DLPI

Application Program Interface Guide

April 2003

Protocols: LAPB, MLP, QLLC, X.25, HDLC/SDLC, LAPD, and Frame Relay

© 1995-2003 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 exists in this document.

Rsystem and SyncSockets 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. 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 and produced by Senior Technical Writer Debra J. Schweiger using Microsoft Word 97SR-2 and FrameMaker 6.0 on a Windows Millenium platform with the help of subject matter specialists Dave Healy and 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 Dr. Savoy, IL 61874 (217) 351-4241

FAX: (217) 351-4240 e-mail: support@gcom.com homepage: http://www.gcom.com



Table of Contents

5	Table of Contents
11	Figures
15	Tables
<i>17</i>	About This Guide
17	Purpose of This Guide
17	Knowledge Requirements
18	Organization of This Guide
18	Conventions Used In This Guide
18	Special Notices
19	Text Conventions
20	Naming Conventions for DLPI Routines
21	Product Overview
23	Product Description
23	Using the DLPI API Library
24	Gcom Remote API
24	Architecture
24	Client Server Model
25	Running the RAPI Server
25	Using the RAPI Library
26	Architecture of the Data Link Layer
28	Model of the Service Interface
31	Modes of Communication
31	Connection-Oriented Service
32	Connectionless Service
33	DLPI Addressing
33	Physical Attachment Identification
34	Data Link User Identification
35	The Service Reference Model
37	Reference Model Terminology
39	Reference Model Primitives
41	DLPI Specification Services Overview
41	Standard DLPI Services
43	GCOM Extensions to the DLPI Services

CONTENTS GCOM, INC.

Local Management Services
Information Reporting Local Service
Open/Close Local Services
Open Service
Close Service
Attach/Detach Local Services
Attach
Detach
Bind/Unbind Services
Bind
Unbind
Connectionless Services
Understanding Connectionless Data Transfer
Beginning a Session—Open, Attach and Bind
Exchanging Data in Connectionless Mode
Transferring TEST and XID U-Frames
XID Service
TEST Service
Closing a Connectionless Session
LAPB/LAPD and LLC-II Connection-Oriented
Services
Understanding Connection-Oriented Data Transfer
Normal Connection Establishment for LAPB/LAPD and
LLC-II
Normal Data Transfer for LAPB/LAPD and LLC-II
U-Frame Services for LAPB/LAPD and LLC-II
XID Service
TEST Service
Connection-Oriented Unnumbered Information U-Frame Data Transfer Service
Connection Establishment Rejections for LAPB/LAPD and LLC-II
Called DLS User Connection Rejection
DLS Provider Connection Rejection
Connection Retraction Services for LAPB/LAPD and LLC-II
User Retracts Connection Request
DL_DISCONNECT_IND Arrives After DL_CONNECT_RES is Sent
Connection Release Services for LAPB/LAPD and LLC-II
DLS User-Invoked Connection Release

76	Simultaneous DLS User-Invoked Connection Release
77	DLS Provider-Invoked Connection Release
78	Simultaneous DLS User- and DLS Provider-Invoked Connection Release
79	Connection Reset Services for LAPB/LAPD and LLC-II
80	DLS User-Invoked Connection Reset
81	Simultaneous DLS User-Invoked Connection Reset
82	DLS Provider-Invoked Connection Reset
83	HDLC/SDLC Connection-Oriented Services
85	Understanding Connection-Oriented Data Transfer
86	Normal Connection Establishment for HDLC/SDLC
88	Normal Data Transfer for HDLC/SDLC
89	U-Frame Service for HDLC/SDLC
89	XID Service
90	TEST Service
91	Connection-Oriented Unnumbered Information U-Frame Service for HDLC/SDLC Primary
92	Connection-Oriented Unnumbered Information U-Frame Service for HDLC/SDLC Secondary
93	Connection Establishment Rejections for HDLC/SDLC
93	Called DLS User Connection Rejection
94	DLS Provider Connection Rejection
95	Connection Retraction for HDLC/SDLC
95	User Retracts Connection Request
96	DL_DISCONNECT_IND Arrives After DL_CONNECT_RES is Sent
97	Connection Release Services for HDLC/SDLC
97	DLS User-Invoked Connection Release for HDLC/SDLC Primary
98	DLS User-Invoked Connection Release for HDLC/SDLC Secondary
99	Simultaneous DLS User-Invoked Connection Release
100	DLS Provider-Invoked Connection Release for HDLC/SDLC Primary
101	DLS Provider-Invoked Connection Release for HDLC/SDLC Secondary
102	Simultaneous DLS User- and DLS Provider-Invoked Connection Release for HDLC/SDLC Primary
103	Simultaneous DLS User- and DLS Provider-Invoked Connection Release for HDLC/SDLC Secondary
104	Connection Reset Service for HDLC/SDLC
105	DLS User-Invoked Connection Reset for HDLC/SDLC Primary
106	DLS User-Invoked Connection Reset for HDLC/SDLC Secondary

CONTENTS GCOM, INC.

```
107
              Simultaneous DLS User-Invoked Connection Reset
108
       Understanding the Sample Application
114
       Understanding the DLPI API Header File
116
           Including the dlpiapi.h Header File in Your Application
116
           Linking DLPI Header Files
117
           DLPI API Constants
118
           DLPI API Library Routine Error Return Defines
119
           DLPI API Logging Options
120
           Signal Handling Prototype
121
           Global Variables Accessible By Your Application
121
              dlpi_bind_ack
121
              dlpi_conn_con
121
              dlpi_conn_ind
121
              dlpi_ctl_buf
122
              dlpi_ctl_cnt
122
              dlpi_data_buf
122
              dlpi data cnt
122
              dlpi_service_mode
123
           DLPI API Fork Options
125
       DLPI API Library Routines Reference
127
           dlpi_attach_ppa()
128
           dlpi_bind_dlsap()
129
           dlpi_clear_zombies()
130
           dlpi_close()
131
           dlpi_complete_req()
133
           dlpi_configure_dlsaps()
134
           dlpi_connect()
136
           dlpi_connect_req()
137
           dlpi_connect_wait()
138
           dlpi_decode_ctl()
139
           dlpi_decode_disconnect_reason()
140
           dlpi_detach_ppa()
141
           dlpi_discon_req()
142
           dlpi_disconnect_req()
143
           dlpi_get_a_msg()
144
           dlpi_get_both()
145
           dlpi_get_info_strm()
146
           dlpi_get_style_strm()
147
           dlpi_init()
```

```
148
          dlpi_init_FILE()
149
          dlpi_listen()
152
          dlpi_open()
153
          dlpi_open_log()
154
          dlpi_perror()
155
          dlpi_print_msg()
156
          dlpi_printf()
157
          dlpi_put_both()
158
          dlpi_put_proto()
159
          dlpi_rcv()
159
              Receiving U-Frames
160
              Non-Blocking I/O
169
          dlpi_rcv_msg()
171
          dlpi_read_data()
172
          dlpi_reset_req()
173
          dlpi_reset_res()
174
          dlpi_send_attach_req()
175
          dlpi_send_bind_req()
176
          dlpi_send_connect_req()
177
          dlpi_send_connect_res()
178
          dlpi_send_detach_req()
179
          dlpi_send_disconnect_req()
180
          dlpi_send_info_req()
181
          dlpi_send_reset_req()
182
          dlpi_send_reset_res()
183
          dlpi_send_stats_req()
184
          dlpi_send_test_req()
185
          dlpi_send_test_res()
186
          dlpi_send_uic()
187
          dlpi_send_unbind_req()
188
          dlpi_send_xid_req()
189
          dlpi_send_xid_res()
190
          dlpi_set_log_size()
191
          dlpi_set_signal_handling()
193
          dlpi_set_unnum_frame_handler()
194
          dlpi_test_req()
195
          dlpi_test_res()
196
          dlpi_uic_req()
```

CONTENTS GCOM, INC.

Figures

22	Figure 1	DLPI provider
27	Figure 2	DLPI architecture
29	Figure 3	DLPI architecture (repeated)
33	Figure 4	Data Link addressing components
36	Figure 5	Reference model
38	Figure 6	Reference model (repeated)
46	Figure 7	Information reporting for all frame level protocols
48	Figure 8	Open services for all frame level protocols
50	Figure 9	Attaching a stream to a physical line for all frame level protocols
51	Figure 10	Detaching a stream from a physical line for all frame level protocols
52	Figure 11	Binding a stream to a DLSAP all frame level protocols
53	Figure 12	Unbinding a stream from a DLSAP for all frame level protocols
58	Figure 13	Connectionless data transfer for frame relay, LAPB/LAPD, LLC-II and HDLC/SDLC
59	Figure 14	XID service for connectionless mode LAPB/LAPD, LLC-II and HDLC/SDLC
60	Figure 15	TEST service for connectionless mode LAPB/LAPD, LLC-II and HDLC/SDLC
66	Figure 16	Successful connection establishment for LAPB/LAPD and LLC-II
67	Figure 17	Normal data transfer for LAPB/LAPD and LLC-II
68	Figure 18	XID service for connection-oriented LAPB/LAPD and LLC-II
69	Figure 19	TEST service for connection-oriented LAPB/LAPD and LLC-II
70	Figure 20	Unnumbered Information U-Frame Data Transfer Service for LAPB/LAPD and LLC-II
71	Figure 21	Called DLS user connection rejection for LAPB/LAPD and LLC-II
72	Figure 22	DLS provider connection rejection for LAPB/LAPD and LLC-II
73	Figure 23	User retracts connection request for LAPB/LAPD and LLC-II
74	Figure 24	DL_DISCONNECT_IND arrives after

CONTENTS GCOM, INC.

		and LLC-II
75	Figure 25	DLS user-invoked connection release for LAPB/LAPD and LLC-II
76	Figure 26	Simultaneous DLS user-invoked connection release for LAPB/LAPD and LLC-II
77	Figure 27	DLS provider-invoked connection release for LAPB/LAPD and LLC-II
78	Figure 28	Simultaneous DLS user- (+) and DLS provider- (++) invoked connection release for LAPB/LAPD and LLC-II
80	Figure 29	DLS user-invoked connection reset for LAPB/LAPD and LLC-II
81	Figure 30	Simultaneous DLS user-invoked connection reset for LAPB/LAPD and LLC-II
82	Figure 31	DLS provider-invoked connection reset for LAPB/LAPD and LLC-II
86	Figure 32	Successful connection establishment for HDLC/SDLC
88	Figure 33	Normal data transfer for HDLC/SDLC
89	Figure 34	XID service for connection-oriented HDLC/SDLC
90	Figure 35	TEST service for connection-oriented HDLC/SDLC
91	Figure 36	Unnumbered information u-frame service for connection-oriented HDLC/SDLC primary
92	Figure 37	Unnumbered information u-frame service for connection-oriented HDLC/SDLC secondary
93	Figure 38	Called DLS user connection rejection for HDLC/SDLC
94	Figure 39	DLS provider connection rejection for HDLC/SDLC
95	Figure 40	User retracts connection request for HDLC/SDLC
96	Figure 41	DL_DISCONNECT_IND Arrives After DL_CONNECT_RES is Sent for HDLC/SDLC
97	Figure 42	DLS user-invoked connection release for HDLC/SDLC primary
98	Figure 43	DLS user-invoked connection release for HDLC/SDLC secondary
99	Figure 44	Simultaneous DLS user-invoked connection release for HDLC/SDLC

100	Figure 45	DLS provider-invoked connection release for HDLC/SDLC primary
101	Figure 46	DLS provider-invoked connection release for HDLC/SDLC secondary
102	Figure 47	Simultaneous DLS User- (+) and DLS provider- (++) invoked connection release for HDLC/SDLC primary
103	Figure 48	Simultaneous DLS user- (+) and DLS provider- (++) invoked connection release for HDLC/SDLC secondary
105	Figure 49	DLS User-Invoked Connection Reset for HDLC/SDLC Primary
106	Figure 50	DLS User-Invoked Connection Reset for HDLC/SDLC Secondary
107	Figure 51	Simultaneous DLS User-Invoked Connection Reset for HDLC/SDLC
109	Figure 51	Sample DLPI API Program
111	Figure 51	Sample DLPI API Program (repeated)

CONTENTS GCOM, INC.

Tables

18	Table 1	Location of Important Information
20	Table 2	Naming Conventions for DLPI Routines
40	Table 1	Standard DLS services and DLPI API library routines
42	Table 2	GCOM extensions to DLPI as implemented in the DLPI API library

CONTENTS GCOM, INC.



About This Guide

Purpose of This Guide

This guide is written for C programmers who intend to transfer data between local and remote peers in a UNIX STREAMS environment using a data link layer protocol. The data and other messages are passed to the streams-based Data Link Provider Interface (DLPI) by making calls to routines contained in the GCOM's DLPI Application Program Interface (API) Library.

Knowledge Requirements

You should be familiar with the OSI Reference Model terminology, OSI Data Link Services and the implementation of UNIX STREAMS that your application is using. You must also be proficient with the C programming language and have specific knowledge of the data link layer protocol that your application is using to transfer data.

GCOM implements a subset of the DLPI standard that is fully described in Univel's *STREAMS Modules and Drivers Manual*, universal part number 10000203. Contact Univel (a division of Novell) at:

Univel 2180 Fortune Drive San Jose, CA 95131 (U.S.A.) (408) 473-8788

The C programming language is described in Kernighan and Ritchie's *The C Programming Language*.

For an understanding of the data link layer protocols, GCOM recommends reading Uyless Black's book, entitled *Data Link Protocols*, © 1993 by PTR Prentice-Hall, Inc. (ISBN 0-13-204918-X).

Preface GCOM, Inc.

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:
Product description and related files	Section 1
Architecture of the Data Link Layer	Section 2
Understanding the service reference model	Section 3
Local Management Services	Section 4
Connectionless Services	Section 5
LAPB/LAPD and LLC-II Connection-Oriented Services	Section 6
HDLC/SDLC Connection-Oriented Services	Section 7
Understanding a sample DLPI API test program and linking header files	Section 8
Understanding the DLPI API header file, dlpiapi.h	Section 9
DLPI API Library routine reference	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.

GCOM, INC. CONVENTIONS: USED IN THIS GUIDE

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 instructs you to enter "cd gcom"

at the system prompt.

PREFACE GCOM, INC.

Naming Conventions for DLPI Routines

The general purpose of most routines can be inferred from the prefix and suffix naming conventions in Table 2.

 Table 2
 Naming Conventions for DLPI Routines

Prefix or Suffix?	String	Description
Prefix	dlpi_	DLPI API Library C routine
Suffix	_req	Request routines
Suffix	_res	Response routines

Product Overview

23	Product D	escription

- Using the DLPI API Library
- 24 GCOM Remote API

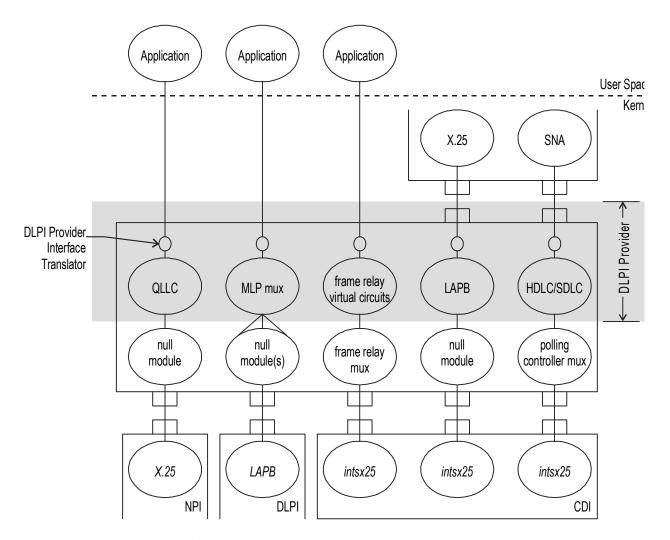


Figure 1 DLPI provider

Product Description

The Data Link Provider Interface (DLPI) enables a Data Link Service (DLS) user to access and use any of a variety of conforming DLS providers without special knowledge of the provider's protocol. The GCOM DLPI Application Program Interface (API) translates DLPI protocol messages to your application by way of subroutine calls to the GCOM DLPI API Library.

The GCOM DLPI API Library is intended to support the data link layer protocols shown in Figure 1:

- LAPB
- HDLC/SDLC
- ISDN LAPD
- QLLC
- the MLP multiplexor
- frame relay
- LLC-II (not pictured)

These protocols connect to your application by using the UNIX STREAMS implementation of DLPI API.

Using the DLPI API Library

In order to utilize the GCOM DLPI API library it is necessary to link it into your program ahead of the "libgcom" library in order to link to the routines that perform the remote functions. A sample command line link for this is as follows:

```
cc -o foo foo.o /usr/lib/gcom/dlpiapi.a /usr/lib/gcom/libgcom.a
```

Two important files that pertain to the API library are described as follows:

/usr/lib/gcom/dlpiapi.a. This file is the library of DLPI/SDLC interface routines that is linked in with your application program.

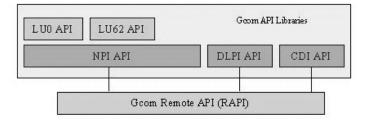
/usr/include/gcom/dlpiapi.h. This header file should be included in your application program. It contains defines and prototype declarations useful to you application program. This header file is compatible with both old-style C language and ANSI C. To use the ANSI style function prototype declarations, you should define the compile-time symbol PROTOTYPE prior to including the gcom/dlpiapi.h file.

GCOM Remote API

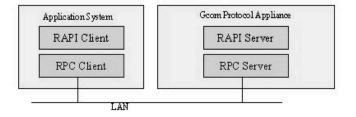
GCOM'S Remote API (RAPI) is a library of functions that allows the standard GCOM APIs to operate on protocol stacks that are configured and running on a remote machine. It is especially useful in situations in which the application code resides on a server system and the protocol processing is performed on a GCOM Protocol appliance attached to the server via a LAN connection.

Architecture

The illustration, below, shows how the GCOM RAPI relates to all GCOM APIs. In the suite of GCOM API libraries, the NPI API interfaces to GCOM's NPI driver for X.25, SNA and Bisync protocols. The GCOM DLPI interfaces to GCOM'S DLPI driver for link layer protocols such as LAPB, LAPD, HDLVC and Frame Relay. The GCOM CDI API library interfaces to GCOM's synchronous protocol drivers directly for raw frame access.



Client Server Model



The Remote API is intended for use in a client/server environment. The user's application program, linked with the GCOM RAPI library, runs on the client system. The server system is typically a GCOM Protocol Appliance. It contains the communication hardware, protocol software and the Remote API server.

Running the RAPI Server

The GCOM Remote API server is named Gcom_rapisvr. It is usually unnecessary to run this program with any arguments. By default the program runs in the background. It can be run from the command line or from a shell script.

It is common to run <code>Gcom_rapisvr</code> under root from an "rc" script. However, if permissions are set appropriately on the files that are to be accessed remotely, it is perfectly possible to run <code>Gcom_rapisvr</code> from a non-root user id.

Note: Additional information on GCOM RAPI arguments, authentication, and other API routines can be found by accessing the GCOM RAPI white paper on the www.gcom.com web site.

Using the RAPI Library

In order to utilize the GCOM RAPI library it is necessary to link it into your program ahead of the "libgcom" library in order to link to the routines that perform the remote functions. A sample command line link for this is as follows:

cc -o foo foo.o /usr/lib/gcom/dlpiapi.a /usr/lib/gcom/rapi.a
/usr/lib/gcom/libgcom.a

Note: If RAPI library is omitted, then all file operations will be executed on the same machine on which the application program is running.

In the application program, be sure to use the correct API routine to open data streams on a remote system. The open routing of each of these routines is passed a parameter which is a pointer to a string which names the remote host. Passing a NULL pointer, or a pointer to an empty string, indicates that the file is to be opened on the local machine.

When opening or closing DLPI protocol data streams, use the functions:

Open routine: *dlpi_open* Close routine: *dlpi_close*

Apart from using the specially provided open and close functions, there are no other programming interface considerations for making an application utilize remote protocol services.



Architecture of the Data Link Layer

28	Model of the Service Interface
31	Modes of Communication
31	Connection-Oriented Service
31	Local Management Phase
31	Connection Establishment Phase
31	Data Transfer Phase
31	Connection Release Phase
32	Connectionless Service
33	DLPI Addressing
33	Physical Attachment Identification
34	Data Link User Identification

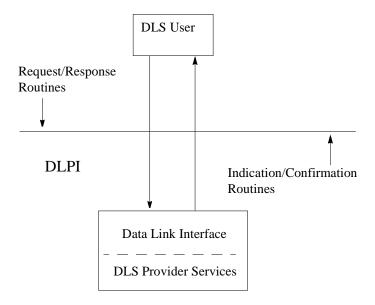


Figure 2 DLPI architecture

GCOM, INC. MODEL OF THE SERVICE INTERFACE

The *data link layer* (layer two in the OSI Reference Model) is responsible for the transmission and error-free delivery of bits of information over a physical communication medium. The data link layer can be used in both a connection-oriented and connectionless environment.

