DLPI
User Manual

February 2002

Protocols: LAPB, MLP, QLLC, HDLC/SDLC,
LAPD, and frame relay

This manual was written, formatted and produced by senior technical writer Scott D. Smith using Microsoft Word 5.1 and FrameMaker 4.04 on an Apple Macintosh platform with the help of subject matter specialists Dave Healy and illustrator and publication specialist Charles Lipp.

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

Table of Contents

PREFACE

Purpose of This Guide

This implementation of the Data Link Provider Interface (DLPI) provides connection oriented access to several data link protocols. It supports connection establishment, flow-controlled data transfer, a re-synchronization mechanism and connection teardown.

Knowledge Requirements

The discussion in this manual assumes the reader is familiar with the DLPI protocol[1] and has some understanding of ATT STREAMS [2]. Some knowledge of the Network Management Interface [3] described in GCOM's *UNIX Streams Administrator's Guide* is useful but not required.

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

For information about:	*Look at:*
Overview of the DLPI multiplexor	Section 1
Using the DLPI protocol and the DLPI implementation	Section 2
DLPI protocol extensions	Section 3
Document references cited in earlier sections	Appendix A

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

DLPI Overview

An overview of the DLPI multiplexor architecture is presented. The intent of the discussion is that the user understand addressing used to establish a DLPI connection and gain a feel for the protocol processing that occurs on a data message.

The DLPI is implemented as a STREAMS multiplexor. A network manager, using the network management interface, is responsible for configuring the DLPI multiplexor. The DLPI user has only limited capability to alter the configuration of a link that the user has bound. See “ioctls” on page 22.

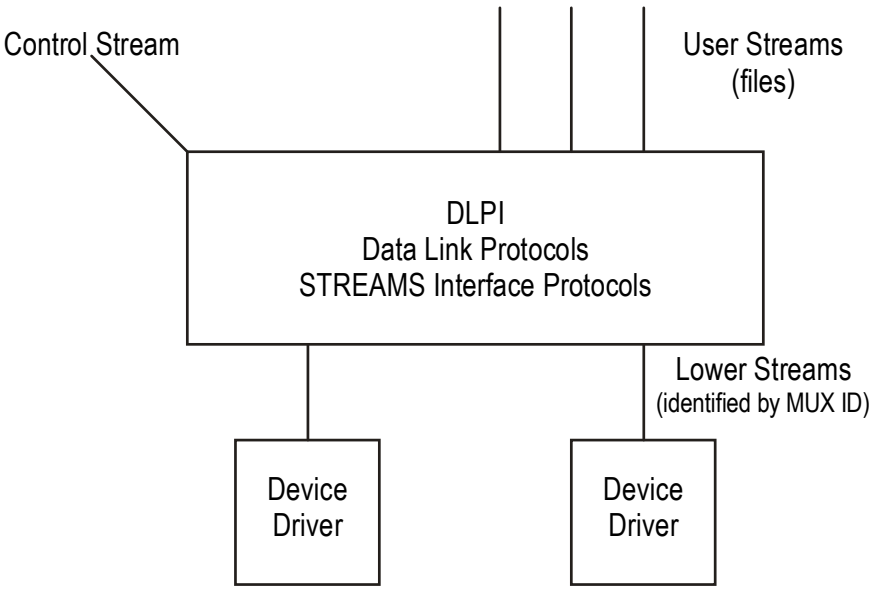

Table 1 Streams Environment

The DLPI multiplexor consists of the DLPI, data link protocols and streams interface protocols. The DLPI is an interface protocol, giving

the STREAMs user access to a data link protocol. The available data link protocols, depending on the configuration, are HDLC, X.25 frame level, and X.25 MLP. The data link protocols do not directly access devices. Instead, a lower streams interface protocol is used to access a STREAMS driver which provides the device access. The lower streams interface protocol must speak the streams protocol expected by the STREAMS device driver. DLPI user protocol and CDI user protocol are supported.

