length of address. This is the same format as in N_BIND_REQ. |
| <i>ADDR_offset</i> | This is the byte offset from the beginning of the M_PROTO message block to the first byte of the address. |
| <i>CONIND_number</i> | This field contains the actual number of outstanding N_CONN_INDs that NPI is prepared to send on the listener stream. This value will be less than or equal to the <i>CONIND_number</i> field contained in the N_BIND_REQ. |
| <i>TOKEN_value</i> | This field is non-zero if the N_BIND_REQ specified the TOKEN_REQUEST flag. It is a number that can be used in the <i>TOKEN_value</i> field of an N_CONN_RES subsequently sent on the listener stream to transfer the incoming call to this stream. For network management purposes, the <i>TOKEN_value</i> is actually the minor device number of the stream that sent the N_BIND_REQ. |

N_CONN_IND for Inbound Connections

The form of the N_CONN_IND sent to the NPI user for incoming connections is as follows:

```
typedef struct
{
    ulong PRIM_type;
    ulong DEST_length;
    ulong DEST_offset;
    ulong SRC_length;
    ulong SRC_offset;
    ulong SEQ_number;
    ulong CONN_flags;
    ulong QOS_length;
    ulong QOS_offset;
} N_conn_ind_t;
```

The fields are used as follows:

| | |
|--------------------|---|
| <i>PRIM_type</i> | Set to N_CONN_IND. |
| <i>DEST_length</i> | Length of destination address—Same format as in N_BIND_REQ. |
| <i>DEST_offset</i> | This is the byte offset from the beginning of the M_PROTO message block to the first byte of the address. |
| <i>SRC_length</i> | Length of source address—Same format as in N_BIND_REQ. |
| <i>SRC_offset</i> | This is the byte offset from the beginning of the M_PROTO message block to the first byte of the address. |
| <i>SEQ_number</i> | Sequence number assigned to N_CONN_IND by NPI—This allows a subsequent N_CONN_RES or N_DISCON_REQ from the user or an N_DISCON_IND from NPI to reference the same connection. |
| <i>CONN_flags</i> | All flags are valid. |
| <i>QOS_length</i> | Specifies length of QOS parameters—This field will always be zero. |
| <i>QOS_offset</i> | Offset to QOS parameters from start of the M_PROTO message block. This field will always be zero. |

The `N_CONN_IND` is sent to the user when an X.25 incoming call packet arrives. The stream is chosen based on the address that was bound to it in the `N_BIND_REQ`. These addresses can contain wildcard characters so as to match patterns of addresses in the incoming call packets.

The `DEST_addr` is formed from the called DTE address in the incoming call. The `SRC_addr` comes from the calling DTE address of the incoming call. The `QOS` field never contains anything in this implementation of NPI. Thus, the `QOS_length` and `offset` fields are both set to zero. The `CONN_flags` field is set from information obtained from an examination of the X.25 incoming call packet. The `REC_CONF_OPT` is set from the X.25 D-bit (the delivery confirmation bit). The `EX_DATA_OPT` bit is set from the expedited data negotiation facility in the incoming call if it is present or is set to zero if that facility is not present in the call.

The `SEQ_number` field is a unique number chosen by NPI that allows the user or NPI to subsequently refer to this connection during the setup phase. This number will be used in an `N_CONN_RES` or `N_DISCON_REQ` sent from the user to accept or reject the connection. NPI can use the number in a subsequent `N_DISCON_IND` if the remote user clears the call before the connection completes. For network management purposes, the `SEQ_number` is actually the Upper Point of Attachment (UPA) number assigned to the incoming call by NPI.

N_CONN_RES for Inbound Connections

The form of the N_CONN_RES sent by the NPI user for incoming connections is as follows:

```
typedef struct
{
    ulong PRIM_type;
    ulong TOKEN_value;
    ulong RES_length;
    ulong RES_offset;
    ulong SEQ_number;
    ulong CONN_flags;
    ulong QOS_length;
    ulong QOS_offset;
} N_conn_res_t;
```

The fields are used as follows:

| | |
|--------------------|---|
| <i>PRIM_type</i> | Set to N_CONN_RES. |
| <i>TOKEN_value</i> | The value supplied by NPI in an N_BIND_ACK on a different stream. |
| <i>RES_length</i> | Length of destination address—This is the same format as in N_BIND_REQ. |
| <i>RES_offset</i> | The byte offset from the beginning of the M_PROTO message block to the first byte of the address. |
| <i>SEQ_number</i> | A sequence number assigned by NPI to the N_CONN_IND to which this N_CONN_RES is a response. |
| <i>CONN_flags</i> | All flags are valid. |
| <i>QOS_length</i> | Specifies length of QOS parameters. |
| <i>QOS_offset</i> | Offset to QOS parameters from the start of the M_PROTO message block. |

The user sends an N_CONN_RES message in order to accept an incoming connection. The *SEQ_number* field is used to reference the particular incoming call that the user is accepting. The number in this field is the same as the number provided by NPI in the N_CONN_IND to which this is a response.

The *TOKEN_value* field is used to transfer the call to another stream. In order to do this, the user must have opened a second stream, issued an N_BIND_REQ requesting a token, and received an N_BIND_ACK that contained the requested token. The token value is a reference number for the second stream. When the user sends an N_CONN_RES it is sent on the same stream on which NPI sent the N_CONN_IND, that is, the listener stream. The *TOKEN_value* then references the second stream and NPI transfers the connection to that stream.

A *TOKEN_value* field of zero means that the user is accepting the connection on the listener stream itself.

The *CONN_flags* field contains user negotiations for receipt confirmation (REC_CONF_OPT) and expedited data (EX_DATA_OPT).

The N_CONN_RES message causes NPI to build and send an X.25 call accepted packet to the network. The called DTE address is supplied by the RES_addr. The calling DTE address is not used. The *facility* field is left empty. The call user *data* field is set from the contents of any M_DATA message that is attached to the M_PROTO by way of the *b_cont* field in the *mblk_t* header for the M_PROTO that carries the N_CONN_RES.