The model of the data link layer is presented here to describe concepts that are used throughout this document. The model is described in terms of an interface architecture, addressing concepts required to identify different components of that architecture. The description of the data link layer model assumes familiarity with the OSI Reference Model.

Model of the Service Interface

Each layer of the OSI Reference Model has two standards:

- •One that defines the services provided by the layer
- •One that defines the protocol through which layer services are provided

Figure 2 shows an abstract view of the DLPI architecture.

The *data link interface* is the boundary between the network and data link layers of the OSI Reference Model. The network layer entity is the user of the services of the data link interface, called the *Data Link Service (DLS) user*. The data link layer entity is the provider of those services, called the *DLS provider*. This interface consists of a set of routines that provide access to the data link layer services plus the rules for accessing the interface. A data link interface service routine might request a particular service or indicate a pending event.

To provide uniformity among the various UNIX system networking products, an effort is underway to develop service interfaces that map to the OSI Reference Model. A set of kernel-level interfaces, based on the streams development environment, constitute a major portion of this effort. The DLPI API Library service routines that make up these interfaces use streams message primitives that are transferred between the user and provider of the service. DLPI is one such kernel-level interface, so it is targeted for streams protocol modules that either use or provide data link services. In addition, your application program can access a streams-based data link provider directly by using the *putmsg*(2) and *getmsg*(2) UNIX system calls.

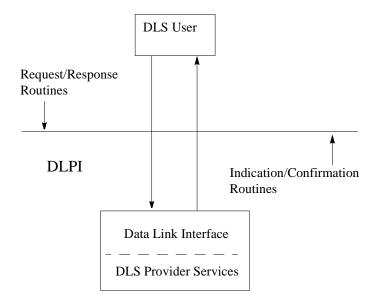


Figure 3 DLPI architecture (repeated)

GCOM, INC. MODEL OF THE SERVICE INTERFACE

Referring to Figure 3, the DLS provider is configured as a streams driver and the DLS user accesses the provider using open(2) to establish a stream to the DLS provider. The stream acts as a communication mechanism between a DLS user and the DLS provider. After the stream is created, the DLS user and DLS provider communicate by using the routines presented in Appendix A on page 125.

DLPI is intended to free data link users from specific knowledge of the services provided by the data link provider. Specifically, the definition of DLPI hopes to allow a DLS user to be implemented independent of a specific communication medium. Any data link provider (supporting any communications medium) that conforms to the DLPI specification can be substituted beneath the DLS user to provide the data link services. Support of a new DLS provider should not require any changes to the implementation of the DLS user.

GCOM, INC. MODES OF COMMUNICATION

Modes of Communication

The GCOM implementation of the data link provider interface supports two modes of communication: connection-oriented and connectionless.

Connection-Oriented Service

The connection-oriented service is characterized by four phases of communication: local management, connection establishment, data transfer and connection release.

Local Management Phase This phase enables a DLS user to initialize a stream for use in communication and establish an identity with the DLS provider.

Connection Establishment Phase

This phase enables two peer DLS users to establish a data link connection between them to exchange data. Typically, one user (the calling DLS user) initiates the connection establishment procedures, while another user (the called DLS user) waits for the incoming connect request. The called DLS user is identified by an address associated with its stream, as will be discussed in "DLPI Addressing" on page 33.



Note: GCOM's implementation of DLPI allows both peers to initiate the connection.

A called DLS user can either accept or deny a request for a data link connection. If the request is accepted, a connection is established between the DLS users and they enter the data transfer phase.

For both the calling and called DLS users, only one connection can be established per stream. Thus, the stream is the communication endpoint for a data link connection.

Data Transfer Phase

In this phase, the DLS users are considered peers and may exchange data simultaneously in both directions (full-duplex) over an established data link connection. Either DLS user can send data to its peer DLS user at any time. Data sent by a DLS user is guaranteed to be delivered to the to the remote user in the order in which it was sent.

Connection Release Phase This phase enables either the DLS user or the DLS provider to break an established connection. The release procedure is considered abortive, so any data that has not reached the destination user when the connection is released may be discarded by the DLS provider.

GCOM, INC. MODES OF COMMUNICATION

Connectionless Service

The connectionless mode service does not use the connection establishment and release phases of the connection-oriented service. The local management phase is still required to initialize a stream. Once initialized, however, the connectionless data transfer phase is immediately entered.

DLPI Addressing

Each DLPI user must establish an identity to communicate with other data link users. Setting up this path involves two steps. First, the DLS user must identify the physical medium over which it will communicate. This is particularly evident on systems that are attached to multiple physical media. Second, the DLS user must register itself with the DLS provider so that the provider can deliver protocol data units destined for that user. Figure 4 illustrates the components of this identification approach, which are explained in the text that follows it.

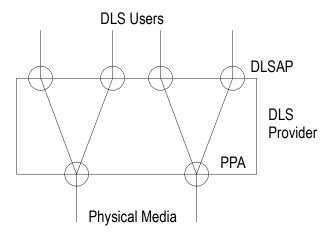


Figure 4 Data Link addressing components

Physical Attachment Identification

The *Physical Point of Attachment* (PPA in Figure 4) is the point at which a system attaches itself to a physical communications medium. All communication on that physical medium funnels through the PPA. On systems where a DLS provider supports more than one physical medium, the DLS user must identify which medium it will communicate through. A PPA is identified by a unique PPA identifier. For media that support physical layer multiplexing of multiple channels over a single physical medium (such as the B and D channels of ISDN), the PPA identifier must identify the specific channel over which communication occurs.

Two styles of DLS provider are defined by DLPI, distinguished by the way they enable a DLS user to choose a particular PPA: style 1 and style 2.

Defined: DL_STYLE1

The DL_STYLE1 provider assigns a PPA based on the major/minor device that the DLS user opened. One possible implementation of a style 1 driver would reserve a major device for each PPA that the data link

driver would support. This would allow the streams clone open feature to be used for each PPA configured. This style of provider is appropriate when few PPAs will be supported.

Defined: DL_STYLE2

If the number of PPAs that a DLS provider will support is large, a style 2 provider implementation is more suitable. The DL_STYLE2 provider requires a DLS user to explicitly identify the desired PPA using the <code>dlpi_attach_ppa()</code> routine. For a style 2 driver, the <code>dlpi_open()</code> creates a stream between the DLS user and DLS provider, and the <code>dlpi_attach_ppa()</code> routine then associates a particular PPA with that stream.

The DLS user utilizes the supported GCOM DLPI API Library routines $dlpi_attach_ppa()$ and $dlpi_bind_dlsap()$ to define a set of enabled physical and service access point (SAP) address components on a perstream basis. It is invalid for a DLS provider to ever send an upstream data message for which the DLS user on that stream has not requested. The burden is on the provider to enforce by any means that it chooses, the isolation of SAP and physical address space effects on a per-stream basis.

Data Link User Identification

A data link user's identity is established by associating it with a *Data Link Service Access Point (DLSAP)*, which is the point through which the user will communicate with the data link provider. A DLSAP is identified by a DLSAP address.

The DLSAP address identifies a particular data link service access point that is associated with a stream (communication endpoint). The *dlpi_bind_dlsap()* routine enables a DLS user to either choose a specific DLSAP by specifying its DLSAP address.

The Service Reference Model

37	Reference Model Terminology
39	Reference Model Primitives
41	DLPI Specification Services Overview
41	Standard DLPI Services
13	CCOM Extensions to the DI DI Services

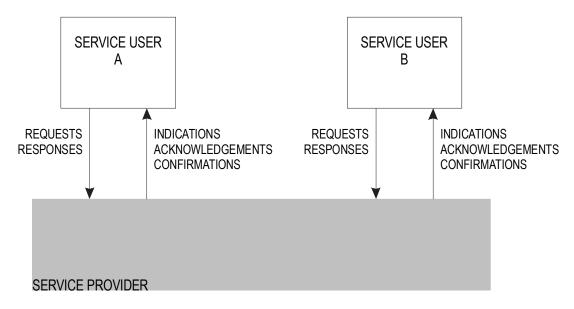


Figure 5 Reference model

GCOM, INC. REFERENCE MODEL TERMINOLOGY

Reference Model Terminology

A useful set of abstractions for discussing communications systems is embodied in a reference model such as the ones established by ISO, the CCITT and other standards bodies. A similar model (referred to hereafter simply as the Reference Model) is used in this manual, but its terms do not conform precisely to those of the standards documents of the above mentioned organizations.

A SERVICE is a set of capabilities provided by a SERVICE PROVIDER.

In the Reference Model, a service is defined as a set of capabilities, such as circuit establishment, error detection and/or data transfer. A service provider is an abstract entity that conceptually includes all the software, hardware, transmission media and so on required to provide a particular service.

A SERVICE USER uses the SERVICES of a SERVICE PROVIDER.

A service user is an entity that uses the capabilities of a service provider. An entity can be a service user with respect to a particular service provider, while at the same time itself being (part of) a service provider for other entities.

Example: The network layer.

Layer three (network layer) in the 7-layer ISO protocol stack is an excellent example. It is a service user with respect to layer two (data link layer), and a service provider to layer four (transport layer).

Figure 5 shows two service users (called A and B) connected to a single service provider. The service provided consists of whatever capabilities are required to achieve data transfer between the two service users.

GCOM, INC. REFERENCE MODEL TERMINOLOGY

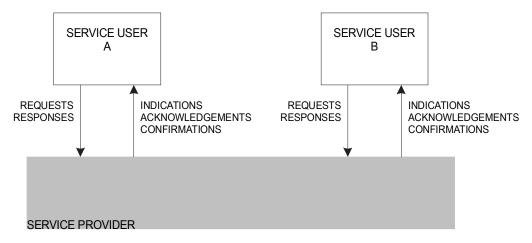


Figure 6 Reference model (repeated)

GCOM, INC. REFERENCE MODEL PRIMITIVES

Reference Model Primitives

The arrows in Figure 2 indicate the sending and receiving entities of the following primitives, which are used to categorize communications between service providers and service users:

request Issued by a service user to a service

provider to initiate an operation.

indication Issued by a service provider to a service

user to initiate an operation or to indicate that an operation has been invoked by a

peer service user.

response Issued by a service user to a service

provider to respond (positively or negatively) to a previous indication.

confirmation Issued by a service provider to a service

user to confirm the successful completion or failure of an operation that involved a response from a peer service user.

acknowledgement Issued by a service provider to a service

user to acknowledge the local successful completion or failure of an operation started as a result of an earlier request.

LAPD examples

The specific types of primitives within each of these categories depends upon the particular service provider. For example, a service provider can support several requests to manage a connection such as a LAPD open request, attach request and bind request, as well as a data request to initiate the transmission of data. A category can also be empty. For example, no confirmations are supported for the LAPD API.

GCOM, INC. REFERENCE MODEL PRIMITIVES

 Table 1
 Standard DLS services and DLPI API library routines

Modes	Service	DLPI API Library Routine(s)
Local Management (for both connectionless and connection-oriented)	Information Reporting	dlpi_rcv(), dlpi_complete_req(), dlpi_send_info_req(), dlpi_get_info()
	Attach/Detach	dlpi_rcv(), dlpi_complete_req(), dlpi_send_attach_req(), dlpi_attach_ppa(), dlpi_send_detach_req(), dlpi_detach_ppa()
	Bind/Unbind	dlpi_rcv(), dlpi_complete_req(), dlpi_send_bind_req(), dlpi_bind_dlsap(), dlpi_send_unbind_req(), dlpi_unbind_dlsap()
Connectionless	Data Transfer	dlpi_send_uic(), dlpi_rcv(), dlpi_read_data()
Connection-Oriented	Connection Establishment	dlpi_rcv(), dlpi_complete_req(), dlpi_send_connect_req(), dlpi_send_connect_res(), dlpi_connect(), dlpi_connect_req(), dlpi_listen()
	Normal Data Transfer	dlpi_read_data(), dlpi_write_data(), dlpi_rcv(),
	U-Frames	<pre>dlpi_rcv(), dlpi_send_xid_req(), dlpi_send_xid_res(), dlpi_read_data(), dlpi_send_test_req(), dlpi_send_uic()</pre>
	Transferring TEST and XID U-frames	<pre>dlpi_rcv(), dlpi_send_xid_req(), dlpi_send_xid_res(), dlpi_read_data(), dlpi_send_test_req()</pre>
	Connection Rejections	dlpi_rcv(), dlpi_complete_req(), dlpi_send_connect_req(), dlpi_send_disconnect_req(), dlpi_connect(), dlpi_connect_req(), dlpi_listen(), dlpi_send_connect_res()
	Connection Release	dlpi_rcv(), dlpi_complete_req(), dlpi_send_disconnect_req(), dlpi_disconnect_req()
	Connection Reset	dlpi_rcv(), dlpi_complete_req(), dlpi_send_reset_req(), dlpi_send_reset_res(), dlpi_reset_req(), dlpi_reset_res(), dlpi_send_conn_req()

DLPI Specification Services Overview

The various features of the Data Link Provider Interface (DLPI) interface are defined in terms of the services provided by the Data Link Service (DLS) provider and the individual primitives that can flow between the DLS user and the DLS provider.

The data link provider interface supports two modes of service:

- Connection-oriented
- Connectionless

Connection-oriented mode requires connection setup/release and guarantees reliable, flow-controlled data delivery with all frames in their original sequence and with no missing or duplicate frames or bit errors within the frames.

Connectionless mode does not require connection setups, but cannot guarantee delivery of data or external flow control.

Standard DLPI Services

Table 2 relates the various phases of connectionless and connectionoriented services to the DLS services and associated GCOM-supplied DLPI API Library routines. The routines are described in detail in Appendix A, page 125.

Both connection-oriented and connectionless service modes require the local management functions shown in Table 1. They both also support U-frames (XIDs, TESTs and unnumbered information U-frames).

 Table 2
 GCOM extensions to DLPI as implemented in the DLPI API library

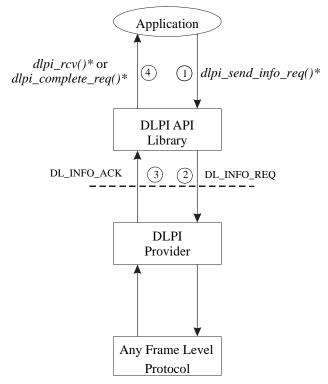
Modes	Service	DLPI API Library Routine(s)
Local Management (applies to both connectionless and connection-oriented)	Initialization	dlpi_init(), dlpi_init_FILE()
	Open/Close	dlpi_open(), dlpi_open/close
Configuration	DLPI Configuration	dlpi_configure_dlsaps()
Connectionless	Data Transfer	dlpi_get_a_msg(), dlpi_put_data(), dlpi_set_unnum_frame_handler()
Connection-oriented	Data Transfer	dlpi_get_a_msg(), dlpi_set_unnum_frame_handler()
Troubleshooting	Troubleshooting	dlpi_printf(), dlpi_decode_ctl(), dlpi_set_signal_handling()
Miscellaneous	Miscellaneous	dlpi_get_style(), dlpi_send_test_res()

GCOM Extensions to the DLPI Services

Beyond the standard DLPI provider services, GCOM has implemented a number of extensions that you might find useful while writing your application, as listed in Table 3. The DLPI API Library routines are described in detail in Appendix A, page 125.

Local Management Services

47	Information Reporting Local Service
48	Open/Close Local Services
48	Open Service
49	Close Service
50	Attach/Detach Local Services
50	Attach
51	Detach
52	Bind/Unbind Services
52	Bind
53	Unbind



* dlpi_get_info() can be substituted for events 1 and 4

Figure 7 Information reporting for all frame level protocols

This section describes the local management services provided by the link-layer DLPI API Library for all connectionless and connection-oriented frame level protocols. These services, which fall outside the scope of standards specifications, define the method for initializing a stream that is connected to a Data Link Service (DLS) provider. DLS provider information reporting services are also supported by the local management facilities. These services are provided by the link-layer DLPI API Library.

Information Reporting Local Service

This service provides information about the DLPI stream to the DLS user. The *dlpi_send_info_req()* routine requests the DLS provider to return operating information about the stream. The DLS provider returns the requested information to one of several DLPI API Library routines in a DL_INFO_ACK message.

GCOM, INC. OPEN/CLOSE LOCAL SERVICES

Open/Close Local Services

The DLPI specification does not define an open or close local management service. However, both are necessary to start and finish a data transfer session in both connectionless and connection-oriented environments.



Note: You must call dlpi_init() before opening a session.

Open Service

It is necessary to open a session before attaching the PPA, binding the DLSAP and (in connection-oriented mode) connecting to a circuit. Therefore, GCOM supplies the <code>dlpi_open()</code> routine, shown in Figure 8, which simply asks DLPI to open a session. After the <code>open()</code> system call is accepted by the DLPI Provider, a file descriptor is returned.

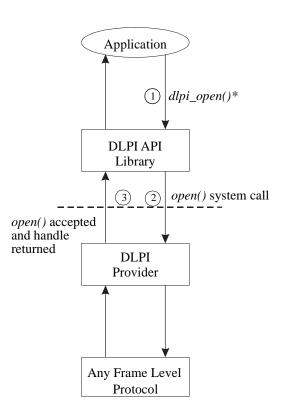


Figure 8 Open services for all frame level protocols

GCOM, INC. OPEN/CLOSE LOCAL SERVICES

Close Service

The last thing your application must do before a session is complete is close the session. This involves unbinding, detaching and closing. GCOM recommends simply using the *dlpi_close* routine. Your application would seldom need to explicitly detach and unbind a session. However, the DLPI API Library supplies such routines if you need to use them.

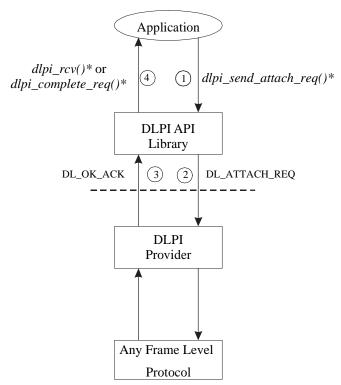
GCOM, INC. ATTACH/DETACH LOCAL SERVICES

Attach/Detach Local Services

Attach

The attach service assigns a Physical Point of Attachment (PPA) to a stream. This service is required for style 2 DLS providers (explained in "DLPI Addressing" on page 33) to specify the physical medium over which communication occurs. The DLS provider indicates success with a DL_OK_ACK and failure with a DL_ERROR_ACK. An attach request is only valid when the stream is detached (DL_UNATTACHED).

The normal sequence of events is illustrated in Figure 9.



^{*} dlpi_attach_ppa() can be substituted for events 1 and 4

Figure 9 Attaching a stream to a physical line for all frame level protocols

GCOM, INC. ATTACH/DETACH LOCAL SERVICES

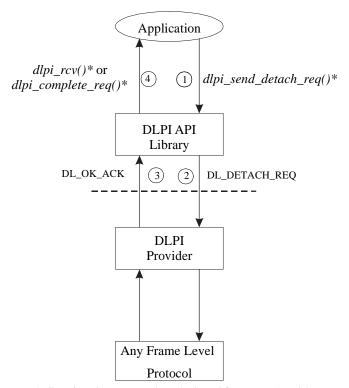
Detach

A Physical Point of Attachment (PPA) can be disassociated with a stream using the *dlpi_send_detach_req()* routine. A detach request is only valid when the stream is attached to the PPA but not bound (DL_UNBOUND).



Note: *GCOM suggests unbinding and detaching using the dlpi_close() command.*

The normal sequence of events is illustrated in Figure 10.



^{*} dlpi_detach_ppa() can be substituted for events 1 and 4

Figure 10 Detaching a stream from a physical line for all frame level protocols

GCOM, INC. BIND/UNBIND SERVICES

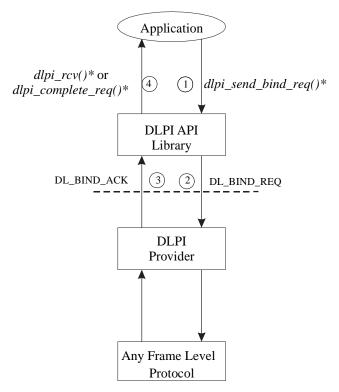
Bind/Unbind Services

The bind service associates a Data Link Service Access Point (DLSAP) with a stream.

Bind

The *dlpi_bind_dlsap()* or *dlpi_send_bind_req()* routine requests that the DLS provider bind a DLSAP to a stream. It also notifies the DLS provider to make the stream active with respect to the DLSAP for processing connectionless data transfer and connection establishment requests. The DLS provider indicates success with a DL_BIND_ACK and failure with a DL_ERROR_ACK. A bind request is valid only when the stream is attached to a PPA but not bound to a DLSAP (DL_UNBOUND).

The normal sequence of events is illustrated in Figure 11.