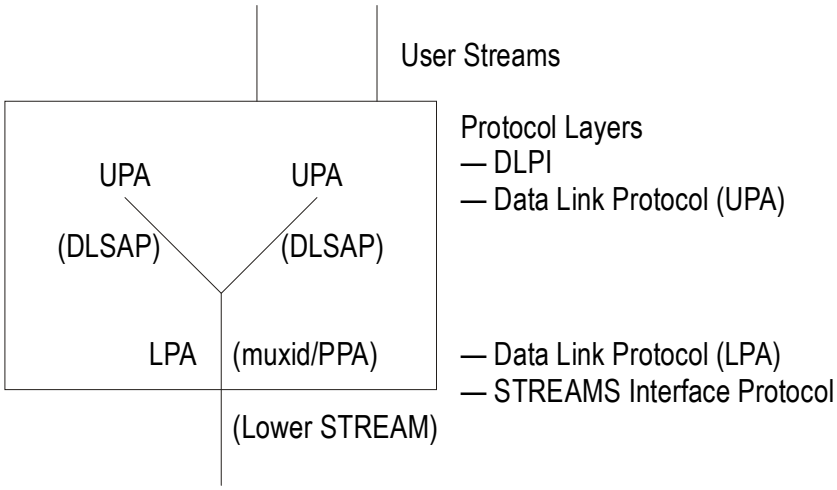

Table 2 Physical Point of Attachment (PPA)/Data Link Service Access Point (DLSAP)

Two layers of data link protocols are supported. In the GCOM *UNIX Streams Administrator's Guide*, these two layers are referred to as Lower Point of Attachment (LPA), and Upper Point of Attachment (UPA). In general, the LPA provides (possibly multiplexed) access to the Physical Point of Attachment (PPA) [ref. 1]. The UPA provides a data link protocol, such as X.25 MLP, HDLC, and X.25 frame level. The LPA protocol can either be a link multiplexor (that is, POLL_MODULE) or a null layer (that is, NULL_MODULE). For the NULL_MODULE, the only supported data link protocol is MLP, which provides its own “device” access.

Before a user can access the DLPI multiplexor, the multiplexor must be configured. Network management facilities, using the control stream, would:

1) For each device, create a stack of required STREAMS drivers, and I_LINK the stack under the multiplexor. The multiplexor id returned in the I_LINK is used to specify the physical point of attachment (PPA) in a DLPI DL_ATTACH_REQ request.

2) For each I_LINK'd lower stream, configure the access point specifying interface protocol and data link protocols. This occurs in two steps. First, the LPA is configured specifying a streams interface protocol and an optional data link protocol. Then one or more UPAs are configured. A configured UPA has a data link protocol module and a logical address called Data Link Service Access Point (DLSAP). The DLSAP is relative to the PPA and specifies a data link provided over the PPA. For each UPA, network management may specify the DLSAP or have it defaulted appropriate to the configuration.

SECTION 2

Using the DLPI Protocol

The following discussion describes how the user establishes a connection, transfers data, re-synchronizes a connection and releases a connection. Those DLPI features not supported by the DLPI multiplexor are also described. An understanding of the DLPI protocol as described in [1] will make the following discussion easier to comprehend. Also discussed is the handling of fatal errors, limits on the number of STREAMS buffers that can be in use for a given stream and the generation of acknowledgments.

Establishing a Connection

To establish a connection, the user must first open a stream to the DLPI multiplexor. A STREAM file is similar to a character special device file. The open must specify a DLPI STREAMs file with a specific non-zero minor device or a CLONEOPEN. No additional privilege checks are performed by the DLPI multiplexor. The DLPI multiplexor allows a minor device to be opened multiple times.

DLPI protocol messages are exchanged between the user and the DLPI provider using STREAMs control messages. DLPI data messages, described as DLSDUs below, are exchanged in STREAMs data messages. The UNIX system calls *putmsg* and *getmsg* can be used to send and receive control and data messages. Write and read can also be used to send and receive data messages.

Once a stream has been opened, the user must specify a PPA and a DLSAP relative to the PPA. These addresses are specified using the DLPI DL_ATTACH_REQ and DL_BIND_REQ DLPI requests. Once these addresses have been established, the user can request that a connection be established or wait for a connection indication. This choice is indicated in a DL_BIND_REQ request. A connection is requested using a DL_CONNECT_REQ. A connection indication is accepted using a DL_CONNECT_RES.