If you want to refuse a connection rather than accept it, you must send an N_DISCON_REQ to NPI in response to an N_CONN_IND. See the next section, "Disconnecting".

2.2 Disconnecting

In NPI a connection can be disconnected by the NPI user, the NPI provider or the remote NPI user or provider. From the user's perspective there are only two cases that matter procedurally, namely, disconnect by the user and disconnect by "the other end" of the connection. In the first case, the user issues an N_DISCON_REQ to the NPI provider; in the second, the user receives an N_DISCON_IND from the provider. The N_DISCON_IND will contain some clues as to which module initiated the disconnect.

In general the NPI disconnect primitives map onto X.25 clear request/indication packets. However, an X.25 restart indication received by the packet level protocol module will result in N_DISCON_IND messages being sent to the user on every stream using that particular X.25 access line. Streams routed to other access lines are unaffected.

When the user issues an N_DISCON_REQ to NPI, an N_OK_ACK or N_ERROR_ACK will be returned to indicate the success or failure, respectively, of the request.

When NPI sends an N_DISCON_IND to the user, there is no response expected from the user.

N_DISCON_REQ

The N_DISCON_REQ message sent to NPI has the following form. We will note anything special about the fields of this message here. Consult the AT&T NPI document for the full definition of the fields of the message.

```
typedef struct
{
    ulong PRIM_type;
    ulong DISCON_reason;
    ulong RES_length;
    ulong RES_offset;
    ulong SEQ_number;
} N_discon_req_t;
```

The fields are used as follows:

| | |
|----------------------|--|
| <i>PRIM_type</i> | Set to N_DISCON_REQ. |
| <i>DISCON_reason</i> | The reason for the disconnection. |
| <i>RES_length</i> | Length of address. This is the same format as in N_BIND_REQ. |
| <i>RES_offset</i> | This is the byte offset from the beginning of the M_PROTO message block to the first byte of the address. |
| <i>SEQ_number</i> | Used to refuse an incoming connection previously indicated by an N_CONN_IND— This is the same number as was present in the N_CONN_IND message. |

The N_DISCON_REQ causes NPI to send a clear request packet on the X.25 access line. The called DTE address field will be set from the *RES_addr* supplied in the N_DISCON_REQ message; the calling DTE address will be left empty.

The low order 16 bits of the DISCON_reason is defined by the NPI specification and the file <npi.h>. The high order 16 bits represent a Gcom extension to the use of this field. The value of the high order byte will be placed into the cause field of the X.25 clear request packet and the next-high order byte will be placed into the diagnostic field.

The clear request packet will have no facilities.

If the user attaches an M_DATA block to the M_PROTO carrying the

N_DISCON_REQ, the content of the M_DATA buffer will be sent in the clear user *data* field of the X.25 clear request packet.

The NPI user will be sent an N_OK_ACK when the clear is confirmed by X.25.

N_DISCON_IND

The form of the N_DISCON_IND sent to the NPI user is as follows:

```
typedef struct
{
    ulong PRIM_type;
    ulong DISCON_orig;
    ulong DISCON_reason;
    ulong RES_length;
    ulong RES_offset;
    ulong SEQ_number;
} N_discon_ind_t;
```

The fields are used as follows:

| | |
|----------------------|--|
| <i>PRIM_type</i> | Set to N_DISCON_IND. |
| <i>DISCON_orig</i> | Gives information concerning the origin of the disconnect procedure. |
| <i>DISCON_reason</i> | The reason for the disconnection. |
| <i>RES_length</i> | Length of address. This is the same format as in N_BIND_REQ. |
| <i>RES_offset</i> | This is the byte offset from the beginning of the M_PROTO message block to the first byte of the address. |
| <i>SEQ_number</i> | Used to disconnect an incoming connection previously indicated by an N_CONN_IND. This is the same number as was present in the N_CONN_IND message. |

The N_DISCON_IND is sent from the NPI provider to the NPI user to disconnect a connection. The *DISCON_orig* field gives information concerning the origin of the disconnect procedures.

This field and the low order 16 bits of the DISCON_reason field are shown in Table 2. The high order 16 bits of the DISCON_reason are encoded with the X.25 cause and diagnostic field value in the same manner as for the N_DISCON_REQ.

Table 2 DISCON_orig Field and Reasons

| <i>DISCON_orig</i> | <i>DISCON_reason</i> | <i>Circumstance</i> |
|--------------------|----------------------|---|
| N_PROVIDER | N_REJ_NSAP_UNREACH_T | NPI reached its retry limit on an outgoing N_CONN_REQ. |
| N_PROVIDER | N_DISC_T | An N_DISCON_IND with a <i>DISCON_orig</i> of N_PROVIDER and a <i>DISCON_reason</i> of N_DISC_T can be issued under any of the following circumstances: <ul style="list-style-type: none"> • A malformed call connected packet was received from X.25. • A call connected packet was received in the wrong state. • A clear indication packet was received in some state in which a connection was not in progress. • A clear confirmation packet was received in the wrong state. • A restart indication was received on the X.25 line. • A reset indication packet was received in a non-data state. • A reset confirmation packet was received in the wrong state. |
| N_PROVIDER | N_REJ_NSAP_UNKNOWN | No route could be found for an N_CONN_REQ. |
| N_USER | N_REJ_UNSPECIFIED | Normal clear by remote user. |

The *RES_addr* field, if present, will be the remote address of the connection.

If the clear is being sent on a listener stream, the *SEQ_number* field will match the *SEQ_number* field of a previously sent N_CONN_IND message on that stream. This can happen when the remote user decides to clear a call before it completes. The user is cautioned that races can occur between an N_DISCON_IND sent by the NPI provider and an N_CONN_RES sent by the user, and it is up to the user to sort things out.

2.3 Resetting a Connection

NPI provides a mechanism by which a connection can be reset while it is in the data state. The reset mechanism involves the flushing of all data on the connection and a confirmation from the remote end of the connection that it has received and processed the reset.

NPI reset messages, `N_RESET_REQ` and `N_RESET_IND`, are mapped onto the X.25 reset request and reset indication packets, respectively. Similarly, the `N_RESET_RES` is mapped onto an outgoing reset confirmation packet, and a received reset confirmation packet is mapped onto an `N_RESET_CON` message.