^{*} dlpi_bind_dlsap() can be substituted for events 1 and 4

Figure 11 Binding a stream to a DLSAP all frame level protocols

GCOM, INC. BIND/UNBIND SERVICES

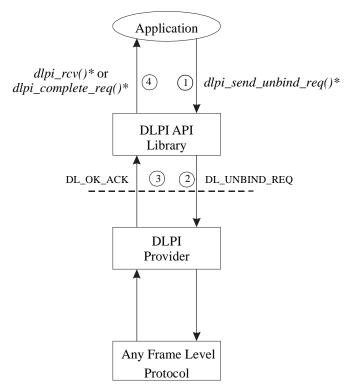
Unbind

The *dlpi_unbind_dlsap()* or *dlpi_send_unbind_req()* routine requests the DLS provider to unbind all DLSAPs from a stream. The DLS provider indicates success with DL_OK_ACK or failure with DL_ERROR_ACK. An unbind request is valid only when the stream is bound to a DLSAP but not in a connected state (DL_IDLE).



Note: GCOM suggests unbinding and detaching using the dlpi_close() command.

The normal sequence of events is illustrated in Figure 12.



^{*} dlpi_unbind_dlsap() can be substituted for events 1 and 4

Figure 12 Unbinding a stream from a DLSAP for all frame level protocols

GCOM, INC.

BIND/UNBIND SERVICES

Connectionless Services

57	Understanding Connectionless Data Transfer
58	Beginning a Session—Open, Attach and Bind
58	Exchanging Data in Connectionless Mode
59	Transferring TEST and XID U-Frames
59	XID Service
60	TEST Service
61	Closing a Connectionless Session

5 · CONNECTIONLESS SERVICES GCOM, INC.

Understanding Connectionless Data Transfer

Once a stream has been initialized using the local management services, it can be used to send and receive connectionless data units.

What?

Connectionless data transfer implies that there is no connection. Data are simply shipped through the circuit, without the guaranteed expectation that it will arrive correctly on the other end. There is nothing that guarantees the safe delivery of data by all parts of the transport mechanism (such as the wire or the network) or that the data will be delivered in its original sequence without duplicated frames. It is up to the client to detect and correct lost data. However, checksums will detect most bit-errors over synchronous lines.

Furthermore, connectionless data transfer has no DLS Provider-to-DLS Provider flow-control. That is, there is internal flow-control that prevents the system from being flooded with data, but there is no peer-to-peer flow-control.

Why? GCOM supports connectionless data transfer to:

- Communicate directly with a driver. That is, obtain direct access to a synchronous line (or port)
- Communicate over a frame relay circuit
- Use the U-frame mechanism of LAPB/LAPD, LLC-II and HDLC/SDLC (assuming you never ask for a connection)



Note: Conversely, connection-oriented data transfer is associated with LAPB/LAPD, LLC-II and HDLC/SDLC running in connection-oriented mode.

How? For successful connectionless data transfer to occur over DLPI, your application must achieve the following three phases:

- Begin the session by issuing an open, attach and bind
- Exchange data across the line
- End the session by issuing an unbind, detach and close

Although frame relay guarantees frame delivery without duplication, complex WAN routing might cause a frame to be duplicated anyway. Therefore, you should consider coding your application to check for such duplication.

5 · CONNECTIONLESS SERVICES GCOM, INC.

Beginning a Session—Open, Attach and Bind

In connectionless data transfer mode, your application first calls *dlpi_init()* and then proceeds to open a session, attach a PPA and bind the DLSAP.

Exchanging Data in Connectionless Mode

As mentioned previously, the connectionless data transfer service provides for the exchange of user data (DLSDUs) in either direction or in both directions simultaneously (full-duplex) without ever establishing a data link connection. Data transfer is not acknowledged, and there is no end-to-end flow control provided. As such, the connectionless data transfer service cannot guarantee reliable delivery of data.

Figure 13, below, shows how data are transferred in a connectionless mode environment. Data are transferred in the form of bits for frame relay and unnumbered information U-frames for all other supported protocols. TESTs and XIDs can also be transferred in a connectionless mode as explained starting on page 59.

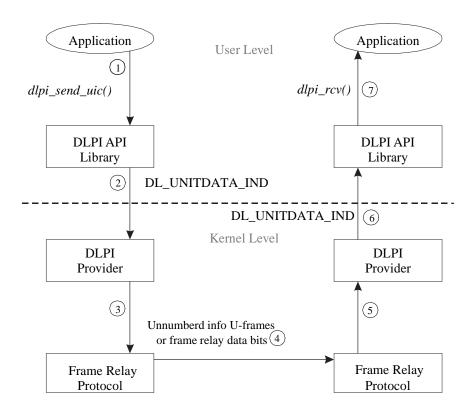


Figure 13 Connectionless data transfer for frame relay, LAPB/LAPD, LLC-II and HDLC/SDLC

Transferring TEST and XID U-Frames

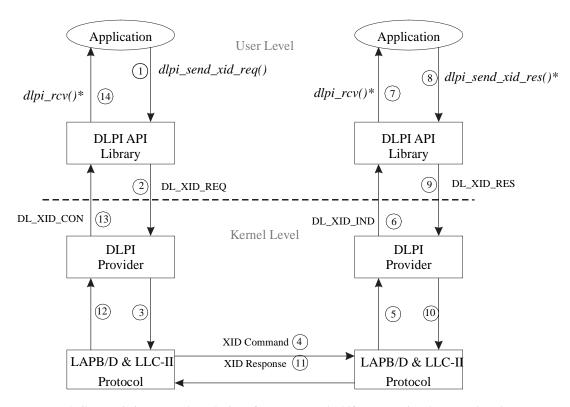
TESTs and XIDs are U-frame types defined by the HDLC/SDLC, LAPB/LAPD and LLC-II protocols. As such, these U-frames can be transferred once a stream has been attached, bound or while a connection exists.



Note: Frame relay does not support TESTs and XIDs.

XID Service

The local DLS user requests that an XID command response be sent to the remote DLS user. The feature that allows the DLS user to request automatic handling of the XID response at bind time is not supported at this time.



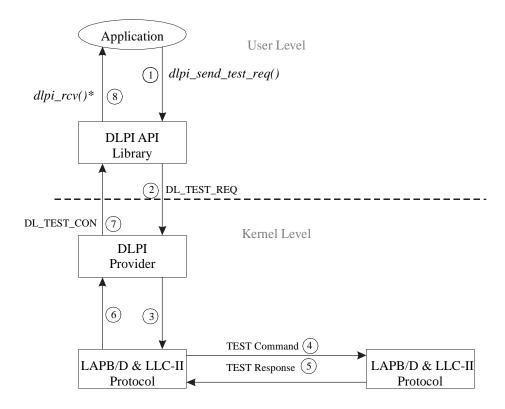
^{*} dlpi_read_data() can also substitute for events 7 and 14 if you are using the unnumbered frame handler routine.

Figure 14 XID service for connectionless mode LAPB/LAPD, LLC-II and HDLC/SDLC

5 · CONNECTIONLESS SERVICES GCOM, INC.

TEST Service

The local DLS user requests that TEST command response be sent to the remote DLS user. According to the DLPI specification, the optional feature that allows the DLS user to request automatic handling of the TEST response at bind time is handled by the DLPI Provider. GCOM's implementation makes this a required feature that is handled by the LAPB/LAPD, LLC-II and HDLC/SDLC protocols as shown in Figure 15.



^{*} dlpi_read_data() can also substitute for events 7 and 14 if you are using the unnumbered frame handler routine.

Figure 15 TEST service for connectionless mode LAPB/LAPD, LLC-II and HDLC/SDLC

Closing a Connectionless Session

To unbind, detach and close, use the *dlpi_close* command, which is documented in your UNIX system administrator's manual. Your application would seldom need to explicitly detach and unbind a session. However, the DLPI API Library supplies such routines if you want to use them.

5 · CONNECTIONLESS SERVICES GCOM, INC.

LAPB/LAPD and LLC-II Connection-Oriented Services

55	Understanding Connection-Oriented Data Transfer
56	Normal Connection Establishment for LAPB/LAPD and
	LLC-II
57	Normal Data Transfer for LAPB/LAPD and LLC-II
58	U-Frame Services for LAPB/LAPD and LLC-II
68	XID Service
69	TEST Service
70	Connection-Oriented Unnumbered Information U-Frame Data Transfer Service
71	Connection Establishment Rejections for LAPB/LAPD
	and LLC-II
71	Called DLS User Connection Rejection
72	DLS Provider Connection Rejection
73	Connection Retraction Services for LAPB/LAPD and
	LLC-II
73	User Retracts Connection Request
74	DL_DISCONNECT_IND Arrives After
	DL_CONNECT_RES is Sent
75	Connection Release Services for LAPB/LAPD and LLC-II
75	DLS User-Invoked Connection Release
76	Simultaneous DLS User-Invoked Connection Release
77	DLS Provider-Invoked Connection Release
78	Simultaneous DLS User- and DLS Provider-Invoked
70	Connection Release
79	Connection Reset Services for LAPB/LAPD and LLC-II
80	DLS User-Invoked Connection Reset
81	Simultaneous DLS User-Invoked Connection Reset
82	DLS Provider-Invoked Connection Reset

The connection-mode services enable a Data Link Service (DLS) user to establish a data link connection, transfer data over that connection, reset the link and release the connection when the conversation has terminated.



Note: For information about HDLC/SDLC connection-oriented services, refer to Section 4, page 83.

Understanding Connection-Oriented Data Transfer

Why? Connection-oriented data transfer normally implies three phases: the connection is set up, data are transferred and the connection is torn down. A connection offers the advantage of reliable data delivery across the network in its original sequence with no missing or duplicate frames/packets and no bit-errors. It is also flow-controlled. That is, the connection itself has a flow-control mechanism at the protocol level, normally implemented using sequence numbers, acknowledgements and windows within the definition of a frame level connection.

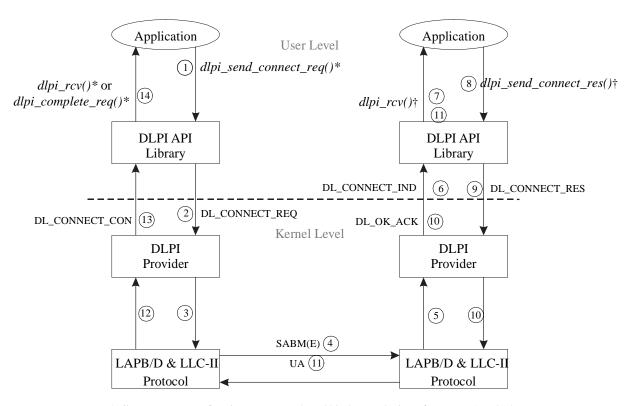
How? For successful connection-oriented data transfer to occur over DLPI, your application must achieve the following three phases:

- Setup the connection
- Exchange data
- End the session

Normal Connection Establishment for LAPB/LAPD and LLC-II

The connection establishment service establishes a data link connection between a local DLS user and a remote DLS user. Only one data link connection is allowed on each stream.

In the connection establishment model, the calling DLS user initiates connection establishment, while the called DLS user waits for incoming requests. DL_CONNECT_REQ requests that the DLS provider establish a connection. DL_CONNECT_IND informs the called DLS user of the request, which may be accepted using DL_CONNECT_RES or using DL_DISCONNECT_REQ. DL_CONNECT_CON informs the calling DLS user that a requested connection has been established. Figure 16, below, illustrates the normal sequence of connection messages.



^{*} dlpi_connect() performing open, attach and bind can substitute for events 1 and 14. dlpi_connect_req() can also substitute for events 1 and 14.

Figure 16 Successful connection establishment for LAPB/LAPD and LLC-II

[†] dlpi_listen() performing open, attach and bind can substitute for events 7, 8 and 11.

Normal Data Transfer for LAPB/LAPD and LLC-II

The connection-oriented data transfer service provides for the exchange of user data in either direction or in both directions simultaneously (full-duplex) between DLS users. Data are transmitted in logical groups called Data Link Service Data Units (DLSDUs). The DLS provider preserves both the sequence and boundaries of DLSDUs as they are transmitted.

Each call to *dlpi_write_data()* in Figure 17, below, conveys a DLSDU from the local DLS user to the DLS provider. Similarly, each call to *dlpi_read_data()* conveys a DLSDU from the DLS provider to the remote DLS user.

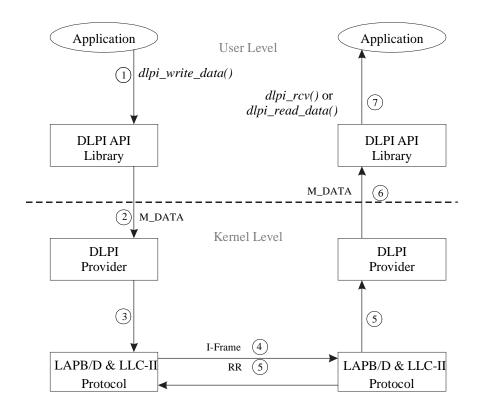


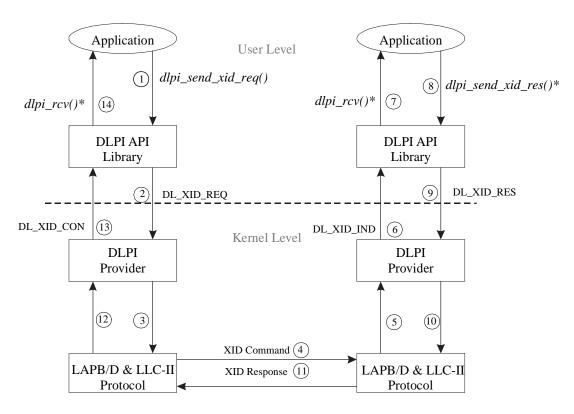
Figure 17 Normal data transfer for LAPB/LAPD and LLC-II

U-Frame Services for LAPB/LAPD and LLC-II

The GCOM DLPI API Library routines support U-frame service for connection-oriented protocols.

XID Service

The local DLS user requests that an XID command response be sent to the remote DLS user. The feature that allows the DLS user to request automatic handling of the XID response at bind time is not supported at this time.

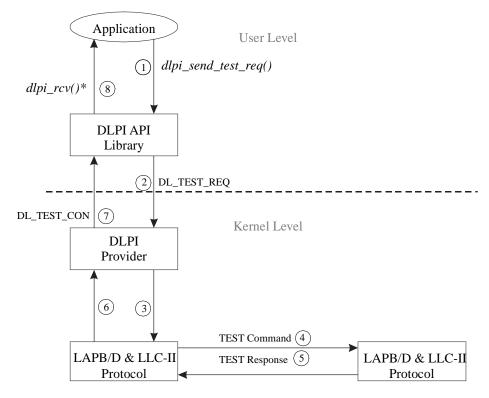


^{*} dlpi_read_data() can also substitute for events 7 and 14 if you are using the unnumbered frame handler routine.

Figure 18 XID service for connection-oriented LAPB/LAPD and LLC-II

TEST Service

The local DLS user requests that TEST command response be sent to the remote DLS user. According to the DLPI specification, the optional feature that allows the DLS user to request automatic handling of the TEST response at bind time is handled by the DLPI Provider. GCOM's implementation makes this a required feature that is handled by the LAPB/LAPD and LLC-II protocols in Figure 19.



^{*} dlpi_read_data() can also substitute for events 7 and 14 if you are using the unnumbered frame handler routine.

Figure 19 TEST service for connection-oriented LAPB/LAPD and LLC-II

Connection-Oriented Unnumbered Information U-Frame Data Transfer Service

Data are transmitted in logical groups called Data Link Service Data Units (DLSDUs). The DLS provider preserves both the sequence and boundaries of DLSDUs as they are transmitted. The amount of user data that can be transferred in a single DLSDU is limited.

Each call to *dlpi_send_uic()* in Figure 20, below, conveys a DLSDU from the local DLS user to the DLS provider. Similarly, each call to *dlpi_read_data()* conveys a DLSDU from the DLS provider to the remote DLS user.

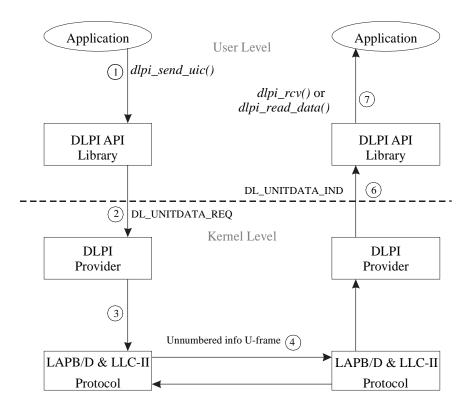


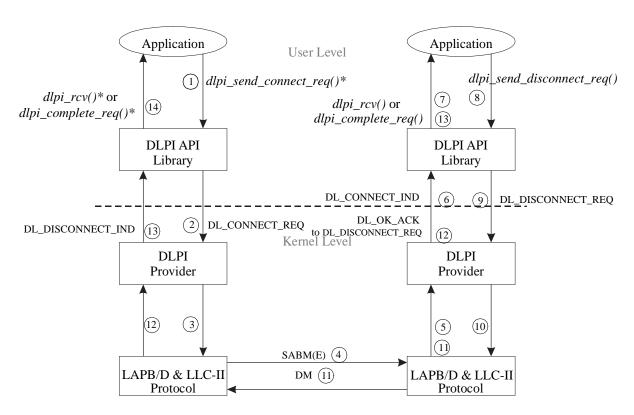
Figure 20 Unnumbered Information U-Frame Data Transfer Service for LAPB/LAPD and LLC-II

Connection Establishment Rejections for LAPB/LAPD and LLC-II

In certain situations, the connection establishment request cannot be completed. The following subsections describe instances where a DL_DISCONNECT_IND DLPI Provider primitive is flowing during the connection establishment phase. That is, a local DLS user issued a $dlpi_send_connect_req()$, $dlpi_connect()$ or $dlpi_connect_req()$, but the peer rejected the connection request.

Called DLS User Connection Rejection

In Figure 21, below, the DLS user chooses to reject the connect request by issuing a DL_DISCONNECT_REQ primitive instead of a DL_CONNECT_RES.

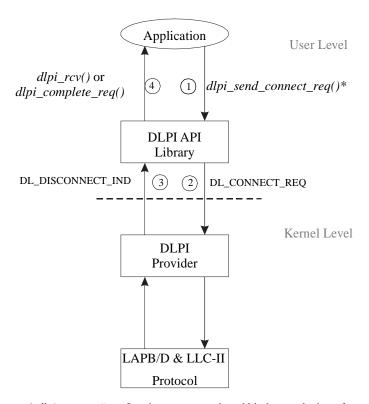


^{*} dlpi_connect() performing open, attach and bind can substitute for events 1 and 14. dlpi_connect_req() can also substitute for events 1 and 14.

Figure 21 Called DLS user connection rejection for LAPB/LAPD and LLC-II

DLS Provider Connection Rejection

In Figure 22, below, the DLS provider rejects a connect request for lack of resources or other reasons. The DLS provider sends DL_DISCONNECT_IND in response to the connect request.



^{*} dlpi_connect() performing open, attach and bind can substitute for event 1. dlpi_connect_req() can also substitute for event 1.

Figure 22 DLS provider connection rejection for LAPB/LAPD and LLC-II

Connection Retraction Services for LAPB/LAPD and LLC-II

The next two figures show scenarios where the calling DLS user chooses to abort a previous connection attempt. That is, the DLS user issues a disconnect request after a connect request. The resulting sequence of events depends on the relative timing of the DLPI primitives involved.

User Retracts Connection Request

Figure 23, below, shows a scenario where the retraction occurs before the local DLS provider can send a SABM(E) to the remote DLS provider.

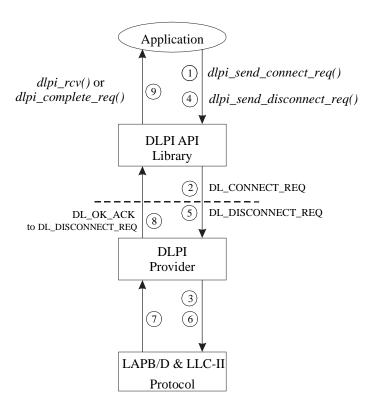
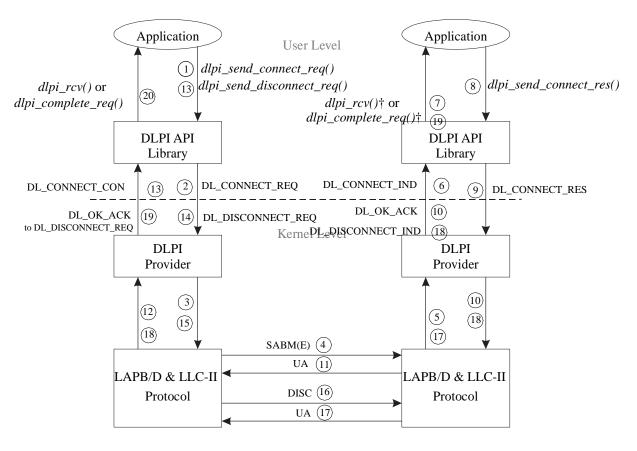


Figure 23 User retracts connection request for LAPB/LAPD and LLC-II

DL_DISCONNECT_IND Arrives After DL_CONNECT_RES is Sent

Figure 24, below, shows a scenario where the calling DLS user chooses to abort a previous connection attempt. That is, the DLS user issues a disconnect request after a connect request. The resulting sequence of events depends on the relative timing of the DLPI primitives involved.