DL_ATTACH_REQ

```
typedef struct
{
     ulong     dl_primitive;
     ulong     dl_ppa;
} dl_attach_req_t;
```

This structure is used to submit an attach request. This request is valid only in the DL_UNATTACHED state, which is the initial state after a successful open.

dl_primitive Set to DL_ATTACH_REQ

dl_ppa Identifies the physical point of attachment—*dl_ppa* is set to the *muxid* of the I_LINK'd lower stream. Multiple users may attach the same PPA. A DL_ATTACH_REQ succeeds even if no data link protocol has been configured. If the UPA is a multiplexor (for example, MLP), any of its LPAs may be attached. An unconfigured PPA causes either the bind request or a connect request to fail. A successful attach results in the user's state being changed to DL_UNBOUND.

DL_BIND_REQ

```
typedef struct
{
     ulong      dl_primitive ;
     ulong      dl_sap ;
     ulong      dl_max_conind ;
     ushort     dl_service_mode ;
     ushort     dl_comm_mgmt ;
} dl_bind_req_t;
```

This structure is used to submit a bind request. A bind request specifies a service access point, whether or not the user will accept a DL_CONNECT_IND, whether or not a connection oriented service is desired, or whether or not the user is a connection manager.

dl_primitive Set to DL_BIND_REQ—This request in valid only in the DL_UNBOUND state. If successful, the user's state is changed to DL_IDLE.

dl_sap A DLSAP described above—It specifies an address to be used when selecting a data link. A DLSAP is implemented as an unstructured array of up to 7

bytes. A log is converted to a DLSAP by finding the high-order non-zero byte and copying that byte and remaining lower-order bytes into the *dlsap*. Zero is a legal *dl_sap* and results in a 1 byte DLSAP.

Note: *The extended address bit (bit 0) is not used to interpret the address.*

dl_max_conind Maximum number of DL_CONNECT_IND the user will accept—A value greater than 1 implies that the user is a connection manager (see *dl_comm_mgmt* below). Connection managers are not supported, and a bind request with a *dl_max_conind* greater than 1 will be rejected.

A *dl_max_conind* of 1 indicates the user will wait for a DL_CONNECT_IND. When a DL_CONNECT_IND arrives, the user can accept the connection with a DL_CONNECT_RES or reject the connection with a DL_DISCONNECT_REQ. If the user specifies a *dl_max_conind* of 1, the user may not initiate a connection with a DL_CONNECT_REQ. It will be rejected.

A *dl_max_conind* of zero indicates that the user does not want to receive a DL_CONNECT_IND. Instead, the user can initiate a connection by sending a DL_CONNECT_REQ. When the connection is established, a DL_CONNECT_CON will be returned to the user.

Connect requests crossing in the mail are supported. If the user and the user's opposite concurrently initiate a connection, the connection will be established.

dl_service_mode Specifies whether connection-oriented (DL_CODLS) or connectionless (DL_CLDLS) data link service is desired—Only DL_CODLS is supported.

dl_comm_mgmt If non-zero, indicates the user is a connection manager. This DLPI facility is not supported.

DL_CONNECT_REQ

```
typedef struct
{
     ulong     dl_primitive ;
     ulong     dl_dest_addr_length ;
     ulong     dl_dest_addr_offset ;
     ulong     dl_qos_length ;
     ulong     dl_qos_offset ;
     ulong     dl_growth ;
} dl_connect_req_t ;
```

A connect request is used to initiate a connection. The user's stream must be in the DL_IDLE state. The bind request must have specified a *dl_max_conind* of zero. If the request is in error, a DL_ERROR_ACK is returned. The user's state is changed to DL_OUTCON_PENDING.

If the attempt to establish a connection fails, the user is sent a DL_DISCONNECT_IND and returned to the DL_IDLE state. Otherwise, after the connection is established, the user will be sent a DL_CONNECT_CON and placed in the DL_DATAXFER state.

dl_primitive	Set to DL_CONNECT_REQ
dl_dest_addr_length, *dl_dest_addr_offset*	These two fields identify the peer's DLSAP—If *dl_dest_addr_length* is non zero, the array referenced by *dl_dest_addr_offset* and *dl_dest_addr_length* must be within the user's request buffer. These fields are otherwise ignored.
dl_qos_length, *dl_qos_offset*	Specifies the quality of service (QOS) parameters desired for the connection. If specified, the QOS parameters must be within the user's request buffer. QOS is not supported and these fields are otherwise ignored.
dl_growth	A field reserved for future use that must be set to zero.