Resets can be initiated by the local user, the NPI provider, the X.25 protocol software, the network or the remote user. From a procedural standpoint the only two cases are that of the local user initiated reset and the “remote” initiated reset; in the former the user issues an `N_RESET_REQ` to the NPI provider, and in the latter the NPI provider sends an `N_RESET_IND` to the user.

`N_RESET_REQ`

The `N_RESET_REQ` sent to NPI has the following form. We will note anything special about the fields of this message here. Consult the AT&T NPI document for the full definition of the fields of the message.

```
typedef struct
{
    ulong PRIM_type;
    ulong RESET_reason;
} N_reset_req_t;
```

The fields are used as follows:

PRIM_type Set to `N_RESET_REQ`.
RESET_reason Reason for the reset.

The NPI `N_RESET_REQ` message causes NPI to issue an X.25 reset request packet to the X.25 line. The low order 16 bits of the `RESET_reason` field are encoded in accordance with the values defined in the file `<npi.h>`. The high order 16 bits are used to encode the X.25 cause and diagnostic fields in the same manner as the `N_DISCON_REQ`.

All queued data are flushed, and data packets received from X.25 will

be discarded until X.25 confirms the reset. When the reset is confirmed, NPI will send an N_RESET_CON to the user.

N_RESET_IND

The form of the N_RESET_IND sent to the NPI user is as follows.

```
typedef struct
{
    ulong PRIM_type;
    ulong RESET_orig;
    ulong RESET_reason;
} N_reset_ind_t;
```

The fields are used as follows:

| | |
|---------------------|---|
| <i>PRIM_type</i> | Set to N_RESET_IND. |
| <i>RESET_orig</i> | Information concerning the origin of the reset procedure. |
| <i>RESET_reason</i> | The reason for the reset. |

NPI sends an N_RESET_IND to the user when it receives a reset indication packet from X.25. The *RESET_orig* will be set to N_USER if the *cause* field of the received reset indication was 0 (reset by remote DTE) or will be set to N_PROVIDER otherwise. Note that if the *RESET_orig* field is set to N_PROVIDER, the reset could have originated in the local X.25 protocol module, the network or at any stage between the local NPI and the remote user.

The *RESET_reason* field is encoded in the same manner as for the N_RESET_REQ with the high order 16 bits containing the X.25 cause and diagnostic field values.

NPI flushes all data when it sends the N_RESET_IND to the user. It will continue to discard data from the user until the user issues an N_RESET_RES to NPI. The N_RESET_RES will cause NPI to send a reset confirmation packet to X.25.

2.4 Data Transfer

NPI supports several types of data transfer messages over an NPI connection:

- normal* Consists of messages submitted by one end of the connection and received at the other. The delivery of normal data is not confirmed to its sender. All data is delivered in sequence and with the original message boundaries preserved.
- confirmed* Data messages sent with receipt confirmation are sequenced along with normal data, but, in this case, the sender is notified that the remote end of the connection has received the data.
- expedited* Expedited data messages are sent at a logical higher priority than normal data. Expedited data messages are not blocked by flow control and are allowed to jump to the head of queues. Multiple expedited messages are delivered in sequence to the remote end of the connection.

NPI also makes provision for the segmenting of messages. If a data message contained in an M_DATA block is prepended with an M_PROTO with an N_DATA_REQ (or N_DATA_IND) header in it, then the user can set a bit in the *DATA_xfer_flags* field of the header, N_MORE_DATA_FLAG, to indicate that this message is but a part of a longer message. Thus, logical message boundaries can be preserved even though messages must be segmented to send on the X.25 virtual circuit.

M_DATA vs. M_PROTO

Ordinarily an NPI data message consists simply of an M_DATA message block. However, if the user needs to use any of the extra services connected with the sending of data, he/she must prepend an M_PROTO to the M_DATA. The extra services are expedited data, more data bit or receipt confirmation. If none of these services are required, then a simple M_DATA will do.

The more data bit and receipt confirmation are carried in a protocol header in an M_PROTO. The header is of type N_DATA_REQ or N_DATA_IND.

The expedited data service is indicated by the presence of an M_PROTO header of type N_EXDATA_REQ or N_EXDATA_IND. See “3.1

Extensions to the Data Primitives” on page 45 for GCOM extensions.

Normal Data

Normal data, as distinct from expedited data, can be sent in either of two forms. The M_DATA message form implicitly states that the N_MORE_DATA_FLAG is zero and the N_RC_FLAG is also zero. The M_PROTO with attached M_DATA form allows the user to specify the N_MORE_DATA_FLAG and the N_RC_FLAG.

The forms of the N_DATA_REQ and N_DATA_IND M_PROTO messages are as follows.

```
typedef struct
{
    ulong PRIM_type;
    ulong DATA_xfer_flags;
} N_data_req_t;
```

```
typedef struct
{
    ulong PRIM_type;
    ulong DATA_xfer_flags;
} N_data_ind_t;
```

The fields are used as follows:

| | |
|------------------------|---|
| <i>PRIM_type</i> | Set to N_DATA_REQ or N_DATA_IND. |
| <i>DATA_xfer_flags</i> | Used to set the N_MORE_DATA_FLAG and the N_RC_FLAG. |

For an N_DATA_REQ received from the user, including a simple M_DATA block, NPI builds an X.25 data packet. It sets the M-bit to one if the N_MORE_DATA_FLAG is set and sets the D-bit to one if the N_RC_FLAG is set.

The D-bit will not be set, even if requested, unless the user specified the REC_CONF_OPT in the connect messages that established the connection.

An N_DATA_IND is generated upon receipt of a data packet from X.25. If the M-bit and D-bit are both zero, then a simple M_DATA block is forwarded to the user, otherwise an M_PROTO is prepended to the M_DATA, and an N_DATA_IND header is built in the M_PROTO. The M-bit in the X.25 packet is copied into the N_MORE_DATA_FLAG bit and the D-bit is copied into the N_RC_FLAG bit.