† dlpi_listen() performing open, attach and bind can substitute for events 7 and 19.

Figure 24 DL_DISCONNECT_IND arrives after DL_CONNECT_RES is sent for LAPB/LAPD and LLC-II

The following network events in Figure 24 are defined as follows:

event 12 Connect sequence ends here

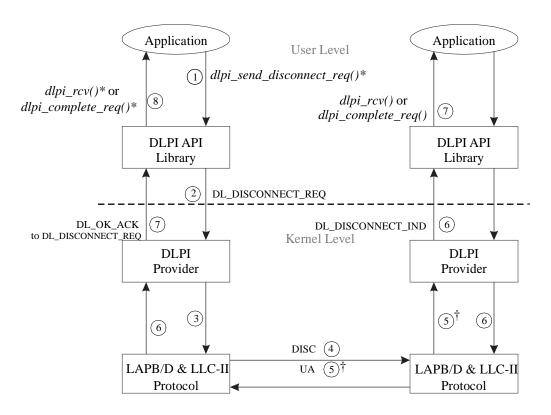
event 13 Disconnect starts here

Connection Release Services for LAPB/LAPD and LLC-II

The connection release service provides for the DLS users or the DLS provider to initiate the connection release. Connection release is an abortive operation, and any data in transit that has not been delivered to the DLS user will be discarded. A DL_DISCONNECT_REQ primitive requests that a connection be released and DL_DISCONNECT_IND informs the DLS user that a connection has been release.

DLS User-Invoked Connection Release

Normally, one DLS user requests disconnection and the DLS provider issues an indication of the ensuing release to the other DLS user, as shown in Figure 25, below.



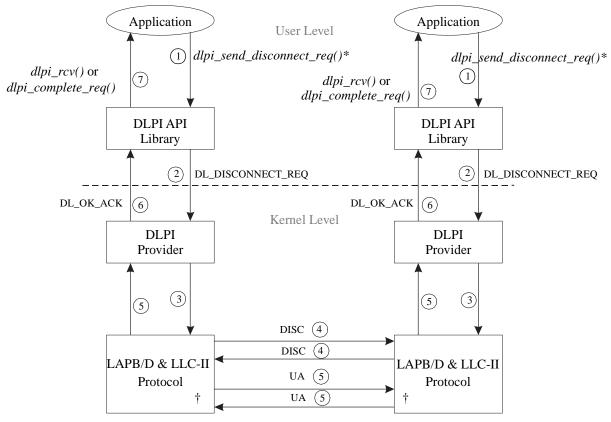
^{*} *dlpi_disconnect_req()* can also substitute for events 1 and 8.

Figure 25 DLS user-invoked connection release for LAPB/LAPD and LLC-II

[†] The time sequence runs uncoupled starting at event 5.

Simultaneous DLS User-Invoked Connection Release

Figure 26, below, illustrates that when two DLS users independently invoke the connection release service, neither one receives a DL_DISCONNECT_IND primitive.



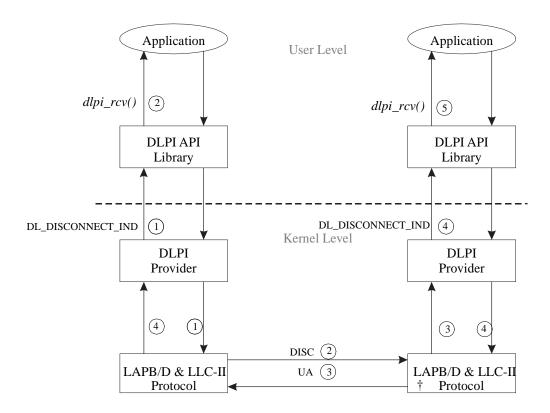
^{*} *dlpi_disconnect_req()* can also substitute for events 1 and 7.

Figure 26 Simultaneous DLS user-invoked connection release for LAPB/LAPD and LLC-II

[†] Rendezvous. Uncoupled after this point.

DLS Provider-Invoked Connection Release

Figure 27, below, illustrates that when a DLS provider initiates the connection release service, each DLS user receives a DL_DISCONNECT_IND primitive.

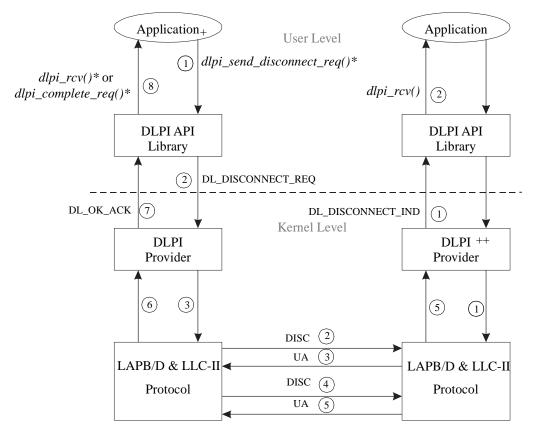


† Events are uncoupled after this point.

Figure 27 DLS provider-invoked connection release for LAPB/LAPD and LLC-II

Simultaneous DLS User- and DLS Provider-Invoked Connection Release

Figure 28, below, shows that when the remote DLS provider and the local DLS user simultaneously invoke the connection release service, the remote DLS user receives a DL_DISCONNECT_IND primitive.



^{*} dlpi_disconnect_req() can also substitute for events 1 and 7.

Figure 28 Simultaneous DLS user- (+) and DLS provider- (++) invoked connection release for LAPB/LAPD and LLC-II

Connection Reset Services for LAPB/LAPD and LLC-II

The *reset service* can be used by the DLS user to resynchronize the use of a data link connection or by the DLS provider to report detected loss of data that is unrecoverable within the data link service. The invocation of the reset service unblocks the flow of Data Link Service Data Units (DLSDUs) if the data link connection is congested. DLSDUs can be discarded by the DLS provider. The DLS users that did not invoke the reset will be notified that a reset has occurred. A reset may require a recovery procedure to be performed by the DLS users.

Interaction between DLS user and DLS provider

The interaction between each DLS user and the DLS provider will be one of the following:

- A reset request is invoked by the DLS user, followed by a reset confirmation from the DLS provider
- A reset indication from the DLS provider, followed by a reset response from the DLS user and an DL OK ACK returned to the DLS user

Primitives involved in the connection reset service

The DL_RESET_REQ acts as a synchronization mark in the streams of DLSDUs that are transmitted by the issuing DLS user. The DL_RESET_IND acts as a synchronization mark in the stream, of DLSDUs that are received by the peer DLS user. Similarly, the DL_RESET_RES acts as a synchronization mark in the stream, of DLSDUs that are transmitted by the responding DLS user. The DL_RESET_CON acts as a synchronization mark in the stream, of DLSDUs that are received by the DLS user that originally issued the reset.

Resynchronizing properties of the reset service

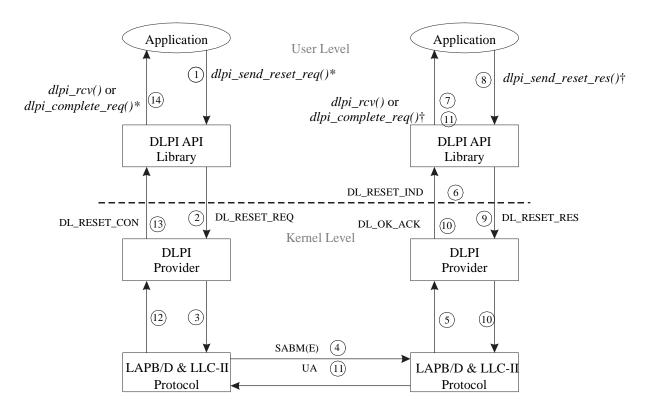
These are the resynchronizing properties of the reset service:

- No DLSDU transmitted by the DLS user, *before* the synchronization mark in that transmitted stream, will be delivered to the other DLS user *after* the synchronization mark in that received stream.
- The DLS provider will discard all DLSDUs submitted before the issuing of the DL_RESET_REQ that have not been delivered to the peer DLS user when the DLS provider issues the DL_RESET_IND.
- The DLS provider discards all DLSDUs submitted before the issuing
 of the DL_RESET_RES that have not been delivered to the initiator of
 the DL_RESET_REQ when the DLS provider issues the
 DL_RESET_CON.
- No DLSDU transmitted by the DLS user after the synchronization mark in that transmitted stream will be delivered to the other DLS user before the synchronization mark in that received stream

The complete message flow depends upon the origin of the reset, which may be the DLS provider or either DLS user.

DLS User-Invoked Connection Reset

Figure 29, below, illustrates the message flow for a reset invoked by one DLS user.



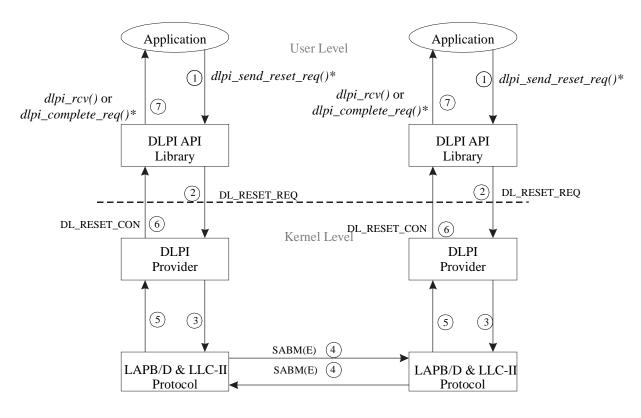
^{*} dlpi_reset_req() can also substitute for events 1 and 14.

Figure 29 DLS user-invoked connection reset for LAPB/LAPD and LLC-II

[†] dlpi_reset_res() can also substitute for events 8 and 11.

Simultaneous DLS User-Invoked Connection Reset

Figure 30, below, illustrates the message flow for a reset invoked by both DLS users simultaneously.

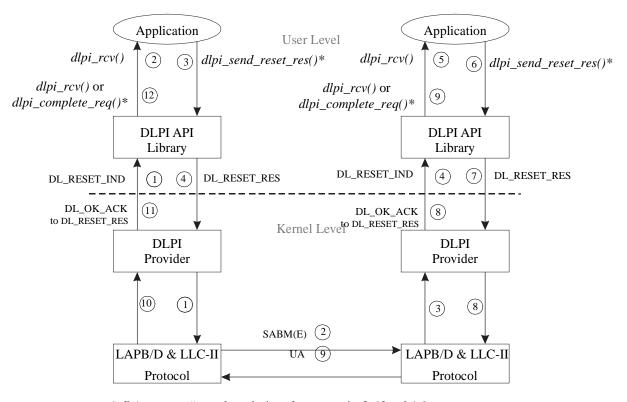


^{*} *dlpi_reset_req()* can also substitute for events 1 and 7.

Figure 30 Simultaneous DLS user-invoked connection reset for LAPB/LAPD and LLC-II

DLS Provider-Invoked Connection Reset

Figure 31, below, illustrates the message flow for a reset invoked by the DLS provider.



^{*} dlpi_reset_res() can also substitute for event pairs 3, 12 and 6, 9.

Figure 31 DLS provider-invoked connection reset for LAPB/LAPD and LLC-II

HDLC/SDLC Connection-Oriented Services

85	Understanding Connection-Oriented Data Transfer
86	Normal Connection Establishment for HDLC/SDLC
88	Normal Data Transfer for HDLC/SDLC
89	U-Frame Service for HDLC/SDLC
89	XID Service
90	TEST Service
91	Connection-Oriented Unnumbered Information U-Frame Service for HDLC/SDLC Primary
92	Connection-Oriented Unnumbered Information U-Frame Service for HDLC/SDLC Secondary
93	Connection Establishment Rejections for HDLC/SDLC
93	Called DLS User Connection Rejection
94	DLS Provider Connection Rejection
95	Connection Retraction for HDLC/SDLC
95	User Retracts Connection Request
96	DL_DISCONNECT_IND Arrives After DL_CONNECT_RES is Sent
97	Connection Release Services for HDLC/SDLC
97	DLS User-Invoked Connection Release for HDLC/SDLC Primary
98	DLS User-Invoked Connection Release for HDLC/SDLC Secondary
99	Simultaneous DLS User-Invoked Connection Release
100	DLS Provider-Invoked Connection Release for HDLC/SDLC Primary
101	DLS Provider-Invoked Connection Release for HDLC/SDLC Secondary
102	Simultaneous DLS User- and DLS Provider-Invoked Connection Release for HDLC/SDLC Primary
103	Simultaneous DLS User- and DLS Provider-Invoked Connection Release for HDLC/SDLC Secondary

104	Connection Reset Service for HDLC/SDLC
105	DLS User-Invoked Connection Reset for HDLC/SDLC Primary
106	DLS User-Invoked Connection Reset for HDLC/SDLC Secondary
107	Simultaneous DLS User-Invoked Connection Reset

The connection-mode services enable a Data Link Service (DLS) user to establish a data link connection, transfer data over that connection, reset the link and release the connection when the conversation has terminated.



Note: For information about LAPB/LAPD and LLC-II connection-oriented services, refer to Section 3, page 63.

Understanding Connection-Oriented Data Transfer

Why? Connection-oriented data transfer normally implies three phases: the connection is set up, data are transferred and the connection is torn down. A connection offers the advantage of reliable data delivery across the network in its original sequence with no missing or duplicate frames/packets and no bit-errors. It is also flow-controlled. That is, the connection itself has a flow-control mechanism at the protocol level, normally implemented using sequence numbers, acknowledgements and windows within the definition of a frame level connection.

How? For successful connection-oriented data transfer to occur over DLPI, your application must achieve the following three phases:

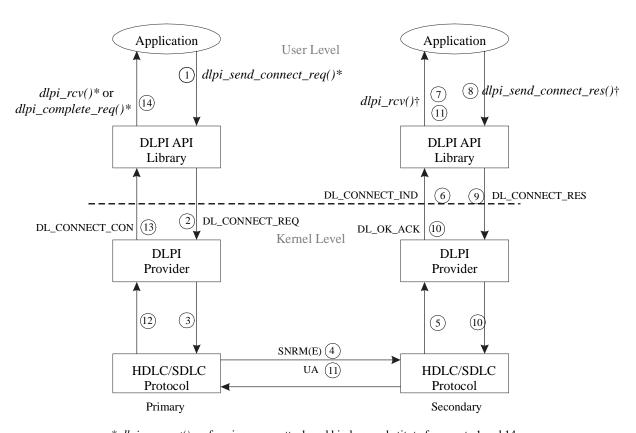
- Setup the connection
- Exchange data
- End the session

Normal Connection Establishment for HDLC/SDLC

The connection establishment service establishes a data link connection between a local DLS user and a remote DLS user. Only one data link connection is allowed on each stream.

In the connection establishment model, the calling DLS user initiates connection establishment, while the called DLS user waits for incoming requests. DL_CONNECT_REQ requests that the DLS provider establish a connection. DL_CONNECT_IND informs the called DLS user of the request, which may be accepted using DL_CONNECT_RES or using DL_DISCONNECT_REQ. DL_CONNECT_CON informs the calling DLS user that a requested connection has been established.

Figure 32, below, illustrates the normal sequence of connection messages.



^{*} dlpi_connect() performing open, attach and bind can substitute for events 1 and 14. dlpi_connect_req() can also substitute for events 1 and 14.

Figure 32 Successful connection establishment for HDLC/SDLC

[†] dlpi_listen() performing open, attach and bind can substitute for events 7, 8 and 11.

Normal Data Transfer for HDLC/SDLC

The connection-oriented data transfer service provides for the exchange of user data in either direction or in both directions simultaneously (full-duplex) between DLS users. Data are transmitted in logical groups called Data Link Service Data Units (DLSDUs). The DLS provider preserves both the sequence and boundaries of DLSDUs as they are transmitted.

Each call to *dlpi_write_data()* in Figure 33 conveys a DLSDU from the local DLS user to the DLS provider. Similarly, each call to *dlpi_read_data()* conveys a DLSDU from the DLS provider to the remote DLS user.

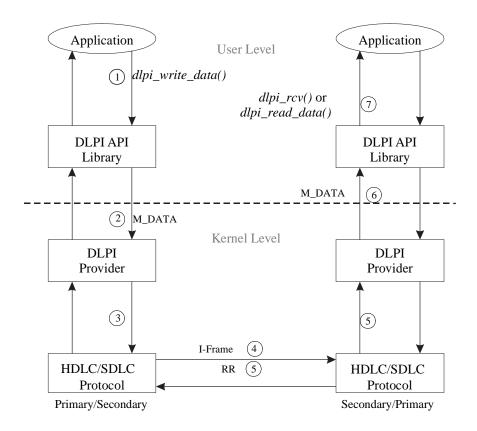


Figure 33 Normal data transfer for HDLC/SDLC

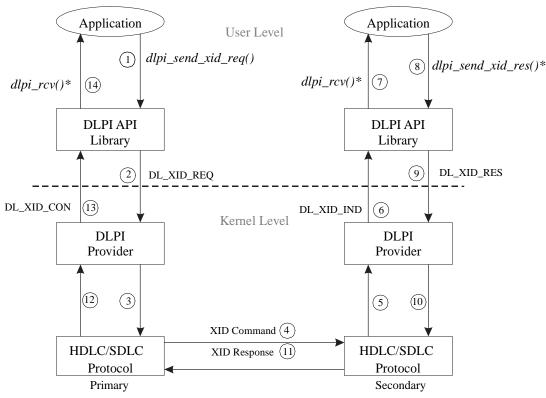
GCOM, INC. U-FRAME SERVICE FOR HDLC/SDLC

U-Frame Service for HDLC/SDLC

The GCOM DLPI API Library routines support U-frame service for XID and TEST commands and responses.

XID Service

The local DLS user requests that an XID command response be sent to the remote DLS user. The feature that allows the DLS user to request automatic handling of the XID response at bind time is not supported.

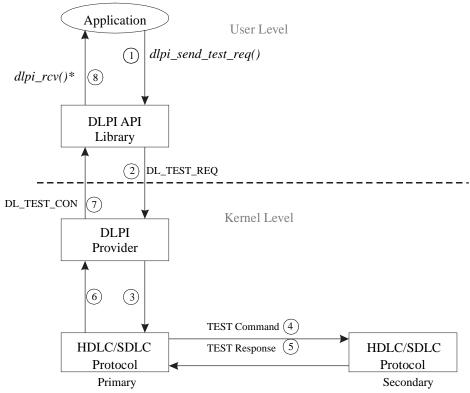


^{*} dlpi_read_data() can also substitute for events 7 and 14 if you are using the unnumbered frame handler routine.

Figure 34 XID service for connection-oriented HDLC/SDLC

TEST Service

The local DLS user requests that TEST command response be sent to the remote DLS user. According to the DLPI specification, the optional feature that allows the DLS user to request automatic handling of the TEST response at bind time is handled by the DLPI Provider. GCOM's implementation makes this a required feature that is handled by the HDLC/SDLC protocols in Figure 35, below.



^{*} *dlpi_read_data()* can also substitute for events 7 and 14 if you are using the unnumbered frame handler routine.

Figure 35 TEST service for connection-oriented HDLC/SDLC

GCOM, INC. U-FRAME SERVICE FOR HDLC/SDLC

Connection-Oriented Unnumbered Information U-Frame Service for HDLC/SDLC Primary

Data are transmitted in logical groups called Data Link Service Data Units (DLSDUs). The DLS provider preserves both the sequence and boundaries of DLSDUs as they are transmitted. The amount of user data that can be transferred in a single DLSDU is limited.

Each call to *dlpi_send_uic()* in Figure 36, below, conveys a DLSDU from the local DLS user to the DLS provider. Similarly, each call to *dlpi_read_data()* conveys a DLSDU from the DLS provider to the remote DLS user.

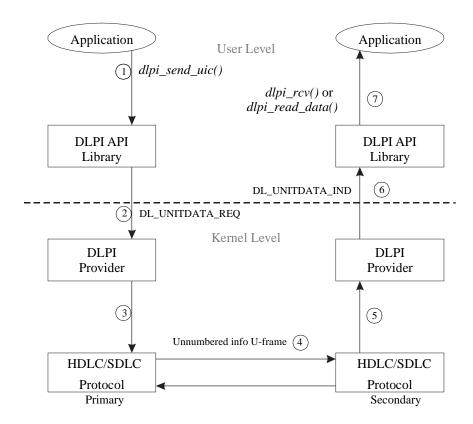


Figure 36 Unnumbered information u-frame service for connection-oriented HDLC/SDLC primary

Connection-Oriented Unnumbered Information U-Frame Service for HDLC/SDLC Secondary

Data are transmitted in logical groups called Data Link Service Data Units (DLSDUs). The DLS provider preserves both the sequence and boundaries of DLSDUs as they are transmitted. The amount of user data that can be transferred in a single DLSDU is limited.

Each call to *dlpi_send_uic()* in Figure 37, below, conveys a DLSDU from the local DLS user to the DLS provider. Similarly, each call to *dlpi_read_data()* conveys a DLSDU from the DLS provider to the remote DLS user.