DL_CONNECT_CON

This message is sent to the user when a previously requested connection (see “DL_CONNECT_REQ” on page 7) is established. The user’s state is changed to DL_DATAXFER.

```
typedef struct
{
     ulong     dl_primitive ;
     ulong     dl_resp_addr_length ;
     ulong     dl_resp_addr_offset ;
     ulong     dl_qos_length ;
     ulong     dl_qos_offset ;
     ulong     dl_growth ;
} dl_connect_con_t ;
```

dl_primitive	Set to DL_CONNECT_CON
dl_resp_addr_length, *dl_resp_addr_offset*	These two fields are used to return the responding DLSAP associated with the newly established connection. This is the *dl_upr_prm*. *dl_bind_dlsap* specified in a network management upper configuration request. *dl_resp_addr_offset* is an offset into the user’s buffer indicating where the DLSAP is located. *dl_resp_addr_length* is the number of bytes required for the DLSAP.
dl_qos_length, *dl_qos_offset*	These two fields are null because Quality of Service is not supported.
dl_growth	Zero

DL_CONNECT_IND

A DL_CONNECT_IND is sent to the user to indicate that the user's opposite is attempting to establish a connection. The user should respond with a DL_CONNECT_RES to accept the connection or a DL_DISCONNECT_REQ to reject the connection. These two user responses are described below. The user's state is changed to DL_INCON_PENDING.

```
typedef struct
{
     ulong     dl_primitive ;
     ulong     dl_correlation ;
     ulong     dl_called_addr_length ;
     ulong     dl_called_addr_offset ;
     ulong     dl_calling_addr_length ;
     ulong     dl_calling_addr_offset ;
     ulong     dl_qos_length ;
     ulong     dl_qos_offset ;
     ulong     dl_growth ;
} dl_connect_ind_t ;
```

dl_primitive	Set to DL_CONNECT_IND
dl_correlation	Set to zero
dl_called_addr_length, *dl_called_addr_offset*	Specifies the location within the user's buffer of a DLSAP—This is the DLSAP specified in the user's DL_BIND_REQ request.
dl_calling_addr_length, *dl_calling_addr_offset*	Specifies the location within the user's buffer of a DLSAP—This is the *dl_peer_dlsap* specified when the UPA was configured.
dl_qos_length, *dl_qos_offset*	Both these fields are null as Quality of Service is not supported.
dl_growth	Set to zero.

DL_CONNECT_RES

This DLPI protocol message is sent by the user to accept a connection indication. The user must be in the DL_INCON_PENDING state. If the request is not in error, the user's state is changed to DL_DATAXFER and sent a DL_OK_ACK. The user may not start sending data until the DL_OK_ACK is received. If the user's request is in error, a DL_ERROR_ACK is returned.

```
typedef struct
{
     ulong     dl_primitive ;
     ulong     dl_correlation ;
     ulong     dl_resp_token ;
     ulong     dl_qos_length; ;
     ulong     dl_qos_offset ;
     ulong     dl_growth ;
} dl_connect_res_t ;
```

dl_primitive	Set to DL_CONNECT_RES
dl_correlation	Must be zero.
dl_resp_token	Must be zero.
dl_qos_length, *dl_qos_offset*	Specifies the quality of service (QOS) parameters desired for the connection—If specified, the QOS parameters must be within the user's request buffer. QOS is not supported and these fields are otherwise ignored.
dl_growth	Must be zero.

Data Transfer

Data may be transferred simultaneously in both directions. Data is transferred in records called data link service data units (DLSDU). The integrity of the DLSDU is maintained by the DLPI multiplexor. DLSDUs are delivered in order, no missing, no duplicates and bit error free.

Data delivery is flow-controlled. Downstream (user to data link) data transfer is flow-controlled using the STREAM's *canput()* flow-control mechanism. When the data link protocol back pressures DLPI, data messages are queued on the user's STREAM's write queue until the flow-control blockage is removed. Upstream (data link to user) data transfer is similarly flow-controlled. *canput()* is used to determine the amount of data that can be delivered upstream. If upstream data transfer is flow-control blocked, the data is queued internally until the blockage is removed. In addition, the flow-control blockage is propagated into the data link protocol module.

The connection must be in the DL_DATAXFER state when the user transmits data. Violating this restriction will result in an M_ERROR being sent to the user and all user messages being discarded until the connection is closed. However, it's possible for the DLPI to have left the DL_DATAXFER state but for the user to have not yet received any messages indicating the state had changed. This can happen when a DLRESET_IND or a DL_DISCONNECT_IND is sent to the user. Therefore, data messages received in the DL_IDLE, DL_PROV_RESET_PENDING or DL_DISCON_IND_PENDING state will be silently discarded by the DLPI multiplexor.