The N_RC_FLAG will not be set unless the user specified the REC_CONF_OPT in the connect messages that established the connection.

Expedited Data

The forms of the N_EXDATA_REQ and N_EXDATA_IND M_PROTO messages are as follows.

```
typedef struct
{
    ulong PRIM_type;
} N_exdata_req_t;
```

```
typedef struct
{
    ulong PRIM_type;
} N_exdata_ind_t;
```

The fields are used as follows:

PRIM_type Set to N_EXDATA_REQ or N_EXDATA_IND.

An N_EXDATA_REQ is translated into an X.25 interrupt packet. The contents of the M_DATA attached to the M_PROTO become the interrupt data field of the interrupt packet. X.25 interrupt packets are not subject to flow control and are sent ahead of normal data packets. However, X.25 has the restriction that only one unconfirmed interrupt request packet for a given direction of data flow can be outstanding at a time. Since NPI has no such restrictions, the NPI provider will queue N_EXDATA_REQ messages as necessary and release them one at a time as prior interrupt packets are confirmed by the receipt of X.25 interrupt confirmation packets.



Note: *Because of this one-at-a-time characteristic, it is very inefficient to use N_EXDATA_REQ to send large quantities of data across an X.25 network.*

The NPI provider translates an interrupt packet received from X.25 into an N_EXDATA_IND message. The interrupt data field of the interrupt packet becomes the M_DATA block in the N_EXDATA_IND message. Once the N_EXDATA_IND has been sent to the user, the NPI provider sends an interrupt confirmation packet to X.25.

Receipt Confirmation

The receipt confirmation feature of NPI allows the NPI user to send an N_DATA_REQ message to the NPI provider and later receive an N_DATAACK_IND when the packet is acknowledged by the remote user. In the opposite direction, the NPI user receives an N_DATA_IND with the receipt confirmation bit set and subsequently sends an N_DATAACK_REQ to the NPI provider. This causes the provider to send an acknowledgment for the packet to X.25.

This scheme is somewhat complicated by the fact that the NPI user can send multiple N_DATA_REQ messages with receipt confirmation requested (hereafter referred to as RC-Data messages) without receiving any N_DATAACK_INDs. The scheme becomes more complicated by the fact that the user can intersperse non-RC-Data in with the RC-data messages.

The NPI provider will actually send up to eight RC-Data messages on the X.25 link before any of them get acknowledged. If the NPI user issues more RC-Data to NPI, then the excess data messages will be queued in the NPI provider's service queue, eventually causing flow control backpressure on the NPI user.

As X.25 acknowledgments come back for RC-Data messages, the NPI provider will send N_DATAACK_IND messages to the user. Acknowledgments for non-RC-data are recognized and do not trigger an N_DATAACK_IND.

In the reverse direction, the NPI provider will send as many as eight RC-Data messages to the user as a result of receiving X.25 data packets with the D-bit set. X.25 acknowledgment of the D-bit packets is withheld until the user issues an N_DATAACK_REQ to the NPI provider. At that point, NPI will acknowledge one D-bit packet, all preceding non-D-bit packets and any subsequent non-D-bit packets up to, but not including, the next outstanding RC-Data message sent to the NPI user.

If NPI receives data packets beyond the eight outstanding RC-Data messages, then the messages are queued rather than forwarded to the user, allowing X.25's window to close.

Packetizing of Data Messages

The data portion of a data message from the NPI User can exceed the negotiated packet size of the X.25 connection. In this case, NPI breaks the data message into a sequence of smaller messages each of which is

the size of a full X.25 packet. It sets the M-bit to 1 and the D-bit to 0 for each of these packets, except for the last one. If the message from the NPI User is not an exact multiple of the X.25 packet size in length then the action on the final packet depends upon the setting of the N_MORE_DATA_FLAG and the N_RC_FLAG. If the N_MORE_DATA_FLAG is 0 then the final fragment is sent with the M-bit set to 0 and the D-bit set to the value of the N_RC_FLAG. If the N_RC_FLAG is set to 1 then the final fragment is sent with the M-bit set to the value of the N_MORE_DATA_FLAG and the D-bit set to 1.

The NPI Provider makes no attempt to accumulate incoming data packets into larger messages to send to the NPI User. Thus, the NPI User receives the packets as they were received by X.25 with the M-bit and D-bit reflected in the DATA_xfer_flags field, if present.

2.5 Option Management

N_INFO_REQ

NPI manages options on a per stream basis by utilizing two message types; the N_INFO_REQ and the N_OPTMGMT_REQ.

The N_INFO_REQ allows the NPI user to interrogate certain parametric information about an NPI connection. The NPI provider responds to an N_INFO_REQ with an N_INFO_ACK. The format of an N_INFO_ACK is as follows:

```
typedef struct
{
    ulong PRIM_type;
    ulong NSDU_size;
    ulong ENSDU_size;
    ulong CDATA_size;
    ulong DDATA_size;
    ulong ADDR_size;
    ulong ADDR_length;
    ulong ADDR_offset;
    ulong QOS_length;
    ulong QOS_offset;
    ulong QOS_range_length;
    ulong QOS_range_offset;
    ulong OPTIONS_flags;
    ulong NIDU_size;
    long  SERV_type;
    ulong CURRENT_state;
    ulong PROVIDER_type;
} N_info_ack_t;
```