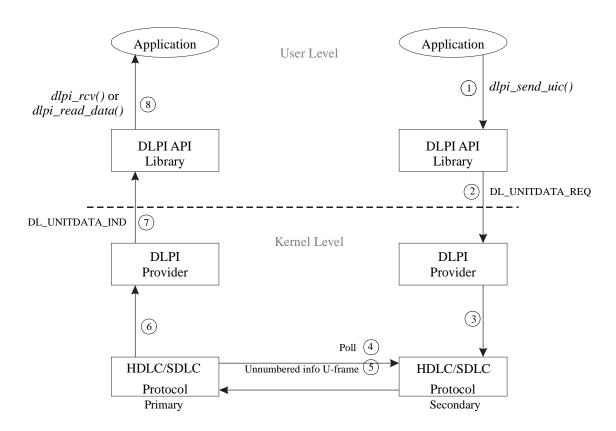


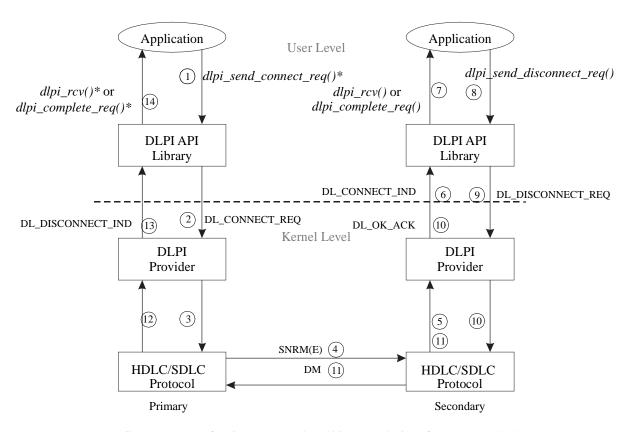
Figure 37 Unnumbered information u-frame service for connectionoriented HDLC/SDLC secondary

Connection Establishment Rejections for HDLC/SDLC

In certain situations, the connection establishment request cannot be completed. The following subsections describe instances where a DL_DISCONNECT_IND DLPI Provider primitive is flowing during the connection establishment phase. That is, a local DLS user issued a $dlpi_send_connect_req()$, $dlpi_connect()$ or $dlpi_connect_req()$, but the peer rejected the connection request.

Called DLS User Connection Rejection

In Figure 38, below, the DLS user chooses to reject the connect request by issuing a DL_DISCONNECT_REQ primitive instead of a DL_CONNECT_RES.

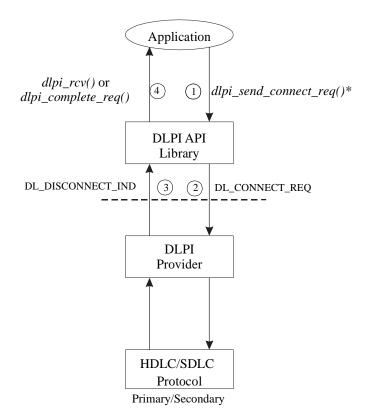


^{*} dlpi_connect() performing open, attach and bind can substitute for events 1 and 14. dlpi_connect_req() can also substituted for events 1 and 14.

Figure 38 Called DLS user connection rejection for HDLC/SDLC

DLS Provider Connection Rejection

In Figure 39, below, the DLS provider rejects a connect request for lack of resources or other reasons. The DLS provider sends DL_DISCONNECT_IND in response to the connect request.



^{*} dlpi_connect() performing open, attach and bind can substitute for event 1. dlpi_connect_req() can also substitute for event 1.

Figure 39 DLS provider connection rejection for HDLC/SDLC

Connection Retraction for HDLC/SDLC

The next subsections show scenarios where the calling DLS user chooses to abort a previous connection attempt. That is, the DLS user issues a disconnect request after a connect request. The resulting sequence of events depends on the relative timing of the DLPI primitives involved.

User Retracts Connection Request

Figure 40, below, shows a situation where the retraction takes place before a SNRM(E) can be sent to the remote DLS provider.

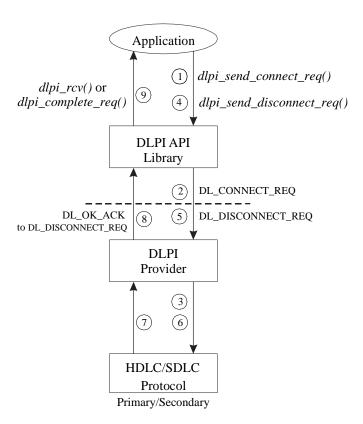


Figure 40 User retracts connection request for HDLC/SDLC

DL_DISCONNECT_IND Arrives After DL_CONNECT_RES is Sent

Figure 41 shows a scenario where the calling DLS user chooses to abort a previous connection attempt. That is, the DLS user issues a disconnect request after a connect request. The resulting sequence of events depends on the relative timing of the DLPI primitives involved. Figure 41 shows a situation where both primitives are destroyed by the provider.

Figure 41 DL_DISCONNECT_IND Arrives After DL_CONNECT_RES is Sent for HDLC/SDLC

The following network events in Figure 41 are defined as follows:

event 12 Connect sequence ends here

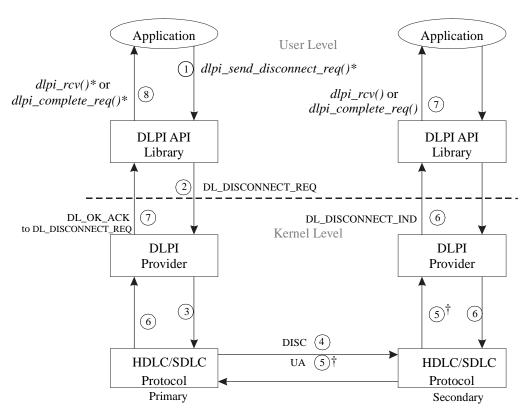
event 13 Disconnect starts here

Connection Release Services for HDLC/SDLC

The connection release service provides for the DLS users or the DLS provider to initiate the connection release. Connection release is an abortive operation, and any data in transit that has not been delivered to the DLS user will be discarded. A DL_DISCONNECT_REQ primitive requests that a connection be released and DL_DISCONNECT_IND informs the DLS user that a connection has been released.

DLS User-Invoked Connection Release for HDLC/SDLC Primary

Normally, one DLS user requests disconnection and the DLS provider issues an indication of the ensuing release to the other DLS user, as shown in Figure 42, below.



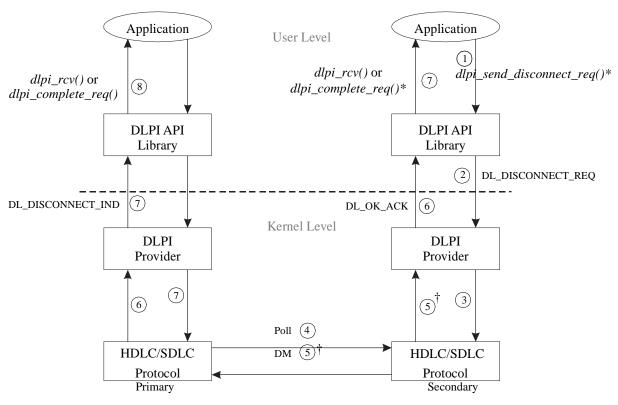
^{*} dlpi_disconnect_req() can also substitute for events 1 and 8.

Figure 42 DLS user-invoked connection release for HDLC/SDLC primary

[†] The time sequence runs uncoupled starting at event 5.

DLS User-Invoked Connection Release for HDLC/SDLC Secondary

Normally, one DLS user requests disconnection and the DLS provider issues an indication of the ensuing release to the other DLS user, as shown in Figure 43, below.



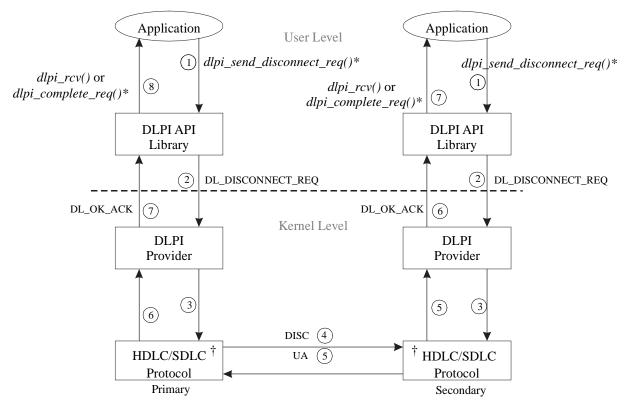
^{*} *dlpi_disconnect_req()* can also substitute for event 1 and 7.

Figure 43 DLS user-invoked connection release for HDLC/SDLC secondary

[†] The time sequence runs uncoupled starting at event 5.

Simultaneous DLS User-Invoked Connection Release

Figure 44, below, illustrates that when two DLS users independently invoke the connection release service, neither one receives a DL_DISCONNECT_IND primitive.



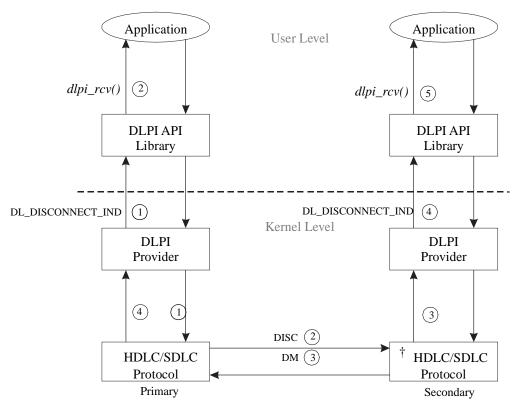
^{*} dlpi_disconnect_req() can also substitute for events 1, 7 and 1, 8.

Figure 44 Simultaneous DLS user-invoked connection release for HDLC/SDLC

[†] Rendezvous. Uncoupled after this point.

DLS Provider-Invoked Connection Release for HDLC/SDLC Primary

Figure 45, below, illustrates that when the DLS provider initiates the connection release service, each DLS user receives a DL_DISCONNECT_IND primitive.

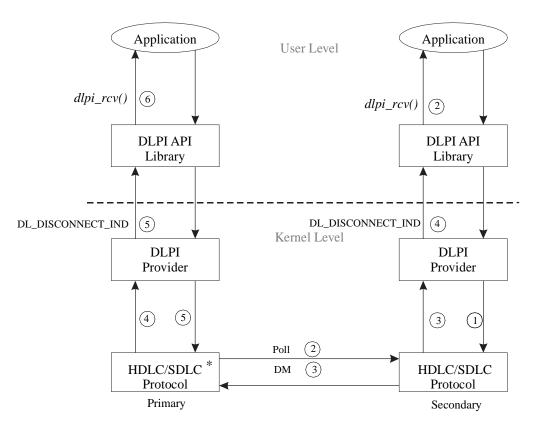


[†] Events are uncoupled after this point.

Figure 45 DLS provider-invoked connection release for HDLC/SDLC primary

DLS Provider-Invoked Connection Release for HDLC/SDLC Secondary

Figure 46, below, illustrates that when the DLS provider initiates the connection release service, each DLS user receives a DL_DISCONNECT_IND primitive.



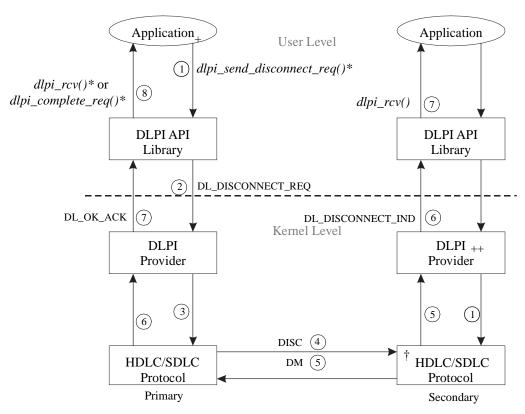
[†] Wait for poll.

Figure 46 DLS provider-invoked connection release for HDLC/SDLC secondary

^{*} Events are uncoupled after this point.

Simultaneous DLS User- and DLS Provider-Invoked Connection Release for HDLC/SDLC Primary

Figure 47, below, shows that when the remote DLS provider and the local DLS user simultaneously invoke the connection release service, the remote DLS user receives a DL_DISCONNECT_IND primitive.



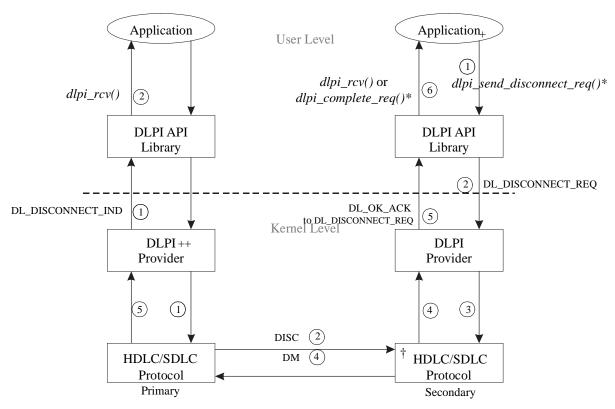
^{*} *dlpi_disconnect_req()* can also substitute for events 1 and 8.

Figure 47 Simultaneous DLS User- (+) and DLS provider- (++) invoked connection release for HDLC/SDLC primary

[†] Wait for Poll/DISC.

Simultaneous DLS User- and DLS Provider-Invoked Connection Release for HDLC/SDLC Secondary

Figure 48, below, shows that when the remote DLS provider and the local DLS user simultaneously invoke the connection release service, the remote DLS user receives a DL_DISCONNECT_IND primitive.



^{*} *dlpi_disconnect_req()* can also substitute for events 1 and 7.

Figure 48 Simultaneous DLS user- (+) and DLS provider- (++) invoked connection release for HDLC/SDLC secondary

[†] Wait for poll; disconnect received. Events are uncoupled after this point.

Connection Reset Service for HDLC/SDLC

The *reset service* can be used by the DLS user to resynchronize the use of a data link connection or by the DLS provider to report detected loss of data that is unrecoverable within the data link service. The invocation of the reset service unblocks the flow of Data Link Service Data Units (DLSDUs) if the data link connection is congested. DLSDUs can be discarded by the DLS provider. The DLS users that did not invoke the reset will be notified that a reset has occurred. A reset may require a recovery procedure to be performed by the DLS users.

Interaction between DLS user and DLS provider

The interaction between each DLS user and the DLS provider will be one of the following:

- A reset request is invoked by the DLS user, followed by a reset confirmation from the DLS provider
- A reset indication from the DLS provider, followed by a reset response from the DLS user and an DL_OK_ACK returned to the DLS user

Primitives involved in the connection reset service

The DL_RESET_REQ acts as a synchronization mark in the streams of DLSDUs that are transmitted by the issuing DLS user. The DL_RESET_IND acts as a synchronization mark in the stream of DLSDUs that are received by the peer DLS user. Similarly, the DL_RESET_RES acts as a synchronization mark in the stream of DLSDUs that are transmitted by the responding DLS user. The DL_RESET_CON acts as a synchronization mark in the stream of DLSDUs that are received by the DLS user that originally issued the reset.

Resynchronizing properties of the reset service

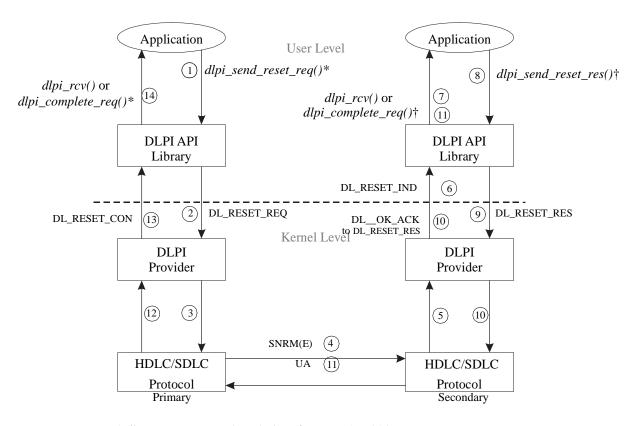
These are the resynchronizing properties of the reset service:

- No DLSDU transmitted by the DLS user *before* the synchronization mark in that transmitted stream will be delivered to the other DLS user *after* the synchronization mark in that received stream.
- The DLS provider will discard all DLSDUs submitted before the issuing of the DL_RESET_REQ that have not been delivered to the peer DLS user when the DLS provider issues the DL_RESET_IND.
- The DLS provider discards all DLSDUs submitted before the issuing
 of the DL_RESET_RES that have not been delivered to the initiator of
 the DL_RESET_REQ when the DLS provider issues the
 DL_RESET_CON.
- No DLSDU transmitted by the DLS user *after* the synchronization mark in that transmitted stream will be delivered to the other DLS user *before* the synchronization mark in that received stream.

The complete message flow depends upon the origin of the reset, which might be the DLS provider or either DLS user.

DLS User-Invoked Connection Reset for HDLC/SDLC Primary

Figure 49 illustrates the message flow for a reset invoked by one DLS user.



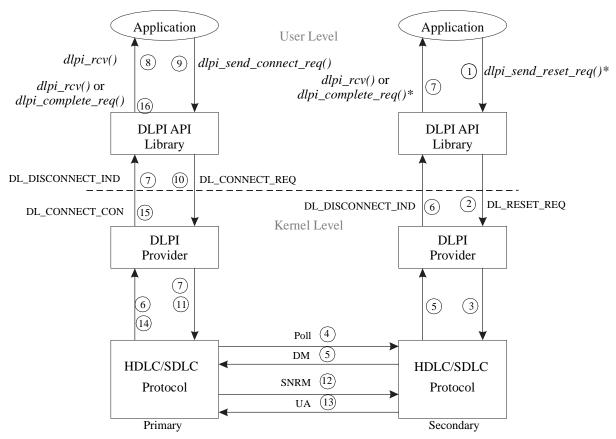
^{*} dlpi_reset_req() can also substitute for events 1 and 14.

Figure 49 DLS User-Invoked Connection Reset for HDLC/SDLC Primary

[†] dlpi_reset_res() can also substitute for events 8 and 11.

DLS User-Invoked Connection Reset for HDLC/SDLC Secondary

Figure 50 illustrates the message flow for a reset invoked by one DLS user. The HDLC/SDLC protocol does not support a secondary initiating a reset sequence. A DLPI provider reset request results in the SDLC connection being terminated. Figure 50 shows the primary side application responding to the disconnect indication with a connect request. The resulting connection setup is not shown for the secondary.

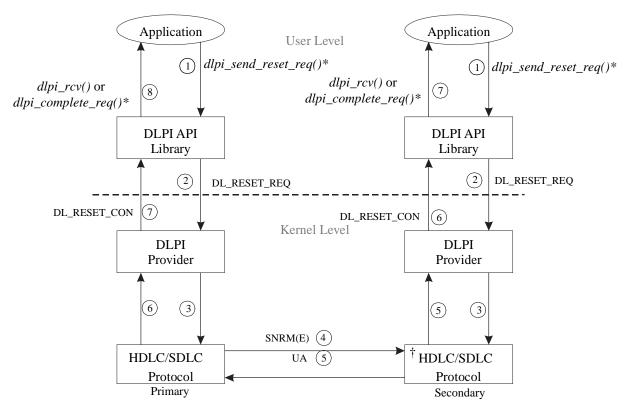


* *dlpi_reset_reg()* can also substitute for events 1 and 15.

Figure 50 DLS User-Invoked Connection Reset for HDLC/SDLC Secondary

Jser-Invoked Connection Reset

nessage flow for a reset invoked by both DLS



^{*} dlpi_reset_req() can also substitute for events 1, 7 and 1, 8.

Figure 51 Simultaneous DLS User-Invoked Connection Reset for HDLC/SDLC

[†] Here, the SNRM(E) arrives at the secondary before the DL_RESET_REQ (event 3).

Understanding the Sample Application

Figure 51 Sample DLPI API Program

```
1 #include <stdio.h>
2 #include "dlpiapi.h"
3 char
                buf[1000] ;
4 int
                fil ;
5 int
                n;
6 int
                frms ;
7
8 main()
9 {
10
       dlpi_init(DLPI_LOG_DEFAULT
                   | DLPI_LOG_RX_PROTOS | DLPI_LOG_TX_PROTOS
12
               ,DLPI_LOG_NAME
13
               ) ;
14
       fil = dlpi_listen(OL, OxO1L, OxO0L, OxO0L, DLPI_LISTEN_NO_FORK);
15
       if (fil < 0) exit(1);
16
       while ((n = dlpi_read_data(fil, buf, sizeof(buf))) >= 0)
17
18
           frms++ ;
           if (dlpi_write_data(fil, buf, n) < 0) exit(1);</pre>
19
20
21
       dlpi_printf("End of test: frames received = %d\n", frms) ;
22
       exit(0);
23 }
```

The test program in Figure 51 illustrates the use of the DLPI API library. The program, called *Gcom_lstn*, is a simple passive loopback program. It listens for any incoming connections, forks off a copy of itself to handle the incoming connection and loops any received data back to the sender.

Refer to Appendix A starting on page 125 for a list of programs you can call from the DLPI API Library.