The DLPI specification places limits on the size of data messages the user can send to the DLPI multiplexor. These are *min_sdu* and *max_sdu*. These are the minimum and maximum number of bytes that can be transmitted in a single data message. These values are set when the data link is configured and can be found in a DL_INFO_ACK. Violating these limits results in an M_ERROR being sent to the user.

Reset Mechanism

The DLPI provides a reset mechanism to report the unrecoverable loss of data and to enable users to recover from errors that cause the user to get out of sync with its peer. That is, the reset mechanism enables peers to restart their dialogue after such errors. DLPI accomplishes this by causing the communication path between the two peers to be flushed and informing the peers when their dialogue may be resumed. If the user's peer-to-peer protocol does not support a re-synchronization event, the user should probably disconnect to prevent a reception of a DL_RESET_IND. For the re-synchronization mechanism to work correctly, the user's peer's interface to the link level must follow the DLPI standard.

If the local user initiates the reset, the user should discard any data received before a DL_RESET_CONFIRM is received. That is, depending on the STREAMs implementation, it might be possible for a data message to cross in the mail with the user's DL_RESET_REQ message.

Another crossing in the mail case is when the DLPI receives a reset indication from its data link protocol module and sends a DL_RESET_IND to the user. The user could simultaneously send a DL_RESET_REQ. The DLPI provider would receive this request in the DL_PROVIDER_RESET_PENDING state. The DLPI State Transition Table treats this as a "cannot occur" transition [ref. 1, pages 87 - 90]. Therefore, the DLPI provider returns a DL_ERROR_ACK. Since the user would have received the DL_RESET_IND, the DL_ERROR_ACK should be ignored and the user should proceed as if a DL_RESET_REQ were never sent. That is, send a DL_RESET_RES and wait for a DL_OK_ACK.

While the connection is being reset, the user may disconnect at any time or possibly receive a DL_DISCONNECT_IND. If the user disconnects, the reset operation is aborted and a DL_OK_ACK referencing the disconnect request is returned to the user. The disconnect causes all internally queued messages to be flushed and a flush request to be sent to the streamhead. That is, there cannot be a previously queued DL_OK_ACK that would cause the DL_OK_ACK to be discarded.

DL_RESET_REQ

```
typedef struct
{
     ulong        dl_primitive ;
} dl_reset_req_t ;
```

Sent by the DLPI user to request an end-to-end re-synchronization. The user's connection must be in the DL_DATAXFER state or a DL_ERROR_ACK is returned. As described above, this request can cross in the mail with a DL_RESET_IND or a DL_DISCONNECT_IND.

dl_primitive DL_RESET_REQ

DL_RESET_IND

```
typedef struct
{
     ulong        dl_primitive ;
     ulong        dl_originator ;
     ulong        dl_reason ;
} dl_reset_ind_t ;
```

Generated by the DLPI provider to report the unrecoverable loss of data or a reset request by its opposite.

dl_primitive DL_RESET_IND

dl_originator Set to DL_PROVIDER

dl_reason. Set to DL_RESET_LINK_ERROR—The DLPI provider cannot determine what actually initiated this event.

DL_RESET_RES

```
typedef struct
{
     ulong        dl_primitive ;
} dl_reset_res_t ;
```

This message is sent by the DLPI user to indicate it is ready to resume a dialogue with its peer. The DLPI provider responds by sending a DL_OK_ACK to the user and completing the reset exchange with the data link layer. When the user receives the DL_OK_ACK referencing the DL_RESET_RES, the user may begin sending data. Any data received by the user after receiving the DL_OK_ACK was sent by its peer after the peer completed its reset operation.

dl_primitive Set to DL_RESET_RES.

DL_RESET_CON

```
typedef struct
{
     ulong        dl_primitive ;
} dl_reset_con_t ;
```

This message is sent by the DLPI in response to a DL_RESET_REQ. It is sent after the reset operation is complete. This message has the same semantics as a DL_OK_ACK to a DL_RESET_RES described directly above.

dl_primitive Set to DL_RESET_CON.