The fields are used as follows:

| | |
|-------------------------|--|
| <i>PRIM_type</i> | Set to N_INFO_ACK. |
| <i>NSDU_size</i> | The maximum size of an M_DATA that the user can send to the provider—The NPI Provider sets this value to "unlimited" by default. |
| <i>ENSDU_size</i> | The maximum size of an expedited data unit—This field is set to a constant value of 32. |
| <i>CDATA_size</i> | The amount of data that can be placed in a connect primitive—This field is set to a constant value of 128. |
| <i>DDATA_size</i> | The amount of data that can be placed in a disconnect primitive—This field is set to a constant value of 128. |
| <i>ADDR_size</i> | The maximum number of digits (not bytes) in a network address—This field is set to a constant value of 32. Note, however, that the escaped form of NSAPs, as discussed in “1.2 NSAPs” on page 13, will allow 32 bytes of ASCII characters following the escape code. |
| <i>ADDR_length</i> | The length of the address that was bound to the stream in bytes. |
| <i>ADDR_offset</i> | The offset from the start of the M_PROTO to where the bound address resides—This field will be zero if no bind has occurred yet. The address is in ASCII so an address of 1234 would appear as four bytes 0x31 0x32 0x33 0x34. |
| <i>QOS_length</i> | Always zero. |
| <i>QOS_offset</i> | Always zero. |
| <i>QOS_range_length</i> | Always zero. |
| <i>QOS_range_offset</i> | Always zero. |
| <i>OPTIONS_flags</i> | The call setup flags that were associated with this connection—This field will report the final negotiated settings of the expedited data and receipt confirmation features for the connection. If the connection has not yet been established, this field will be zero. |
| <i>NIDU_size</i> | This field is set to the negotiated X.25 outgoing |

packet size. If the N_INFO_REQ is issued on a stream prior to the establishment of a connection then this field contains the same value as the NSDU_size field.

| | |
|----------------------|--|
| <i>SERV_type</i> | Set to the constant value of N_CONS (connection oriented service). |
| <i>CURRENT_state</i> | The NPI defined state of the connection. |
| <i>PROVIDER_type</i> | Set to the constant value of N_SUBNET. |

N_OPTMGMT_REQ

The NPI message N_OPTMGMT_REQ is concerned with the specifying and negotiating of QOS parameters. It is not supported by the NPI provider and should not be sent by the NPI user.

Section 3

GCOM Extensions to NPI

GCOM has added extensions to the NPI protocol in order to enhance its effectiveness as an interfacing protocol to X.25 and SNA. This section documents these extensions.

3.1 Extensions to the Data Primitives

The *N_data_req_t* and *N_data_ind_t* structures define the structure of an *M_PROTO* that precedes an *M_DATA*. This form of data message is used when it is necessary to convey certain flag bits along with the data contents. The NPI protocol defines two flag bits in the *DATA_xfer_flags* field of these structures as follows:

```
#define N_MORE_DATA_FLAG      0x00000001L      /*indicates that the next NIDU*/
                                   /*is part of this NSDU /
#define N_RC_FLAG            0x00000002L      /* indicates if receipt */
                                   /* confirmation is required */
```

GCOM has added bits to this field to give the user access to the X.25 Q-bit and certain SNA mechanisms. The GCOM extensions to the *DATA_xfer_flags* are as follows (These bits are defined in the file **npixt.h**):

This group defines the bit usage for X.25.

```
#define N_X25_Q_BIT          0x80000000L      /* X.25 Q-bit */
#define N_X25_D_BIT          N_RC_FLAG        /* 0x00000002 in NPI relocated
                                             * from 0x40000000
                                             */
#define N_X25_M_BIT          N_MORE_DATA_FLAG /* 0x00000001 in NPI relocated
                                             * from 0x10000000
                                             */
```

This group defines the bit usage for SNA.

```

#define N_SNA_PRIM      0x80000000    /* SNA Primary emulation/testing */
#define N_SNA_D_BIT     N_RC_FLAG     /* SNA receipt confirmation */
#define N_SNA_FI       0x20000000    /* Format Header 1 data rcvd */
#define N_SNA_MoreChain N_MORE_DATA_FLAG /* SNA not end-chain */
#define N_SNA_ALT_CODE  0x08000000    /* SNA alternate code (RFU) */
#define N_SNA_CDI       0x04000000    /* SNA change direction indicator */
#define N_SNA_BB        0x02000000    /* SNA begin bracket */
#define N_SNA_EB        0x01000000    /* SNA end bracket */
#define N_LU62_CEBI     0x01000000    /* LU62 conditional end bracket */
#define N_SNA_DR2       0x00800000    /* LU62 set DR2 bit */
#define N_LU62_CONF     0x00800000    /* LU62 send confirmation */
#define N_SNA_RSVD      0x00400000    /* Unused flag field */
#define N_SNA_RQE       0x00200000    /* LU62 requests exception Response */
#define N_SNA_FIS       0x00100000    /* first in segment data indicator */
#define N_SNA_SEG       0x00080000    /* last in segment data indicator */
#define N_SNA_LIS       0x00080000    /* last in segment data indicator */
#define N_SNA_FIC       0x00040000    /* first in chain indicator */
#define N_SNA_LIC       0x00020000    /* last in chain indicator */
#define N_SNA_POSR      0x00010000    /* Definite Resp Required */
#define N_SNA_DR1       0x00010000    /* LU62 set DR1 bit */
#define N_SNA_PAD       0x00008000    /* padded data indicator */
#define N_SNA_ENCRYPT    0x00004000    /* encrypted data indicator */
#define N_SNA_COMPRES   0x00002000    /* compressed data indicator */
#define N_SNA_QRI       0x00001000    /* Queued Response Indicator */
#define N_SNA_MIS       0x00000000    /* middle in segment */
#define N_SNA_MIS       0x00000000    /* middle in segment */

```

3.2 Extensions to Bind Request

GCOM has extended the NPI bind request primitive in two different ways. First, additional bind flags have been introduced which control some of the per-connection features of NPI. Second, extended forms of the bind request message have been defined to provide additional call routing functionality.

EXTENDED BIND FLAGS

The GCOM extensions to the `BIND_flags` field apply to the standard NPI bind request as well as to all of the extended forms. The following table describes the extended bind flags as defined in `<npiext.h>`.

START_FC_ZERO Start flow control at zero. See `N_FLOW_REQ`.

USE_EXT_CONN_IND

Send `N_EXT_CONN_IND` for incoming calls.

USE_EXT_CONN_CON Send `N_EXT_CONN_CON`.

USE_ASCII_NSAP Always encode NSAPs in ASCII.

INHIBIT_PKTIZE Do not break outbound NPI messages into packets. Forward data as-is regardless of packet size.

USE_EXT2_CONN_IND

Send `N_EXT2_CONN_IND` for incoming calls. Takes precedence over `USE_EXT_CONN_IND`.