The sample program on the opposite page contains the following features:

- Line 2 The program includes the **dlpiapi.h** file.
- Lines 10–13 The program initializes the DLPI API library. The first parameter to the *dlpi_init()* function specifies logging options to be used by the library. The set of options on line 11 make the logging verbose and unsuitable for production code. The second parameter to *dlpi_init()* is the name of the log file that the library is to use. The default name is /usr/spool/gcom/dlpi.log.
- Line 14 The program calls the *dlpi_listen()* function to wait for incoming calls. The *dlpi_listen()* function returns the file descriptor of a stream which has an open connection associated with it. The first parameter is zero, which indicates that the stream is to be attached to the default physical line. The second parameter is the bind DLSAP that is to be used for the stream. The third and fourth parameters are the local and remote addresses to configure for the stream. A zero value equals the default setting.

The fifth parameter is an option that specifies that the $dlpi_listen()$ function should not fork a new copy of itself to process the incoming connection. Thus, when $dlpi_listen()$ returns it will be on the same process as the original caller of $dlpi_listen()$.

Figure 51 Sample DLPI API Program (*repeated*)

```
1 #include <stdio.h>
2 #include "dlpiapi.h"
3 char
                 buf[1000] ;
4 int
                 fil ;
5 int
                 n;
6 int
                frms ;
7
8 main()
9 {
10
       dlpi_init(DLPI_LOG_DEFAULT
11
                   | DLPI_LOG_RX_PROTOS | DLPI_LOG_TX_PROTOS
12
               ,DLPI_LOG_NAME
13
               ) ;
14
       fil = dlpi_listen(0L, 0x01L, 0x00L, 0x00L, DLPI_LISTEN_NO_FORK);
15
       if (fil < 0) exit(1);
       while ((n = dlpi_read_data(fil, buf, sizeof(buf))) >= 0)
16
17
18
           frms++ ;
           if (dlpi_write_data(fil, buf, n) < 0) exit(1);</pre>
19
20
21
       dlpi_printf("End of test: frames received = %d\n", frms) ;
22
       exit(0);
23 }
```

Line 16 The program calls the *dlpi_read_data()* function to read a single data message from the stream. The function will return a negative number if a protocol message arrives or if the stream encounters an error.



Note: The parameters for dlpi_read_data() are the same as for the standard C library read routine.

Line 19 The program writes back any received data to the stream in loopback fashion.



Note: The parameters for dlpi_write_data() are the same as for the standard C library write routine.

Line 21 The program uses the *dlpi_printf()* routine to write a message into the log file. This routine has the same calling sequence as the C library *printf()* routine.

Understanding the DLPI API Header File

110	including the diplapi.h Header File in Your Application
116	Linking DLPI Header Files
117	DLPI API Constants
118	DLPI API Library Routine Error Return Defines
119	DLPI API Logging Options
120	Signal Handling Prototype
121	Global Variables Accessible By Your Application
121	dlpi_bind_ack
121	dlpi_conn_con
121	dlpi_conn_ind
121	dlpi_ctl_buf
122	dlpi_ctl_cnt
122	dlpi_data_buf
122	dlpi_data_cnt
123	DLPI API Fork Options

GCOM, INC.

The **dlpiapi.h** file contains defines, global variables and prototypes that can be used by the application program to communicate with the GCOM DLPI streams driver.

Including the dlpiapi.h Header File in Your Application

Prototypes for the API procedures and callbacks, as well as various useful macros and definitions of symbolic constants, are contained in the **dlpiapi.h** header file. This header files should be *#included* in application source files containing code that interacts with the API.

Linking DLPI Header Files

To link the header files with a program, issue the UNIX cc command using the following format:

cc -o program_name program_name.c /usr/lib/gcom/dlpiapi.a

Example The following command links header files for the *foo()* program:

> cc -o foo foo.c /usr/lib/gcom/dlpiapi.a

GCOM, INC. LINKING DLPI HEADER FILES

DLPI API Constants

The following constants parameterize the internal behavior of the DLPI API library. These constants were used when the library was compiled. Changing these in your application will not have the desired effect on the GCOM DLPI API Library.

DLPI_CTL_BUF_SIZE The size of the character array dlpi_ctl_buf

compiled into the DLPI API library. This array is available for use by the application (see below). Its size is larger than the largest possible streams message buffer. At this writing, it is set to 5,000.

DLPI_DATA_BUF_SIZEThe size of the character array dlpi_data_buf

compiled into the DLPI API library. This array is available for use by the application (see below). Its size is larger than the largest possible streams message buffer. At this writing, it is set to 5,000.

writing, it is set to 5,000.

DLPI_DEFAULT_PPA This is the default PPA used for the attach

phase. At this writing, this is an unsigned

long type with a value of ~ 0 .

DLPI_LOG_NAME The default name of the log file into which

the DLPI API library writes messages. At

this writing, this is set to /usr/spool/gcom/dlpi.log.

DLPI_N_CONINDS Number of incoming connection indications

that can be queued on a listener stream. This

constant is passed to DLPI in a

DL_BIND_REQ for a listener stream. Since the DLPI driver does not support connection manager streams, the maximum useful value is 1. The default value is set to

1.

DLPI API Library Routine Error Return Defines

When a DLPI API routine is called, a negative return value indicates an error. Many of those routines use the following return values when an error occurs:

DLPIAPI UNSUPPORTED MSG

A DLPI API Library routine received an inappropriate message from the data stream. For example, $dlpi_read_data()$ does not process reset messages.

DLPIAPI_NO_NOTHING

This is a hang-up notification. The UNIX *getmsg* system call succeeded but did not return control or normal data.

DLPIAPI_STYLE_UNKNOWN

Provider style unknown. This error should not occur.

DLPIAPI_PARAM_ERROR

Invalid parameter error. The buffer pointer and output pointer must be non-NULL and the buffer length must be a positive integer.

DLPIAPI NOT INIT

dlpi_init() has not been called.

DLPIAPI_OPEN_ERROR

Data stream open error.

DLPIAPI REJECT

Either *dlpi_complete_req()* was rejected with a DL_ERROR_ACK or *dlpi_connect_req()* was rejected with a DL_DISCONNECT_IND.

DLPIAPI_UNUSABLE

A UNIX streams *getmsg* or *putmsg* system call failed. One possible error recovery action is to close the stream and start over.

DLPIAPI EINTR

A UNIX system call interrupted; retry not provided.

DLPIAPI EAGAIN

Non-blocking I/O is set on the stream and no DLPI messages are available to process.

DLPI API Logging Options

The DLPI API logging options control the kinds of messages that are written into the DLPI API log file. These options represent individual bits of a mask. They are ordered together to form the set of options passed to *dlpi_init()*.

DLPI_LOG_FILE Write messages to the log file.

DLPI_LOG_STDERR Write messages to *stderr*.

DLPI_LOG_RX_PROTOS

Log all received protocol messages (M_PROTOS) in ASCII decoded form.

DLPI_LOG_TX_PROTOS

Log all transmitted protocol messages (M_PROTOS) in ASCII decoded form.

DLPI_LOG_ERRORS Log all UNIX error messages in a manner

similar to *perror*.

DLPI_LOG_SIGNALS

Log signal handling.

DLPI_LOG_DISCONNECTS

Log disconnect indications.

DLPI_LOG_RESETS Log reset indications.

DLPI_LOG_DEFAULT

The default settings of logging options,

consisting of DLPI_LOG_FILE,

DLPI_LOG_STDERR, DLPI_LOG_ERRORS,

DLPI_LOG_DISCONNECTS and

DLPI_LOG_RESETS.

Signal Handling Prototype

This is the typedef for the user-supplied function passed to $dlpi_set_signal_handling()$:

When called, the value returned from the user-supplied function is currently ignored by the API. Returning zero is recommended.

Global Variables Accessible By Your Application

The DLPI API package makes a certain set of its global variables available to the user. The variables and their intended uses are defined in the following subsections.

dlpi_bind_ack

Prototype

```
unsigned char dlpi_bind_ack[];
```

This array contains a copy of the DL_BIND_ACK received by the DLPI API routines. You may examine it for whatever information you may find useful.

dlpi_conn_con

Prototype

```
unsigned char dlpi_conn_con[];
```

This array contains a copy of the DL_CONNECT_CON (connect confirm) received by the DLPI API library in response to a DL_CONNECT_REQ sent by the library (active connection).

dlpi_conn_ind

Prototype

```
unsigned char dlpi conn ind[];
```

This array contains a copy of the DL_CONNECT_IND received by the DLPI API routines. When forking child processes, each child will have its own copy of this protocol message.

dlpi_ctl_buf

Prototype

```
unsigned char dlpi_ctl_buf[DLPI_CTL_BUF_SIZE];
```

This array is used by the DLPI API driver to receive the M_PROTO portion of messages from the DLPI driver. It will contain the most recent M_PROTO received by the DLPI API library. A call to dlpi_read_data() will overwrite this area with an invalid value; upon return, if the PRIM_type field does not contain all ones, then an M_PROTO was received while reading data.

dlpi_ctl_cnt

Prototype

```
int dlpi ctl cnt;
```

This variable contains the length of the M_PROTO portion of the last message received from the DLPI driver. If the last received message is a data-type message, then this count will be negative. A call to $dlpi_read_data()$ may fail because of a received prototype message. In such a case, the count in $dlpi_ctl_cnt$ will be positive and the protocol message will reside in the array $dlpi_ctl_buf$.

dlpi_data_buf

Prototype

```
unsigned char dlpi_data_buf[DLPI_DATA_BUF_SIZE] ;
```

This array may be used by the application program as a place to build data messages to send on a connection or as a place into which to receive data messages. It is an array that is made available for use to the application program.

dlpi_data_cnt

Prototype

```
int dlpi_data_cnt;
```

This variable contains the count of the number of bytes contained in the most recently sent or received data message from the DLPI driver.

dlpi_service_mode

Prototype

This variable can be set to the logical "or" of any of the following values from <sys/dlpi.h>:DL_CODLS (connection oriented service), DL_CLDLS (connection-less service), DL_ACLDLS (acknowledged connection-less service). The default value is DL_CODLS. DL_CODLS is appropriate for reliable link layer protocols such as HDLC, LAPB, LLC AND LAPD. DL_CLDLS is appropriate for non-reliable protocols such as frame relay. The value DL_ACLDLS is not supported by the GCOM DLPI provider.

The value of the dlpi_service_mode variable is used by the API routine dlpi_bind_dlsap in sending a DL_BIND_REQ to the DLPI provider.

DLPI API Fork Options

At this writing, the fork options for the DLPI API are reserved for future use. DLPI_LISTED_NO_FORK is the only valid option. This option returns without forking after accepting an incoming connection.

GCOM, INC.

APPENDIX A

DLPI API Library Routines Reference

127	dlp1_attach_ppa()
128	dlpi_bind_dlsap()
129	dlpi_clear_zombies()
130	dlpi_close()
131	dlpi_complete_req()
133	dlpi_configure_dlsaps()
134	dlpi_connect()
136	dlpi_connect_req()
137	dlpi_connect_wait()
138	dlpi_decode_ctl()
139	dlpi_decode_disconnect_reason()
140	dlpi_detach_ppa()
141	dlpi_discon_req()
142	dlpi_disconnect_req()
143	dlpi_get_a_msg()
145	dlpi_get_info()
146	dlpi_get_style()
147	dlpi_init()
148	dlpi_init_FILE()
149	dlpi_listen()
152	dlpi_open()
153	dlpi_open_log()
154	dlpi_perror()
155	dlpi_print_msg()
156	dlpi_printf()
157	dlpi_put_both()

```
158
           dlpi_put_proto()
159
           dlpi_rcv()
159
              Receiving U-Frames
160
              Non-Blocking I/O
169
           dlpi_rcv_msg()
171
           dlpi_read_data()
172
           dlpi_reset_req()
173
           dlpi_reset_res()
174
           dlpi_send_attach_req()
175
           dlpi_send_bind_req()
176
           dlpi_send_connect_req()
177
           dlpi_send_connect_res()
178
           dlpi_send_detach_req()
179
           dlpi_send_disconnect_req()
180
           dlpi_send_info_req()
181
           dlpi_send_reset_req()
182
           dlpi_send_reset_res()
183
           dlpi_send_stats_req()
184
           dlpi_send_test_req()
185
           dlpi_send_test_res()
186
           dlpi_send_uic()
187
           dlpi_send_unbind_req()
188
           dlpi_send_xid_req()
189
           dlpi_send_xid_res()
190
           dlpi_set_log_size()
191
           dlpi_set_signal_handling()
193
           dlpi_set_unnum_frame_handler()
194
           dlpi_test_req()
195
           dlpi_test_res()
196
           dlpi_uic_req()
197
           dlpi_unbind_dlsap()
198
           dlpi_xid_req()
199
           dlpi_xid_res()
200
           dlpi_write_data()
201
           dlpi_xray_req()
```

This section of the manual documents the routines that are provided in the **dlpiapi.a** library. Refer to Architecture of the Data Link Layer, page 26, for the sample DLPI API application.

dlpi_attach_ppa()

Include File(s) **<gcom/dlpiapi.h>**

Description This routine attaches the stream to a Physical Point of Attachment

(PPA). A PPA can be expressed as a UNIX multiplexor ID or a driver Lower Point of Attachment (LPA). This routine can be used for both

connectionless and connection-oriented data transfer mode.

Parameters dlpi_data stream file ID. This is the file descriptor returned by an

open call on the /dev/dlpi_clone file.

ppa If this is negative, it is an LPA. Otherwise, it is a streams

multiplexor ID.

Return < 0 The attach failed.

> 0 The attach succeeded.

dlpi_bind_dlsap()

Prototype int dlpi_bind_dlsap(int dlpi_data, unsigned long dlsap,

int conind_nr);

Include File(s) **<gcom/dlpiapi.h>**

Description This routine issues a DL_BIND_REQ to DLPI and waits for a

DL_BIND_ACK response. The bind acknowledgment is copied to the global array *dlpi_bind_ack* so that it can be viewed directly by the

application program.

Parameters dlpi_data stream over which to send the DL_BIND_REQ.

dlsap DLSAP address to be bound to the stream.

conind_nr Number of connection indications that can be queued for

this stream. If this number is non-zero, then the stream becomes a listener stream. That is, it becomes eligible as a target of in incoming connection. If this number is zero then this stream cannot be the target for an incoming connection but is to be used for an outgoing connection.

Return < 0 An error occurred.

> 0 Size of DL_BIND_ACK if success.

GCOM, INC. DLPI_BIND_DLSAP()

dlpi_clear_zombies()

Prototype void dlpi_clear_zombies(void) ;

Include File(s) **<gcom/dlpiapi.h>**

Description This routine can be used by a parent process to "clean up" after its

children. Under most Unix-like operating systems, when a parent process forks a child and the child exits, the operating system retains status information for the child. Until the parent process retrieves this status information, the child process is maintained in an expired state called a "zombie" state. This routine provides a convenient way to clean up zombie processes when the status information will not be checked.

dlpi_close()

Prototype int dlpi_close(int fd)

Include Files <gcom/dlpiapi.h>

Description Closes the file that was opened by $dlpi_open()$. This routine should be

used to close the file rather than calling the system "close" routine directly. The *dlpi_close* routine takes into account the possibility that the

file may be opened to a remote server via the Remote API.

Return 0 Success

-1 Failure. Upon failure, "errno" is set to indicate

the reason for failure

GCOM, INC. DLPI_BIND_DLSAP()

dlpi_complete_req()

Include File(s) <gcom/dlpiapi.h>, <gcom/dlpi.h>, <gcom/dlpi_ext.h>

Description This routine is used to wait for an acknowledge to the specified DLPI

primitive and report whether or not it was accepted by the driver. This procedure processes all incoming messages while waiting for the request to complete. The **dlpi.h** and **dlpi_ext.h** files are required so that the request defines will be properly translated into the numerical request

designation.

Parameters dlpi_data stream over which to send the DL_BIND_REQ.

request One of the following DLPI primitives:

• DL ATTACH REQ

• DL_DETACH_REQ

• DL BIND REQ

• DL_UNBIND_REQ

DL_CONNECT_REQ

• DL_CONNECT_RES

DL RESET REQ

DL_RESET_RES

DL_INFO_REQ

• DL_DISCONNECT_REQ

*caller Used in generating error messages.

discard_un_iframes If this flag is set, U-frames received while

waiting for the request to complete are discarded. Otherwise, if your application set up a U-frame handler routine, the frame is delivered to your

application using that routine.

Return	< 0	The <i>dlpi_complete_req()</i> routine returns interrupt system calls (EINTR). See for a complete list of the possible error return values.
	== 0	The request completed, but the stream is disconnected. This happens, for example, when a DL_CONNECT_REQ is rejected with a DL_DISCONNECT_IND.
	>0	DLPI primitive accepted by the DLPI provider.

GCOM, INC. DLPI_CONFIGURE_DLSAPS()

dlpi_configure_dlsaps()

Include File(s) **<gcom/dlpiapi.h>**

Description This routine configures the stream for the DLSAPs passed in as

parameters. When a DLPI data stream is in the DL_IDLE state (bound but not connected), the protocol's local and remote addresses can be set. An address of 0L for both *local_dlsap* and *remote_dlsap* implies that no

action is to be taken.

Once bound, the *local_dlsap* and *remote_dlsap* values are used to configure the addresses of the station. If the local and remote DLSAPs are zero, then the configured defaults for the station will be used. For an NRM primary or secondary, only one of the local or remote DLSAPs need be non-zero since these modes of operation require only one address. For ABM, both addresses must be supplied (or neither if using defaults).

Parameters dlpi_data stream to be configured

local_dlsap The DLSAP of the local system secondary address

remote_dlsap The DLSAP of the remote system primary address

Return < 0 Failure

> 0 Configuration succeeded

dlpi_connect()

Include File(s)

<gcom/dlpiapi.h>

Description

This routine opens a DLPI stream, binds it using the *bind_sap* and then uses *dlpi_connect_req()* to issue a DL_CONNECT_REQ to DLPI and wait for a DL_CONNECT_CON response. The connect confirm response is copied to the global array *dlpi_conn_con* so that it can be viewed directly by the application program.

The routine *dlpi_connect()* is equivalent to *dlpi_connect_host()*.

The ppa parameter specifies either the multiplexor ID or the Lower Point of Attachment (LPA) over which the connection is to be issued. If the number is negative, then the absolute value is assumed to be the LPA. Otherwise, it is assumed to be the multiplexor ID of the lower stream of the DLPI multiplexor. If the value 0L is passed, the default of -1L (line 1) is substituted.

The bind_dlsap is used to bind the stream to a station. Once bound, the local_dlsap and remote_dlsap values are used to configure the addresses of the station. If the local and remote DLSAPs are zero, then the configured defaults for the station will be used. For an NRM primary or secondary, only one of the local or remote DLSAPs need be non-zero since these modes of operation require only one address. For ABM both addresses must be supplied (or neither for using defaults).

For DLPI driver style 1, the DL_ATTACH_REQ is not sent to DLPI.

Parameters

ppa The multiplexor ID or LPA to which to attach the stream

bind_dlsap The DLSAP to use in the DL_BIND_REQ

local_dlsap The DLSAP to use as the local address

remote_dlsap The DLSAP of the remote system

GCOM, INC. DLPI_CONFIGURE_DLSAPS()



Note: All of these parameters are of type long.

Return < 0 An error occurred (message written to log file)

>= 0 File descriptor of an DLPI stream which can be used to send data

dlpi_connect_req()

Include File(s) **<gcom/dlpiapi.h>**

Description This routine issues a DL_CONNECT_REQ to DLPI and waits for a

DL_CONNECT_CON response. Your application must have

previously attached and bound the stream.

The connect confirm response is copied to the global array

dlpi_conn_con so that it can be viewed directly by the application

program.

Parameters dlpi_data stream over which to send the DL_CONDL_REQ

peer_sap Address of the peer to which you wish to be connected

Return < 0 An error occurred (message written to log file)

> 0 Size of DL_CONNECT_CON if success.

GCOM, INC. DLPI_CONNECT_REQ()

dlpi_connect_wait()

Prototype int dlpi_connect_wait(int data);

Include File(s) <gcom/dlpiapi.h>

Description The dlpi_connect_wait() routine provides a direct way to "listen" for

incoming connections on a data stream which has already been opened,

attached, and bound.

Parameters data The file descriptor for the data stream on which

connections should be received.

Return Values >0 File descriptor of the connected stream.

<0 An error occurred (message written to log file).</p>

dlpi_decode_ctl()

Prototype void dlpi_decode_ctl(char *p) ;

Include File(s) **<gcom/dlpiapi.h>**

Description This routine decodes the DLPI protocol message located in the global

dlpi_ctl_buf and writes it to the log file. The log message starts with prefix, which is typically the name of the procedure making the call.This routine is used mostly internally to log errors and message traffic.

Parameters prefix A character string to be prepended to the log messages.

GCOM, INC. DLPI_CONNECT_REQ()

dlpi_decode_disconnect_reason()

 ${\it Prototype} \quad \hbox{ char *dlpi_decode_disconnect_reason (int } \quad \hbox{dlpi_data,}$

dl_uns32 pf,
char *datap,
int length);

Include File(s) **<gcom/dlpiapi.h>**

Description This routine returns a pointer to a string describing the reason code

reason.