Connection Release (Disconnection)

DL_DISCONNECT_REQ

```
 typedef struct
{
     ulong     dl_primitive.;
     ulong     dl_reason.;
     ulong     dl_correlation.;
} dl_disconnect_req_t ;
```

This request is used to disconnect an existing connection. It is valid in the DL_DATAXFER, DL_DISCON_IND_PENDING, DL_OUTCON_PENDING, DL_INCON_PENDING, DL_USER_RESET_PENDING and DL_PROV_RESET_PENDING states. The user will receive a DL_OK_ACK response when the connection is disconnected. The connection is returned to the DL_IDLE state.

It is possible for user's DL_DISCONNECT_REQ to cross in the mail with a DL_DISCONNECT_IND. Therefore, in response to a legitimate DL_DISCONNECT_REQ, the user can receive:

- DL_OK_ACK if the DLPI provider has not received a disconnect indication from the data link layer.
- DL_DISCONNECT_IND if this was queued for delivery to the user when the DLPI provider received the DL_DISCONNECT_REQ.
- DL_ERROR_ACK if a DL_DISCONNECT_IND has been sent to the user and the stream is now in the DL_IDLE state.

Once the disconnect is complete, the user may send another DL_CONNECT_REQ or wait for a DL_CONNECT_IND as specified in the DL_BIND_REQ. Alternatively, the user may unbind the DLSAP (see below).

dl_primitive	Set to DL_DISCONNECT_REQ
dl_reason	Reason the user is terminating the connection—This field is ignored.
dl_correlation	Must be zero

DL_DISCONNECT_IND

```
typedef struct
{
     ulong     dl_primitive ;
     ulong     dl_originator ;
     ulong     dl_reason; ;
     ulong     dl_correlation ;
} dl_disconnect_ind_t ;
```

This message is generated by the DLPI provider and sent to the user. It is used to report that the user's opposite has terminated the connection. No response by the user is required; the connection state is changed to DL_IDLE.

dl_primitive DL_DISCONNECT_IND

dl_originator Set to DL_PROVIDER.

dl_reason Set to DL_DISC_UNSPECIFIED.

dl_correlation Set to zero.

DL_UNBIND_REQ

```
typedef struct
{
     ulong        dl_primitive ;
} dl_unbind_req_t ;
```

An unbind request is used to disassociate the DLSAP from the stream. This request is valid only in the DL_IDLE state. If successful, a DL_OK_ACK is returned to the user and the state is changed to DL_UNBOUND.

dl_primitive DL_UNBIND_REQ

DL_DETACH_REQ

```
typedef struct
{
     ulong     dl_primitive ;
} dl_detach_req_t ;
```

A detach request is used to disassociate the PPA from the stream. This request is valid only in the DL_UNBOUND state. If successful, a DL_OK_ACK is returned to the user and the state is changed to DL_UNATTACHED.

dl_primitive DL_DETACH_REQ

Fatal Errors

The DLPI specification has the notion of a fatal error. When a fatal error occurs, an M_ERROR is sent to the user and all messages are discarded until the user closes the stream. Two such cases involving data transfer were mentioned above. There are additional events that cause a stream to be aborted. These are:

1) An attached LPA has been unlinked.
2) The DLPI's control stream has been closed.
3) An internal error is detected regarding buffer conversion, or pointer linkage.
4) A network management MN_ON_OFF_REQ request turns off the UPA that the user has successfully bound.
5) Data is received from the data link layer in an inappropriate state.

Limiting Number of STREAMS Buffers in Use

One of the goals of the DLPI protocol design appears to limit the number of STREAM's buffers that can be in use for any given STREAM. Data delivery is flow-controlled in both directions. Downstream data is placed on the write queue so upstream modules (or the streamhead) obeying *canput()* flow-control will propagate upstream the flow-control back pressure.

The other potential abuse of STREAM's buffers involves abusing the connection control messages, such as DL_BIND_REQ. For example, if the user were in an infinite loop sending DL_BIND_REQs, but not reading the responses, it might be possible for all the STREAMs buffers to end up on the user's streamhead as DL_ERROR_ACKs. The DLPI specification writers apparently considered this possibility, because acknowledgments (DL_INFO_ACK, DL_BIND_ACK, DL_SUBS_BIND_ACK, DL_OK_ACK, DL_ERROR_ACK, and DL_TOKEN_ACK) are delivered in a M_PCPROTO STREAM's message. An M_PCPROTO message, as stated in the STREAMs manual[1]:

> "The Stream head will allow only one M_PCPROTO message to be placed in the read queue at a time. If an M_PCPROTO message is already in the queue when another arrives, the second message is silently discarded and its message blocks freed."