USE_EXT2_CONN_CON

Send `N_EXT2_CONN_CON`. Takes precedence over `USE_EXT_CONN_CON`.

N_EXT_BIND_REQ

The form of the N_EXT_BIND_REQ sent to the NPI provider is as follows:

```
typedef struct
{
    Nuns32 PRIM_type;                /*always N_EXT_BIND_REQ*/
    Nuns32 ADDR_length;             /*length of address*/
    Nuns32 ADDR_offset;            /*offset of address*/
    Nuns32 CONIND_number;          /*requested # of connect-*/
                                   /*indications to be queued*/
    Nuns32 BIND_flags;             /*bind flags*/
    /*
     * Gcom extensions
     */
    Nuns32 REM_length;              /* length of remote addr */
    Nuns32 REM_offset;             /* offset to remote addr */
    Nuns32 LPA_number;             /* LPA to listen on */
} N_ext_bind_req_t;
```

The non-extended fields are used in the same manner as in the N_BIND_REQ primitive.

The extended fields are as follows:

| | |
|-------------------|--|
| <i>REM_length</i> | The number of bytes in the NSAP contained within the M_PROTO message. If present, this remote NSAP is saved and used to filter incoming calls that would otherwise be directed to this stream. In order for this stream to receive the call not only must the X.25 called address match the NSAP indicated by the ADDR_length and offset but the X.25 calling address must match the NSAP indicated by the REM_length and offset. This NSAP may contain wildcard characters. |
| <i>REM_offset</i> | The byte offset from the beginning of the M_PROTO where the NSAP begins. An offset of zero, accompanied by a REM_length of zero, indicates that no NSAP is present. |
| <i>LPA_number</i> | The LPA from which an incoming call must be received in order to be routed to this stream. The value of zero matches any LPA. |

N_EXT2_BIND_REQ

The form of the N_EXT2_BIND_REQ sent to the NPI provider is as follows:

```
typedef struct
{
    Nuns32 PRIM_type;                /*always N_EXT2_BIND_REQ*/
    Nuns32 ADDR_length;             /*length of address*/
    Nuns32 ADDR_offset;            /*offset of address*/
    Nuns32 CONIND_number;          /*requested # of connect-*/
    Nuns32 CONIND_indications;     /*indications to be queued*/
    Nuns32 BIND_flags;             /*bind flags*/
    /*
     * Gcom extensions
     */
    Nuns32 REM_length;             /* length of remote addr */
    Nuns32 REM_offset;            /* offset to remote addr */
    Nuns32 LPA_number;            /* LPA to listen on */
    Nuns8  DATA_val[16];         /* value of data field */
    Nuns8  DATA_mask[16];       /* mask bits for value */
} N_ext2_bind_req_t;
```

The non-extended fields are used in the same manner as in the N_BIND_REQ primitive.

The extended fields are as follows:

| | |
|-------------------|---|
| <i>REM_length</i> | Same as for N_EXT_BIND_REQ. |
| <i>REM_offset</i> | Same as for N_EXT_BIND_REQ. |
| <i>LPA_number</i> | Same as for N_EXT_BIND_REQ. |
| <i>DATA_val</i> | An array of 16 bytes that are compared to the bytes of the incoming call packet's user data field. Each byte of the DATA_val array is ANDed with the corresponding byte from the DATA_mask array. Each byte of the incoming call user data field is also ANDed with the corresponding byte of the DATA_mask array. The two derived bytes are then compared. If all bytes in the user data field match for up to 16 bytes, or the length of the user data field, whichever is smaller, then the call can be routed to this stream if all other criteria are met. A zero length user data field matches all patterns. A DATA_mask of all zeros will cause every user data field to match. |

DATA_mask Bit mask array.

3.3 Extensions to Connection Primitives

GCOM has extended the connection primitives of NPI so that X.25 facilities can be passed between the application and NPI. The NPI primitives that are extended are the *N_CONN_REQ*, *N_CONN_RES*, *N_CONN_IND*, and *N_CONN_CON* primitives. Each of these primitives has been extended in the same way, namely, each structure has had the fields *FAC_length* and *FAC_offset* added to them. These fields specify the offset from the beginning of the *M_PROTO* of the *facilities* field and the length of that field. The facilities appear in CCITT binary form just as specified in the X.25 recommendation.

The extended primitives have unique encodings so that they can be distinguished from the ordinary NPI primitives. The extended primitives are of type *N_EXT_CONN_REQ*, *N_EXT_CONN_RES*, *N_EXT_CONN_IND*, and *N_EXT_CONN_CON*, respectively.



Note: *These primitives are sent by the user to NPI.*

The structures described in the following subsections, which are declared in **npixt.h**, define the extended primitives.

N_EXT_CONN_REQ

The form of the N_EXT_CONN_REQ sent to the NPI provider is as follows:

```
typedef struct
{
    Nuns32 PRIM_type;           /* always N_EXT_CONN_REQ */
    Nuns32 DEST_length;        /* destination address length */
    Nuns32 DEST_offset;        /* destination address offset */
    Nuns32 CONN_flags;          /* bit masking for options flags */
    Nuns32 QOS_length;          /* length of QOS parameter values */
    Nuns32 QOS_offset;          /* offset of QOS parameter values */
    /*
     * Gcom extensions
     */
    Nuns32 FAC_length;          /* length of facilities */
    Nuns32 FAC_offset;          /* offset to facilities */
} N_ext_conn_req_t;
```

This primitive is sent from the NPI user to the NPI provider to initiate a connection. The non-extended fields are used in the same manner as in the N_CONN_REQ primitive. The extended fields are as follows:

| | |
|-------------------|--|
| <i>FAC_length</i> | The number of bytes of facilities contained within the M_PROTO. The facilities are in the form specified by the CCITT in recommendation X.25. The length indicator is contained in this field and is not present in the facilities portion of the M_PROTO. A <i>FAC_length</i> of zero indicates that no facilities are present. |
| <i>FAC_offset</i> | The byte offset from the beginning of the M_PROTO where the facilities begin. An offset of zero, accompanied by a <i>FAC_length</i> of zero, indicates that no facilities are present. |