Parameters reason The disconnect reason code to decode.

dlpi_detach_ppa()

Prototype int dlpi_detach_ppa ;

Include File(s) **<gcom/dlpiapi.h>**

Description This routine sends a DL_DETACH_REQ primitive to the DLPI

provider and waits for a DL_OK_ACK response.

Parameters dlpi_data This is the file descriptor returned by an open call on the

/dev/dlpi_clone file. Typically, your application previously used DLPI API procedures to attach a PPA, bind a DLSAP and listen for or initiate a connection.

However, these steps are not required.

Return < 0 See for a complete list of the possible error return values.

>= 0 The stream is now detached.

GCOM, INC. DLPI_DETACH_PPA()

dlpi_discon_req()

Include File(s) <gcom/dlpiapi.h>, <gcom/dlpi.h>, <gcom/dlpi_ext.h>

Description This routine is a deprecated form of dlpi_disconnect_req(). See

dlpi_disconnect_req() on page 142 for additional details.

Parameters dlpi_data The streams file identifier. This is the file descriptor

returned by a call to *dlpi_open()*.

reason The disconnect reason code.

Return < 0 See for a complete list of the possible error return values.

>= 0 The stream is now detached.

dlpi_disconnect_req()

Include File(s) <gcom/dlpiapi.h>, <gcom/dlpi.h>, <gcom/dlpi_ext.h>

Description This routine sends a disconnect request primitive

(DL_DISCONNECT_REQ) to DLPI and waits for a DL_OK_ACK response. Any XIDs, TESTs, unnumbered information U-frames and

data messages are discarded while disconnecting.

Parameters dlpi_data This is the file descriptor returned by an open call on

the /dev/dlpi_clone file. Typically, your application previously used DLPI API procedures to attach a PPA, bind a DLSAP and listen for or initiate a connection.

However, these steps are not required.

reason The code to go into the dl_reason field of the

DL_DISCONNECT_REQ. Valid codes include the

following:

DL_DISC_NORMAL_CONDITION. Normal release

of a data link connection.

DL_DISC_ABNORMAL_CONDITION. Abnormal

release of a data link connection.

DL DISC PERMANENT CONDITION. A

permanent condition caused the rejection of a connect

request.

DL_DISC_TRANSIENT_CONDITION. A transient (momentary) condition caused the rejection of a

connect request.

DL DISC UNSPECIFIED. Reason unspecified.

Return < 0 A failure to read/write on the stream occurred. See for

a complete list of the possible error return values.

>= 0 Disconnect successful.

GCOM, INC. DLPI_GET_A_MSG()

dlpi_get_a_msg()

Include File(s) <gcom/dlpiapi.h>, <gcom/dlpi.h>, <gcom/dlpi_ext.h>

Description

This routine issues a *getmsg* system call on the indicated stream. The M_PROTO portion of the received message is read into the global array *dlpi_ctl_buf* and *dlpi_ctl_cnt* gives its length, negative if no M_PROTO portion received. The data portion is read into your buffer. The number of data bytes read is contained in the global *dlpi_data_cnt* upon return from the routine, negative if no data portion was read.

If the received data does not fit in your application's buffer, the UNIX getmsg() MOREDATA bit is returned. If the received control (M_PROTO) does not fit in the API's global $dlpi_ctl_buf$, MORECTL is returned. However, the MORECTL should not actually occur, because the API $dlpi_ctl_buf$ is large. See the UNIX manual page concerning getmsg for further information.

Parameters	dlpi data	The stream from which to get a message.	
1 arameters	aipi aaia	The stream from which to get a message.	

buf The buffer into which to receive the data portion of the message.

The maximum amount of data to receive.

Return < 0 An error occurred (message optionally written to log file).

See for a complete list of the possible error return values.

== 0 Success.

cnt

> 0 MOREDATA and/or MORECTL bit set as described

previously.

dlpi_get_both()

Prototype: int dlpi_get_both(int strm,

char **ctrl_ptr,
char **data_ptr);

Include Files: <gcom/dlpiapi.h>, <gcom/dlpi_ext.h>

Description: Much like the dlpi_get_a_msg(strm, data_buf, data_buf_size) with a

different interface. Call the getmsg streams routine. Any data is read

into the dlpi_data_buf, while any control bytes are read into

dlpi_ctl_buf. The caller does not need to be aware of the names of the

buffers, as the buffer pointers are returned. The global variable *dlpi_data_cnt* contains the number of data bytes read, while the global

variable *dlpi_ctl_cnt* contains the number of control bytes.

Parameters: strm The stream from which to get data and control bytes.

ctrl_ptr Points to the dlpi_ctl_buf char pointer

data_ptr Points to the dlpi_data_buf char pointer

Return: < 0 An error occurred (message optionally written to log file).

See "DLPI API Library Routine Error Return Defines" on page 118 for a complete list of the possible error return

values.

== 0 Success.

> 0 MOREDATA and/or MORECTL bit set as described

previously.

$dlpi_get_info_strm()$

Prototype	<pre>int dlpi_get_info_strm (int dlpi_data, char *ptr) ;</pre>		
Include File(s)	<gcom dlpiapi.h=""></gcom>		
Description	waits for the	e requests default information from the DLPI provider and e response. The information is obtained pertaining to the esented by the file descriptor that is passed in as a parameter.	
	Upon successful return, the user's buffer contains a <i>dl_info_ack</i> structure containing the information.		
Parameters	dlpi_data	Open stream file descriptor to the DLPI Provider.	
	*ptr	Application's buffer.	
Return	< 0	See "DLPI API Library Routine Error Return Defines" on page 118 for a complete list of the possible error return values.	
	>0	Information returned. This is the size of the message pointed to by *ptr.	

dlpi_get_style_strm()

Prototype int dlpi_get_style_strm(int dlpi_data) ;

Include File(s) **<gcom/dlpiapi.h>**, **<gcom/dlpi.h>**

Description This routine returns the style used to configure the DLPI provider. The

inquiry is made using the stream represented by the file descriptor

parameter.

Return DLPIAPI_UNUSABLE

Clone open failed or unable to get style from provider (probably because provider was not initialized by the

dlpi_init() routine).

> 0 The provider style is returned. The two legal values are as

follows:

DL_STYLE1 This indicates that the Physical Point of

Attachment (PPA) is implicitly bound at

open time.

Note: *In practice, this style almost never occurs.*

DL_STYLE2 This indicates that the PPA must be explicitly attached. Refer to *dlpi_attach_req()* for further details.

GCOM, INC. DLPI_GET_STYLE()

dlpi_init()

Prototype int dlpi_init(unsigned log_optns, char *log_name); <gcom/dlpiapi.h> *Include File(s)* Description This routine performs initialization functions for the DLPI API library. It should be called prior to using any other library routine. **Parameters** log_optns Options for controlling messages that the DLPI API library writes to its log file. This parameter consists of a number of single-bit values that are ORed together to form the parameter value. See "DLPI API Logging Options" on page 119 for the options. log_name If a NULL log file name was specified, and if the DLPI_LOG_FILE options was specified, the log file default is as defined in "DLPI API Logging Options" on page 119. For unsuccessful initialization. Return < 0 >0 For successful initialization.

dlpi_init_FILE()

Include File(s) **<gcom/dlpiapi.h>, <stdio.h>**

Description This routine functions the same as the dlpi_init() routine except that it

causes the DLPI API library to used the passed FILE for its log file

rather than opening one of its own.

Parameters log_optns Options for controlling messages that the DLPI API library

writes to its log file. This parameter consists of a number of single-bit values that are ORed together to form the parameter value. See Section 3.3, *Logging Option*, page 8,

for the options.

log_FILE Pointer to a FILE data structure.

Return < 0 For unsuccessful initialization.

> 0 For successful initialization.

GCOM, INC. DLPI_LISTEN()

dlpi_listen()

Description

This routine listens for an incoming connection and returns the file descriptor of a data stream which is open for reading and writing data to DLPI.

The routine *dlpi_listen()* is equivalent to *dlpi_listen_host()*.

The *ppa* parameter specifies either the multiplexor ID or the Lower Point of Attachment (LPA) over which the connection is to be issued. If the number is negative, then the absolute value is assumed to be the LPA, otherwise it is assumed to be the multiplexor ID of the lower stream. If the value 0L is passed, the default of -1L (line 1) is substituted.

The routine issues a DL_ATTACH_REQ to the indicated PPA to attach a DLPI data stream to a line.

The routine issues a DL_BIND_REQ using the routine *dlpi_bind_dlsap()*. Once bound, it configures the addresses as specified by the local and remote DLSAPs. It then waits for an incoming DL_CONNECT_IND message from DLPI. It accepts the connection and returns the file descriptor for the new stream to the caller.

The bind_dlsap is used to bind the stream to an SDLC station. Once bound, the local_dlsap and remote_dlsap values are used to configure the addresses of the station. If the local and remote DLSAPs are zero, then the configured defaults for the station will be used. For an NRM primary or secondary, only one of the local or remote DLSAPs need be non-zero since these modes of operation require only one address. For ABM, both addresses must be supplied (or neither for using defaults).

Parameters ppa The multiplexor ID or LPA to which to attach the DLPI

stream.

bind_dlsap DLSAP to use in the DL_BIND_REQ.

local_dlsap The DLSAP to use as the local address.

remote_dlsap The DLSAP of the remote system.

fork_options Reserved for future use. Must be set to 0.

GCOM, INC. DLPI_LISTEN()

Return < 0

Any kind of error that occurred in the binding or connection process. See "DLPI API Library Routine Error Return Defines" on page 118 for a complete list of the possible error return values.

> 0 Successful return. The return value is the UNIX file descriptor for the stream that has the open connection assigned to it. This descriptor is suitable for passing to any of the DLPI API routines that have a stream parameter, or the UNIX read(), write(), getmsg(), putmsg() family of routines.

dlpi_open()

Prototype New: int dlpi_open(char *hostname)

Old: int dlpi_open_data(void);

Include File(s) **<gcom/dlpiapi.h>**

Description This routine opens a stream to the DLPI driver. It uses the clone open

facility to do so. This routine is used internally by the DLPI API library to open files to the DLPI driver. It is ordinarily not called by the user but is provided to allow you to perform low level functions with the DLPI

driver.

The dlpi_open_data(void) routine is equivalent to dlpi_open(NULL).

Return < 0 An error occurred indicating that the system was not

initialized or not opened. See "DLPI API Library Routine Error Return Defines" on page 118 for a complete list of

the possible error return values.

>= 0 File descriptor handle (identifier) for the open file.

Parameters hostname A pointer to an ASCII string name of the machine on

which the operation is to be performed. If the pointer is NULL, or points to an empty string, the operation is performed on the machine on which the call to the API

library is made (i.e., the local host).

GCOM, INC.

DLPI_LISTEN()

$dlpi_open_log()$

Prototype void dlpi_open_log(void) ;

Include File(s) **<gcom/dlpiapi.h>**

Description This routine is used internally by the API to open the log files. The

application ordinarily should not need to invoke it directly.

dlpi_perror()

Prototype void dlpi_perror(char *prefix) ;

Include File(s) <gcom/dlpiapi.h>

Description This routine mimics the standard C library's perror() routine. The

dlpi_perror() routine differs in that dlpi_perror() prints to the DLPI

logfile instead of standard error.

Parameters prefix The string pointed to by this argument will be printed, then

a colon and a space, then a message describing the most

recent error, then a newline.

GCOM, INC. DLPI_LISTEN()

dlpi_print_msg()

Prototype void dlpi_print_msg(unsigned char *msg,

unsigned length,

int indent) ;

Include File(s) **<gcom/dlpiapi.h>**

Description This routine translates a binary message into an ASCII hex dump. This

can be useful when an application wishes to log message content.

Parameters msg Pointer to the message to be printed.

length The number of bytes to print from the message.

indent The number of spaces to indent the printout.

dlpi_printf()

Prototype void dlpi_printf(char *fmt, ...);

Include File(s)

<gcom/dlpiapi.h>

Description

This routine performs the same function as the UNIX *printf()* routine. The logging options specified in the *dlpi_init()* call determine whether the output is written to the log file, *stderr*, both or neither. This is the routine that the DLPI API library uses internally to write messages to the log.

The format of the output produced by $dlpi_printf()$ consists of the process ID of the process that called the function, a time stamp and the message formatted according to the arguments passed to $dlpi_printf()$. For example:

```
dlpi_printf("Hello world!\n") ;
```

would produce output similar to the following:

```
609 08:14:33 Hello world
```

Parameters

fmt

Format string compatible with UNIX printf function

... Additional arguments to print (optional)

GCOM, INC. DLPI_PRINTF()

dlpi_put_both()

```
Prototype
                  int dlpi_put_proto(int
                                                dlpi_data,
                                         char *hdr_ptr,
                                                hdr_lgth,
                                         int
                                         char *data ptr,
                                         int
                                                data_lgth ) ;
Include File(s)
                  <gcom/dlpiapi.h>
  Description
                  This routine sends a message with both control and data parts.
                  Generally, this routine will only be used internally, but it is presented
                  here in the event that an application may need this functionality.
   Parameters
                  dlpi_data
                               stream to which to write protocol message.
                  hdr ptr
                               Pointer to the control part of the message. May be null if
                               hdr_lgth is \leq 0.
                               Length of the data pointed to by hdr_ptr.
                  hdr_lgth
                               Pointer to the data part of the message. May be null if
                  data_ptr
                               data_lgth is \leq 0.
                               Length of the data pointed to by data_ptr.
                  data_lgth
       Return
                  < 0
                               See "DLPI API Library Routine Error Return Defines" on
                               page 118 for a complete list of the possible error return
                               values.
                  >=0
                               Return value from the UNIX putmsg call.
```

dlpi_put_proto()

Include File(s) **<gcom/dlpiapi.h>**

Description This routine writes a prototype message to the DLPI driver on the given

stream. The prototype message is assumed to have been built in the global array *dlpi_ctl_buf*. The system call to write the protocol message is retried in the event that the call was interrupted by a UNIX signal.

Parameters dlpi_data stream to which to write protocol message.

lgth Number of bytes to write.

Return < 0 See for a complete list of the possible error return values.

> 0 Return value from the UNIX *putmsg* call.

GCOM, INC. DLPI_RCV()

dlpi_rcv()

Include File(s)

<gcom/dlpiapi.h>, <gcom/dlpi.h>, <gcom/dlpi_ext.h>

Description

This routine receives data generated from connectionless or connectionoriented environments and processes DLPI control messages from the stream. It is designed to support applications that need to do more than simply set up a connection and exchange data with a peer until the connection is terminated. Your application can control the behavior of $dlpi_rcv()$ by setting input flags.

The *dlpi_rcv()* routine supports the following features:

- Responses to management requests from your application are processed (such as DL_ATTACH_REQ, DL_BIND_REQ and DL_INFO_REQ)
- Transparent connection maintenance after the connection is established
- Application buffers smaller that incoming frames by interpreting the getmsg MOREDATA bit
- · Connectionless and connection-oriented data transfer
- U-frame exchanges while connected

Received data are read into your application's buffer with the length of the received data specified in the global integer *dlpi_data_cnt*. Any received control information is read into the *dlpi_ctl_buf* global, with its length specified in global integer *dlpi_ctl_cnt*.

Receiving U-Frames

U-frames (XIDs, TESTs and unnumbered information U-frames) can be exchanged once the stream has been attached and bound or when a connection has been established. Five return codes are used to signal the delivery of U-frames: DLPIAPI_XID_IND, DLPIAPI_XID_CON, DLPIAPI_TEST_IND, DLPIAPI_TEST_CON and DLPIAPI UIC IND. (These are described later.)

To receive U-frames, the corresponding input *flags* bit must be set: ReturnXID, ReturnTEST or ReturnUIC. If the corresponding *flags* bit is not set, an attempt is made to deliver the U-frame using a U-frame handler. Your application may set up a U-frame handler using

dlpi_set_unnum_frame_handler(). If a U-frame handler has not been specified, the U-frame is silently discarded.

Your application buffer might not be large enough to hold the U-frame. If so, the U-frame is delivered in fragments, one per call on $dlpi_rcv()$ to your application.

First fragment in U-Frame

In the first U-frame fragment, DLPIAPI_MOREDATA is set in *out_code*. The return value corresponds to the type of U-frame received (XID, TEST or unnumbered information U-frame).

Middle U-Frame Fragments

In fragments that occur after the first and before the last fragment, DLPIAPI_MOREDATA is set in *out_code* and the return value is set to DLPIAPI_DATA.

Last Fragment in U-Frame

In the last fragment in a segmented U-frame, DLPIAPI_MOREDATA is not set in *out_code* and the return value is set to DLPIAPI_DATA.

Non-Blocking I/O

Non-blocking I/O is partially supported. That is, non-blocking I/O is supported in the sense that if *getmsg* returns -1 and if *errno* equals EAGAIN, DLPIAPI_EAGAIN is returned. Non-blocking I/O is enabled by setting the O_NDELAY flag, which called the O_NONBLOCK flag for POSIX compatibility (see the *fcntl()* UNIX system call). Your application must recover from any write calls (to the *putmsg* command) that defer because non-blocking I/O is set on the stream.

If the AutoConnect, AutoReset, or AcceptConnect *flags* bit is set, the API generates DLPI protocol messages in response to specific DLPI protocol messages. If your application has specified non-blocking I/O on the data stream, then an API attempt to send such a message may fail. The API does not have access to a retry mechanism. That is, if your applications using non-blocking I/O, these flags should not be used. Your application may use *dlpi_send_reset_res()*, *dlpi_send_connect_res()*, and *dlpi_send_connect_req()* in response to connection-oriented events.

GCOM, INC. DLPI_RCV()

Parameters dlpi_data

stream file ID. This is the file descriptor returned by an open call on the /dev/dlpi_clone file. Typically, your application previously used DLPI API procedures to attach a PPA, bind a DLSAP and listen for or initiate a connection. However, these steps are not required.

*data_ptr

Pointer to your application's buffer. Connection-oriented data and U-frames are returned to your application using this pointer. The *data_ptr* parameter must not be NULL. The length of the actual data placed in your application's buffer is in the integer global *dlpi_data_cnt*.

bfr_len

A positive integer specifying the length of the application's buffer.

out_code

A pointer to an application variable used to returned additional information. The pointer must *not* be null. The semantics of the value returned via this pointer depends on the value returned by the function, as follows:

If the DLPIAPI_DATA return code is set and *out_code* is set to DLPIAPI_MOREDATA, this indicates a fragment (more data). Other bits are reserved for future use. The last fragment in a frame is indicated by DLPIAPI MOREDATA being reset.

DLPIAPI_POLL_FINAL

This is set when delivering DLPIAPI_XID_IND, DLPIAPI_XID_CON, DLPIAPI_TEST_IND, DLPIAPI_TEST_CON or DLPIAPI_UIC_IND if the poll/final bit is set in the received message.

DLPIAPI_OK_ACK.

The acknowledged DLPI primitive

DLPIAPI_ERROR_ACK.

The rejected DLPI primitive

DLPIAPI_RESET_INDICATION.

Reason code

DLPIAPI_DISC_IND.

Reason code

DLPIAPI_OTHER.

Unsupported DLPI primitive

See **dlpi.h** for DLPI primitive definitions and disconnect and reset indication reason codes.

flags

Input flags used to control the operation of $dlpi_rcv()$ by indicating how it should react to various DLPI protocol messages. The flags specify how received U-frames are to be handled (or discarded) and how $dlpi_rcv()$ should react to connect, disconnect and reset messages.

Input Flags in Detail

Each valid *flags* value is explained in more detail as follows:

AutoReset

Requests $dlpi_rcv()$ to respond to a DLPI reset indication with a reset response. The $dlpi_rcv()$ routine then returns DLPIAPI_RESET_INDICATION. Your application and its peer cannot yet exchange data. After your application has completed its response to the reset, it should call $dlpi_rcv()$ back and wait for the reset sequence to complete. The reset complete is indicated when DLPIAPI_RESET_COMPLETE is returned.

If the AutoReset *flags* bit is not set, DLPIAPI_RESET_INDICATION is returned without further action taken.



Note: If your application has specified non-blocking I/O on the data stream (see the UNIX fcntl() system call), this flag should not be used.

AutoConnect

Requests $dlpi_rcv()$ to respond to a DLPI disconnect indication with a connect request. The $dlpi_rcv()$ routine has no way to determine the destination DLSAP used when the connection was originally setup. Therefore, a DLSAP of "0L" is used. When the connection completes,

DLPIAPI_CONNECT_COMPLETE is returned to your application.



Note: If your application has specified non-blocking I/O on the data stream (see the UNIX fcntl() system call), this flag should not be used.

GCOM, INC. DLPI_RCV()

Input Flags in Detail (continued) AutoListen

Requests *dlpi_rcv()* to respond to a DLPI disconnect indication by waiting for an incoming connect indication. When the connect indication arrives, a connect response is sent to the peer and DLPIAPI_CONNECT_IND is returned to your application.

After your application has completed its response to the DLPIAPI_CONNECT_IND event, your application should call *dlpi_rcv()* to wait for the connection setup to be completed. This is reported to your application by returning DLPIAPI CONNECT COMPLETE.