The queue flushing that occurs during reset and disconnect operations prevents the upstream queue from filling with connect, reset, or disconnect protocol messages. In summary, the user need not be concerned about using an abnormal number of STREAMs buffers when communicating with the DLPI provider.

Acknowledgment Generation

Acknowledgments are sent in response to user protocol messages. Acknowledgments are sent to confirm that a desired action was taken (DL_OK_ACK), to report that a user request was in error (DL_ERROR_ACK), or to return information (DL_INFO_ACK). The DLPI protocol is designed so that at most one acknowledgment is queued for delivery to the user at any given time. Acknowledgments are sent in response to user protocol messages. The user's state transition table definition requires that the user wait for a response (ack/confirm/indication) before proceeding. This handshake protocol prevents the buildup of queued acknowledgments and is reinforced by using M_PCPROTO messages to deliver these acknowledgments.

The DLPI provider must internally queue an acknowledgment for later delivery to the user when a sufficiently large buffer is not available to carry the acknowledgment. It is possible for the user to violate the DLPI protocol in such a way that the DLPI provider would have to queue two or more acknowledgments. Since acknowledgments are delivered in M_PCPROTO messages, this situation is analogous to having a M_PCPROTO message arrive at the queue head while a M_PCPROTO message is already queued. That is, the DLPI provider will silently discard an acknowledgment if another is already queued for delivery.

Streams modules should take care when processing an acknowledgment because the DLPI provider processes streams messages on top of its *put* procedure. That is, the user streams module is usually recursively invoked to deliver an acknowledgment. The delivery is deferred if no buffer is immediately available for the ack or if a data link layer interaction is required.

DLPI Features Not Supported

The following DLPI protocol features are not supported.

Connection Manager

That is, in a DL_BIND_REQ request, *dl_conn_mgmt* must be set to zero.

DL_STYLE1

Only DL_STYLE2 is supported—This is set by network management.

Subsequent Bind Request

A DL_ERROR_ACK is returned to a DL_SUBS_BIND_REQ and a DL_SUBS_UNBIND_REQ. A DL_SUBS_BIND_ACK is not generated by the DLPI provider.

Quality of Service (QOS) Fields

Quality of service (QOS) fields, when provided by the user, are ignored. However, if the user specifies a QOS field, it must lie within the user's buffer. QOS parameters are not generated by the DLPI provider. A user generated DL_UDQOS_REQ is rejected with a DL_ERROR_ACK.

SECTION 3

Protocol Extensions

This section describes extensions to the DLPI specification supported by the DLPI provider. The following extensions are optionally enabled by the network manager when a UPA is configured.

XIDs/TESTs

The exchange of XIDs and TESTs is supported while a connection is in the DL_IDLE state. Any queued XIDs/TESTs are flushed when the connection leaves the DL_IDLE state.

ACKs/NAKs

A protocol module can be configured to generate an ACK or NAK when a data message is acknowledged or rejected by its peer. If so configured, the DLPI provider will forward ACK/NAKs to the user.

DisableResets

When loss of data occurs on the link, the protocol module flushes any queued data and sends a reset indication to the DLPI provider. The provider in turn flushes any queued data, and sends a M_FLUSH and DL_RESET_IND upstream. This mechanism can be disabled by the network manager when it configures a UPA.

ioctls

The DLPI provider supports several *ioctl* calls on a data stream. The *ioctl* calls can be used to change the configuration of a UPA, and to obtain statistics. A description of the supported *ioctl*s can be found in the GCOM *UNIX Streams Administrator's Guide*.

APPENDIX A

References

1) A STREAMS-based Data Link Provider Interface - Version 1.2. R.J. Lewis, J.K. Fellin and D.J. Olander. AT&T Bell Laboratories

2) UNIX[TM] SYSTEM V RELEASE 4 Programmer's Guide: STREAMS. UNIX Software Operation. 1990

3) UNIX Streams Administrator's Guide. GCOM. 1994

4) Extended Datalink Layer Provider Interface Specification. NCR Corporation, Inc. April 17, 1990

5) Proposed extension of the DLPI for use between MLP and LAPB. GCOM. September 13, 1990

6) CDI specification

INDEX

e

f

g

h

i

l

m

n

o

p

q

r

s

t

u

w

x