N_EXT_CONN_RES

The form of the N_EXT_CONN_RES sent to the NPI user is as follows:

```
typedef struct
{
    Nuns32 PRIM_type;           /* always N_EXT_CONN_RES */
    Nuns32 TOKEN_value;       /* NC response token value */
    Nuns32 RES_length;        /* responding address length */
    Nuns32 RES_offset;        /* responding address offset */
    Nuns32 SEQ_number;        /* sequence number */
    Nuns32 CONN_flags;        /* bit masking for options flags */
    Nuns32 QOS_length;        /* length of QOS parameter values */
    Nuns32 QOS_offset;        /* offset of QOS parameter values */
    /*
    * Gcom extensions
    */
    Nuns32 FAC_length;        /* length of facilities */
    Nuns32 FAC_offset;        /* offset to facilities */
} N_ext_conn_res_t;
```

This primitive is sent from the NPI user to the NPI provider to respond to an N_CONN_IND. The non-extended fields are used in the same manner as in the N_CONN_RES primitive. The extended fields are as follows:

FAC_length The number of bytes of facilities contained within the M_PROTO. The facilities are in the form specified by the CCITT Recommendation X.25. The length indicator is contained in this field and is not present in the facilities portion of the M_PROTO. A *FAC_length* of zero indicates that no facilities are present.

FAC_offset The byte offset from the beginning of the M_PROTO where the facilities begin. An offset of zero, accompanied by a *FAC_length* of zero, indicates that no facilities are present.

N_EXT_CONN_IND

The form of the N_EXT_CONN_IND sent to the NPI user is as follows:

```
typedef struct
{
    Nuns32 PRIM_type;           /* always N_EXT_CONN_IND */
    Nuns32 DEST_length;        /* destination address length */
}
```

```

    Nuns32 DEST_offset;           /* destination address offset */
    Nuns32 SRC_length;           /* source address length */
    Nuns32 SRC_offset;           /* source address offset */
    Nuns32 SEQ_number;           /* sequence number */
    Nuns32 CONN_flags;           /* bit masking for options flags */
    Nuns32 QOS_length;           /* length of QOS parameter values */
    Nuns32 QOS_offset;           /* offset of QOS parameter values */
/*
 * Gcom extensions
 */
    Nuns32 FAC_length;           /* length of facilities */
    Nuns32 FAC_offset;           /* offset to facilities */
} N_ext_conn_ind_t;

```

This primitive is sent from the NPI provider to the NPI user to indicate that an incoming connection request has been received. The non-extended fields are used in the same manner as in the N_CONN_IND primitive. The extended fields are as follows:

| | |
|-------------------|---|
| <i>FAC_length</i> | The number of bytes of facilities contained within the M_PROTO. The facilities are in the form specified by the CCITT Recommendation X.25. The length indicator is contained in this field and is not present in the facilities portion of the M_PROTO. A <i>FAC_length</i> of zero indicates that no facilities are present. |
| <i>FAC_offset</i> | The byte offset from the beginning of the M_PROTO where the facilities begin. An offset of zero, accompanied by a <i>FAC_length</i> of zero, indicates that no facilities are present. |

N_EXT2_CONN_IND

The form of the N_EXT2_CONN_IND sent to the NPI user is as follows:

```

typedef struct
{
    Nuns32 PRIM_type;           /* always N_EXT2_CONN_IND */
    Nuns32 DEST_length;         /* destination address length */
    Nuns32 DEST_offset;         /* destination address offset */
    Nuns32 SRC_length;           /* source address length */
    Nuns32 SRC_offset;           /* source address offset */
    Nuns32 SEQ_number;           /* sequence number */
    Nuns32 CONN_flags;           /* bit masking for options flags */
    Nuns32 QOS_length;           /* length of QOS parameter values */

```

```

Nuns32 QOS_offset;          /* offset of QOS parameter values */
/*
 * Gcom extensions
 */
Nuns32 FAC_length;        /* length of facilities */
Nuns32 FAC_offset;       /* offset to facilities */
Nuns32 LPA_number;       /* LPA of interface that received the call */

} N_ext2_conn_ind_t;

```

This primitive is sent from the NPI provider to the NPI user to indicate that an incoming connection request has been received. The non-extended fields are used in the same manner as in the N_CONN_IND primitive. The extended fields are as follows:

| | |
|-------------------|--|
| <i>FAC_length</i> | Same as for N_EXT_CONN_IND. |
| <i>FAC_offset</i> | Same as for N_EXT_CONN_IND. |
| <i>LPA_number</i> | The Lower Point of Attachment (LPA) number of the originator of the X.25 incoming call packet. This is a number from 1 to n and generally represents the line number of the X.25 protocol stack that received the incoming call and sent it upstream to NPI. |

N_EXT_CONN_CON



Note: *This primitive has not been implemented as of the current release of NPI.*

The form of the N_EXT_CONN_CON sent to the NPI user is as follows:

```
typedef struct
{
    Nuns32 PRIM_type;           /* always N_EXT_CONN_CON */
    Nuns32 RES_length;         /* responding address length */
    Nuns32 RES_offset;        /* responding address offset */
    Nuns32 CONN_flags;         /* bit masking for options flags */
    Nuns32 QOS_length;         /* length of QOS parameter values */
    Nuns32 QOS_offset;        /* offset of QOS parameter values */
    /*
     * Gcom extensions
     */
    Nuns32 FAC_length;         /* length of facilities */
    Nuns32 FAC_offset;        /* offset to facilities */
} N_ext_conn_con_t;
```

This primitive is sent from the NPI provider to the NPI user to indicate that an outgoing connection request has been accepted by the remote peer. The non-extended fields are used in the same manner as in the N_CONN_IND primitive. The extended fields are as follows:

| | |
|-------------------|---|
| <i>FAC_length</i> | The number of bytes of facilities contained within the M_PROTO. The facilities are in the form specified by the CCITT Recommendation X.25. The length indicator is contained in this field and is not present in the facilities portion of the M_PROTO. A <i>FAC_length</i> of zero indicates that no facilities are present. |
| <i>FAC_offset</i> | The byte offset from the beginning of the M_PROTO where the facilities begin. An offset of zero, accompanied by a <i>FAC_length</i> of zero, indicates that no facilities are present. |

N_EXT2_CONN_CON

The form of the N_EXT2_CONN_CON sent to the NPI user is as follows:

```
typedef struct
{
    Nuns32 PRIM_type;           /* always N_EXT2_CONN_CON */
    Nuns32 RES_length;         /* responding address length */
    Nuns32 RES_offset;        /* responding address offset */
    Nuns32 CONN_flags;         /* bit masking for options flags */
    Nuns32 QOS_length;         /* length of QOS parameter values */
    Nuns32 QOS_offset;        /* offset of QOS parameter values */
    /*
     * Gcom extensions
     */
    Nuns32 FAC_length;         /* length of facilities */
    Nuns32 FAC_offset;        /* offset to facilities */
    Nuns32 LPA_number;         /* LPA of interface that received the call */
} N_ext2_conn_con_t;
```

This primitive is sent from the NPI provider to the NPI user to indicate that an outgoing connection request has been accepted by the remote peer. The non-extended fields are used in the same manner as in the N_CONN_IND primitive. The extended fields are as follows:

| | |
|-------------------|--|
| <i>FAC_length</i> | Same as for N_EXT_CONN_CON. |
| <i>FAC_offset</i> | Same as for N_EXT_CONN_CON. |
| <i>LPA_number</i> | The Lower Point of Attachment (LPA) number of the X.25 interface to which the original call request packet was directed. This is a number from 1 to n and generally represents the line number of the X.25 protocol stack to which the call request packet was directed as a result of NPI's call routing. |

3.4 Flow Control Primitives

The NPI Provider uses the following primitive as a means of regulating the flow of inbound messages between itself and the NPI User. This structure definition can be found in the file <npiext.h>.

```
typedef struct
{
    Nuns32 PRIM_type;           /* always N_FLOW_REQ */
    Nuns32 flow_incr;         /* flow control increment */
} N_flow_req_t;
```

The values of the flow_incr field can be any 16 bit number plus the following two special values:

```
#define NP_FC_ZERO          0x10000 /* set level to zero */
#define NP_FC_INFINITE     0x1FFFF /* set level to infinite */
```

The default value, used for streams on which this primitive is never sent, is NP_FC_INFINITE.

This primitive is used to set the level of in-bound flow control in terms of messages. It is sent from the user to the NPI provider and is unconfirmed.

This message can be sent at any time during the lifetime of an NPI connection. It does not change the state of the connection.

The level set prior to a reset exchange remains in place after the reset exchange.

The value conveyed in the message is an increment for the flow control level maintained by the NPI provider. Each time the provider sends a message upstream it decrements its level. When the level reaches zero, the provider begins queuing messages causing flow control back-pressure to propagate to the distant end of the connection.

Two distinguished values can be passed down in the flow control req. NP_FC_ZERO forces the level to zero, in effect instantly shutting off the flow of incoming data to the user. It should be noted, however, that a number of messages may have already been sent from the provider to the user at the time the provider receives the N_FLOW_REQ. Those messages will be delivered to the user. Further messages will be queued.

The value of `NP_FC_INFINITE` sets the flow control level to infinite. The NPI provider never decrements this value. The user is at liberty to change the level to a finite value or to `NP_FC_ZERO` at any future time.

The user can pursue any of three strategies in managing this flow control.

The first strategy is to ignore the whole issue. The connection starts off with infinite flow control by default (but see note on bind options). The user can effect flow control back-pressure by not reading the stream.

Not reading the stream, however, means that the user also cannot read `N_DATAACK_IND` messages which the NPI provider sends to the user to acknowledge `N_DATA_REQ` messages sent by the user to the provider with the receipt confirmation requested.

So, the second strategy is to keep the flow control at infinite until the user decides not to accept "any more" data. At that time the user can send an `N_FLOW_REQ` with the value `NP_FC_ZERO`. Of course, an indeterminate number of messages may be in transmit from the provider to the user and the user must handle these somehow. If the computer is much faster than the line speed then the number of such messages is likely to be small.

The third strategy is to send an authorization level of 'n' after binding or connecting the stream. The user keeps track of how many messages have been received and renews the authorization when the number of authorized messages becomes too small. By setting the level to 1, the user can impose a one-at-a-time discipline on inbound message traffic at the expense of efficiency.

For purposes of counting, an `N_DATA_IND` with the more data indicator set counts as one message. An incompletely read message does not count as one message. That is, if the application is using `npi_rcv` to receive inbound messages and if `npi_rcv` returns the bit `NPIAPI_FRAGMENT` in the output flags, then a partial message has been read and is not to be counted against flow control levels.

3.5 Draining Data

Ordinarily an `N_DISC_REQ` will cause any outbound queued data to be flushed by NPI prior to sending an X.25 clear request packet. This special primitive can be used to drain queued data prior to disconnecting, or at any other time. The primitive has the following form, defined in `<npiext.h>`:

```
typedef struct
{
    Nuns32 PRIM_type;           /* always N_DRAIN_REQ
*/
    Nuns32 drain_option;      /* type of drain condition */
} N_drain_req_t ;
```

The `drain_option` field is set to one of the following values:

N_DRAIN_IMMED Acknowledge immediately.

N_DRAIN_SENT Acknowledge after all data packets have been sent from NPI to the X.25 protocol module, but not necessarily sent on the access line.

N_DRAIN_ACKD Acknowledge after all data packets have been sent and acknowledged by X.25.

The NPI Provider acknowledges this primitive by sending an `N_OK_ACK` at the appropriate time. The `CORRECT_prim` field will be set to `N_DRAIN_REQ`.

Index

a

angle bracket conventions 9

b

boldface text conventions 9

c

cautions, purpose of 8

conventions

 notes, cautions and warnings 8

 text 9

e

enter vs. type 9

i

italic text conventions 9

n

notes, purpose of 8

s

screen display 9

t

terminology conventions 9

text conventions 9

type vs. enter 9

w

warnings, purpose of 8