If AutoListen and AcceptConnect are reset when a DLPI disconnect indication is received, DLPIAPI DISC IND is returned to your application without further action.

AcceptConnect The semantics of this flag are similar to those of the AutoListen flag.

> This flag is useful to listen for an incoming connection. If *dlpi_rcv()* receives a DLPI connect indication and AcceptConnect or AutoListen is set, a connect response is returned to the peer and DLPIAPI_CONNECT_IND is returned to your application.

After your application has completed its response to the DLPIAPI_CONNECT_IND event, your application should call *dlpi_rcv()* to wait for the connection setup to be completed. This is reported to your application by returning DLPIAPI_CONNECT_COMPLETE.

If AutoListen and AcceptConnect are reset when a DLPI connected indication is received, DLPIAPI_CONNECT_IND is returned to your application without further action.



Note: If your application has specified non-blocking I/O on the data stream (see the UNIX fcntl() system call), this flag should not be used.

ReturnInfo

If this flag is set, DLPIAPI_INFO_ACK is returned to your application in response to a DLPI information acknowledge (DL_INFO_ACK). Otherwise the DL_INFO_ACK is silently discarded.

Input Flags in Detail (continued)

RetryOnSignal If RetryOnSignal is set, *dlpi_rcv()* retries the *getmsg* call when a system call is interrupted. Otherwise, DLPIAPI_EINTR is returned.

> If a putmsg is interrupted with a signal, the putmsg is retried independent of the setting of the RetryOnSignal flag.

> UNIX signals may interrupt most system calls, including getmsg (see the EINTR signal defined in dlpiapi.h). If your application is using signals, your application may need to control how *dlpi_rcv()* reacts when *getmsg* is interrupted with a signal.

ReturnXID ReturnUIC ReturnTEST

For your application to receive a U-frame event code (XID, TEST or unnumbered information U-frame), the corresponding *flags* bit must be set.

GCOM, INC. DLPI_RCV()

Return Values

The return value is an event code corresponding to the message type read from the stream. The set of return codes that your application may encounter depends on the *flags* bits that are set and how your application sets up a DLPI connection, if any. For example, if *dlpi_connect()* is used to set up a connection, the DLPI connection primitive are not processed here. In addition, U-frames are silently discarded if the unnumbered input flags are reset.

In general, the return code values are grouped into three categories:

- negative values indicate an error
- zero indicates data (normal data or U-frame fragment)
- positive values indicate a DLPI protocol message.

Negative error codes

See for a complete list of the possible error return values.

Zero data code

DLPIAPI_DATA

Data has been returned. The *dlpi_data_cnt* global contains the length of the data read into your application's buffer.

If your application buffer is not large enough to hold the received frame, DLPIAPI_MOREDATA is returned through *out_code*. That is, the frame is returned in fragments to your application. Your application can detect the last fragment of the frame when DLPIAPI_MOREDATA is reset to zero.



Note: The U-frames (TESTs, XIDs and unnumbered information U-frames) described later may also be received in fragments when your application buffer is too small. For the U-frames, the first fragment in the frame is returned with the corresponding event code, while the remaining fragments are returned with the DLPIAPI_DATA event code.

DLPIAPI_DATA_ACK

Reserved for Future Use.

Return Values (continued)

Positive protocol message event codes

DLPIAPI_ATTACHED DLPIAPI_DETACHED DLPIAPI_BOUND DLPIAPI_UNBOUND

These four event codes are received by your application only when it is explicitly attaching/binding or detaching/unbinding. The DLPI API primitives used for these functions wait for the response from the DLPI Provider. For further information, refer to $dlpi_attach_ppa()$, $dpi_send_attach_req()$, $dlpi_send_unbind_req()$, $dlpi_send_dlsap()$, $dlpi_send_bind_req()$ and $dlpi_send_detach_req()$.

DLPIAPI_DISC_IND

If the AutoConnect *flags* bit is set when a DL_DISCONNECT_IND protocol message is received, a connect request is returned to the DLPI provider, and *dlpi_rcv()* reads again, waiting for a response to the connect request.



Note: The API sleeps for one second after sending the connect request to prevent extended buzz loops.

If the AutoListen *flags* bit is set instead when a DL_DISCONNECT_IND protocol message is received, *dlpi_rcv()* reads again waiting for a connect indication or U-frames.

If AutoConnect and AutoListen are not set, the DLPIAPI_DISC_IND is returned without further action.

DLPIAPI_DISC_COMPLETED

Your application previously sent a disconnect request and the DLPI Provider's response has been received.

DLPIAPI_RESET_INDICATION

Indicates that a DL_RESET_IND protocol message was received. Data may have been lost. If the AutoReset *flags* bit was set, a reset response was returned to the DLPI Provider. Your application should call *dlpi_rcv()* again to wait for the reset sequence to complete.

GCOM, INC. DLPI_RCV()

Return Values (continued)

DLPIAPI_RESET_COMPLETE

A reset sequence has completed and your application may resume transmitting data. The reset sequence may have been initiated by your application with a reset request, or by the peer with a reset indication.

DLPIAPI_CONNECT_IND

The peer is requesting a connection. If the AcceptConnect *flags* bit was set, *dlpi_rcv()* has returned a connect response. Your application should call *dlpi_rcv()* to wait for connection setup to complete.

DLPIAPI_CONNECT_COMPLETE

A connection setup sequence has completed and your application can now exchange data with it's peer. The connection setup may have been initiated by your application with a connection request or by the peer with a connection indication.

DLPIAPI ERROR ACK

The DLPI Provider rejected a request. The DLPI request code is returned in *out_code*.

DLPIAPI OK ACK

This return value should not occur. The DLPI Provider is reporting the acceptance of a request.

DL_OK_ACKs to attach, bind, connect, reset and disconnect requests should be reported through other return values.

The accepted request code is returned through *out_code*.

DLPIAPI_INFO_ACK

Contains a response to a previously issued information request. The response is in the global *dlpi_ctl_buf*.

DLPIAPI_TOKEN_ACK

Reserved for future use.

DLPIAPI_PHYS_ADDR_ACK

Reserved for future use.

DLPIAPI STATISTICS ACK

Reserved for future use.

Return Values (continued)

DLPIAPI_OTHER

A DLPI primitive unsupported by $dlpi_rcv()$ was read. The DLPI primitive code is returned through out_code . Your application can examine the received message through the $dlpi_ctl_buf$ global or it can ignore the message entirely.

DLPIAPI_XID_IND DLPIAPI_XID_CON DLPIAPI_TEST_IND DLPIAPI_TEST_CON DLPIAPI_UIC_IND

These five event codes are used to deliver U-frames to your application. Your application must explicitly request U-frames or else they are silently discarded.

For your application to receive a U-frame event code, the corresponding *flags* bit (ReturnXID, ReturnTEST, ReturnUIC) must be set.

If the appropriate *flags* bit is not set, an attempt is made to deliver the U-frame using the unnumbered handler mechanism. See *dlpi_set_unnum_frame_handler()* for details. If your application has not requested this service, the U-frame is silently discarded.

DLPIAPI_UIC_ERROR_IND

Reserved for Future Use.

GCOM, INC. DLPI_RCV()

dlpi_rcv_msg()

Include File(s) <**gcom/dlpiapi.h>**

Description This routine

Parameters dlpi_data stream from which to read data

data_ptr Buffer to which to read the data

bfr_len Maximum number of bytes to read

flags Flags controlling the behavior of dlpi_rcv_msg(). A bitwise OR of the following options:

AutoReset: If a reset indication is received, this procedure will complete the reset sequence, then will return a DL_OK_ACK for a DL_RESET_CON when completed. Failure to complete the reset sequence will be indicated with a DL_ERROR_ACK for a DL_RESET_REQ.

RetryOnSignal: If the getmsg() system call is interrupted with a signal and this flag is set, the getmsg() will be tried again.

DISC_CONNECT: If specified, respond to a DL_DISCONNECT_IND by initiating a connect request. The *peer_sap* specified in the most recent *dlpi_send_connect_req()* or *dlpi_connect_req()* is used. Note that only one *peer_sap* is saved per process. A *peer_sap* is NOT saved for each connected stream. Failure will be indicated with a DL_ERROR_ACK of a connect primitive.

DISC_LISTEN: If specified, respond to a DL_DISCONNECT_IND by listening for an incoming connect indication. A successful connection will be indicated with a DL_OK_ACK of a DL_CONNECT_CON. Failure will be indicated with a DL_ERROR_ACK of a connect primitive.

ReturnInfo: If specified, DL_INFO_ACK, DL_TOKEN_ACK, DL_PHYS_ADDR_ACK, and DL_GET_STATISTICS_ACK messages are returned to the user. Otherwise, they are discarded. These messages are generated by the DLPI Provider in response to their corresponding requests.

ReturnUnnum: If set, XIDs, TESTs, and UIs are returned to the user. If not set, then the un-numbered frame handler mechanism is used to optionally deliver these frames to the user.

Return < 0 An error occurred (message written to log file)
 DLPI control message is present in dlpi_ctl_buf, length dlpi_ctl_cnt. Any associated data is in the user's buffer, length in the global dlpi_data_cnt.
 Number of bytes read into user buffer

GCOM, INC. DLPI_READ_DATA()

dlpi_read_data()

Include File(s)

<gcom/dlpiapi.h>

Description

This routine reads a single data message from the DLPI driver on the indicated stream. Up to *cnt* bytes of data can be read into the buffer area pointed to by *buf*. Connection-oriented data are returned directly by this routine. If connectionless data are being returned, *dlpi_read_data()* makes an attempt to call the *dlpi_set_unnum_frame_handler()* routine, whose code must be modified to fit the circumstances. The *dlpi_set_unnum_frame_handler()* routine is described on page 193.

This routine does not expect to read a protocol message from the stream. It is considered an error to receive a protocol message. Similarly, if the return from *getmsg* indicates that no data message has been read, an error is returned. Under these circumstances, messages are written to the log file.



Note: To read data and process protocol messages, use the dlpi_rcv() procedure. For connectionless data transfer environments, using dlpi_read_data() is usually sufficient.

The ordinary case is that $dlpi_read_data()$ will receive a message with no protocol portion and valid data. In such a case it simply returns the length of the data.

Parameters dlpi_data stream from which to read data

buf Buffer to which to read the data

cnt Maximum number of bytes to read

Return < 0 An error occurred (message written to log file)

> 0 Number of bytes read into user buffer

dlpi_reset_req()

Prototype	<pre>int dlpi_reset_req (int dlpi_data) ;</pre>		
Include File(s)	<gcom dlpiapi.h=""></gcom>		
Description	This routine sends a DL_RESET_REQ primitive and waits for the corresponding DL_RESET_CON.		
Parameters	dlpi_data	stream file ID. This is the file descriptor returned by an open call on the /dev/dlpi_clone file.	
Return	< 0	See for a complete list of the possible error return values.	
	== 0	A disconnect indication was received. Connection is now in the DL_IDLE state.	
	>0	Reset sequence completed successfully. Connection is now in the data transfer state.	

GCOM, INC. DLPI_RESET_RES()

dlpi_reset_res()

Prototype int dlpi_reset_res(int dlpi_data) ;

Include File(s) **<gcom/dlpiapi.h>**

Description This routine sends a reset response to DLPI and awaits the receipt of a

DL_OK_ACK from DLPI. It is appropriate to call this routine after

receiving a DL_RESET_IND on a data stream.

If you are using $dlpi_read_data()$ and if it returns a specific result, you may want to check the $dlpi_ctl_buf$ to see if it contains a

DL_RESET_IND. If so, and if you wish to keep your connection going, you may then call *dlpi_reset_res()* to acknowledge the reset and put the

connection back in the data state.

1

Caution: The receipt of a DL_RESET_IND means that data messages may have been lost.

Parameters dlpi_data stream to which to send the DL_RESET_RES.

Return < 0 See for a complete list of the possible error return values.

== 0 Connection disconnected.

> 0 Reset sequence completed successfully. Connection is

now in the data transfer state.

dlpi_send_attach_req()

Prototype int dlpi_send_attach_req (int dlpi_data, unsigned long ppa) ; <gcom/dlpiapi.h> *Include File(s)* This routine sends an attach request to the DLPI provider. Your Description application is responsible for reading the response. For example, you might want to use *dlpi_complete_req()* to read the response. This request is legal only in the DL_UNATTACHED state. **Parameters** stream file ID. This is the file descriptor returned by an dlpi_data open call on the /dev/dlpi_clone file. If this is negative, it is an LPA. Otherwise, it is a streams ppa multiplexor ID. Return < 0 See for a complete list of the possible error return values. >=0Successfully sent.

GCOM, INC. DLPI_SEND_BIND_REQ()

dlpi_send_bind_req()

Prototype int dlpi send bind req (int dlpi data, unsigned long dlsap, int conind nr, int service_mode, int conn mgmt, int auto_flags) ; *Include File(s)* <gcom/dlpiapi.h>, <gcom/dlpi.h>, <gcom/dlpi_ext.h> This routine sends a DL BIND REQ primitive to the DLPI provider. **Description** Your application is responsible for reading the response. This request is legal only in the DL_UNBOUND state. **Parameters** stream file ID. This is the file descriptor returned by an dlpi_data open call on the /dev/dlpi_clone file. DLSAP address to be bound to the stream. dlsap conind_nr Connection indication value. Zero implies connect and one implies listen. service mode The service mode, which can be DL_CODLS (indicating connection-oriented service) or DL_CLDLS (indicating connectionless service). This parameter is unsupported at this time and should conn mgnt be set to zero. A non-zero value would indicate connection management. This parameter is ignored. auto_flags Return < 0See for a complete list of the possible error return values. >0Success. This is set to the size of the successful bind request.

$dlpi_send_connect_req()$

Prototype	int dlpi_s	send_connect_req(int dlpi_data, unsigned long peer_sap);	
Include File(s)	<gcom dlpiapi.h=""></gcom>		
Description	This routine formats and sends a DLPI connect request in a connection-oriented environment.		
Parameters	dlpi_data	stream file ID. This is the file descriptor returned by an open call on the /dev/dlpi_clone file.	
	peer_sap	Address of the peer to which you wish to be connected.	
Return	< 0	See "DLPI API Library Routine Error Return Defines" on page 118 for a complete list of the possible error return values.	
	>= 0	Success.	

GCOM, INC. DLPI_SEND_CONNECT_RES()

dlpi_send_connect_res()

```
Prototype
                 int dlpi_send_connect_res(int
                                                        dlpi data,
                                              ulong correlation,
                                               ulong dlpi_token );
Include File(s)
                  <gcom/dlpiapi.h>
  Description
                  This routine formats and sends a connect response in a connection-
                  oriented environment. It does not wait for a response. DLPI must be
                  initialized for this to work.
   Parameters
                  dlpi_data
                              stream file ID. This is the file descriptor returned by an
                              open call on the /dev/dlpi_clone file.
                  correlation Typically, this is the correlation value contained in a
                              dl_connect_ind_t structure.
                  dlpi_token
                              This parameter must be set to zero.
       Return
                  < 0
                              See "DLPI API Library Routine Error Return Defines" on
                              page 118 for a complete list of the possible error return
                              values. Perhaps DLPI was not initialized.
                  >=0
                              Success.
```

dlpi_send_detach_req()

Prototype int dlpi_send_detach_req(int dlpi_data) ;

Include File(s) **<gcom/dlpiapi.h>**

Description This optional routine sends a detach request to DLPI. It can be used in

both connectionless and connection-oriented environments. However, $dlpi_send_detach_req()$ is usually not called, because a simple $dlpi_$

close command is sufficient in place of the more formal

unbind/detach/close process. Refer to your UNIX system administration

manual for details about the close command.

Parameters dlpi_data stream file ID. This is the file descriptor returned by an

open call on the /dev/dlpi_clone file.

Return < 0 See "DLPI API Library Routine Error Return Defines" on

page 118 for a complete list of the possible error return

values.

>= 0 Success.

dlpi_send_disconnect_req()

Include File(s) <gcom/dlpiapi.h>, <gcom/npi.h>, <gcom/npi_ext.h>

Description This routine formats and sends a disconnect request primitive

(DL_DISCONNECT_REQ) to DLPI. Any XIDs, TESTs, unnumbered

information U-frames and data messages are discarded while disconnecting. Your application is responsible for reading the response.

Parameters dlpi data stream file ID over which to send the

DL_DISCONNECT_REQ. This is the file descriptor returned by an open call on the /dev/dlpi_clone file. Typically, your application previously used DLPI API procedures to attach a PPA, bind a DLSAP and listen for or initiate a connection. However, these steps are not

required.

reason The code to go into the dl_reason field of the

DL_DISCONNECT_REQ. Valid codes include the

following:

DL_DISC_NORMAL_CONDITION. Normal release of a

data link connection.

DL_DISC_ABNORMAL_CONDITION. Abnormal

release of a data link connection.

DL_DISC_PERMANENT_CONDITION. A permanent condition caused the rejection of a connect request.

DL_DISC_TRANSIENT_CONDITION. A transient (momentary) condition caused the rejection of a connect

request.

DL DISC UNSPECIFIED. Reason unspecified.

Return < 0 See "DLPI API Library Routine Error Return Defines" on

page 118 for a complete list of the possible error return

values.

>= 0 Success.

$dlpi_send_info_req()$

Include File(s) <gcom dlpiapi.h=""> Description This routine formats and sends a DL_INFO_REQ primitive. Your application is responsible for reading the response. Parameters dlpi_data stream file ID. This is the file descriptor returned by an open call on the /dev/dlpi_clone file. Return <0 See "DLPI API Library Routine Error Return Defines" on page 118 for a complete list of the possible error return values. >= 0 Request successfully sent.</gcom>	Prototype	<pre>int dlpi_send_info_req (int dlpi_data) ;</pre>		
application is responsible for reading the response. *Parameters dlpi_data stream file ID. This is the file descriptor returned by an open call on the /dev/dlpi_clone file. *Return < 0 See "DLPI API Library Routine Error Return Defines" on page 118 for a complete list of the possible error return values.	Include File(s)	<gcom dlpiapi.h=""></gcom>		
open call on the /dev/dlpi_clone file. Return < 0 See "DLPI API Library Routine Error Return Defines' on page 118 for a complete list of the possible error return values.	Description			
on page 118 for a complete list of the possible error return values.	Parameters	dlpi_data	± • • • • • • • • • • • • • • • • • • •	
>= 0 Request successfully sent.	Return	< 0		
		>= 0	Request successfully sent.	

GCOM, INC. DLPI_SEND_RESET_REQ()

dlpi_send_reset_req()

Prototype int dlpi_send_reset_req (int dlpi_data) ; <gcom/dlpiapi.h> *Include File(s)* Description This routine formats and sends a DL_RESET_REQ primitive on the specified data stream. Your application is responsible for reading the response. stream file ID. This is the file descriptor returned by an **Parameters** dlpi_data open call on the /dev/dlpi_clone file. Return < 0 See "DLPI API Library Routine Error Return Defines" on page 118 for a complete list of the possible error return values. >=0Success.

dlpi_send_reset_res()

Prototype int dlpi_send_reset_res(int dlpi_data) ;

Include File(s) **<gcom/dlpiapi.h>**

Description This routine formats and sends a DLPI reset response on the specified

connection-oriented data stream. Your application is responsible for

reading the response.

Parameters dlpi_data stream file ID. This is the file descriptor returned by an

open call on the /dev/dlpi_clone file.

Return < 0 See "DLPI API Library Routine Error Return Defines"

on page 118 for a complete list of the possible error

return values.

>= 0 Message sent.

GCOM, INC. DLPI_SEND_RESET_REQ()

dlpi_send_stats_req()

Prototype int dlpi_send_stats_req(int dlpi_data) ; *Include File(s)* <gcom/dlpiapi.h> Description This routine formats and sends a DLPI Get Statistics Request. The caller is responsible for interpreting the response. stream file ID. This is the file descriptor returned by **Parameters** dlpi_data dlpi_open(). Return See "DLPI API Library Routine Error Return Defines" < 0 on page 118 for a complete list of the possible error return values. >= 0Message sent.

dlpi_send_test_req()

Prototype int dlpi_send_test_req (int dlpi data, pfb, ulong char *datap, int length unsigned char *addr ptr, addr_len) ; int *Include File(s)* <gcom/dlpiapi.h> Description This routine sends a TEST request. If *addr_ptr* is not NULL and addr_len is greater than zero, the address in addr_ptr is sent as the destination address. Otherwise, the TEST's destination address is set to NULL. **Parameters** dlpi_data stream file ID. This is the file descriptor returned by an open call on the /dev/dlpi_clone file. Poll/Final bit. pfb datap Pointer to the data for the TEST primitive. length Length of data pointed to by *datap*. Pointer to an address. *addr_ptr addr_len Length of the address pointed to by *addr_ptr*. Return < 0 Failure. See "DLPI API Library Routine Error Return

>= 0 The TEST request was successfully sent.

error return values.

Defines" on page 118 for a complete list of the possible

GCOM, INC. DLPI_SEND_TEST_REQ()

int dlpi_send_test_res (int

dlpi_send_test_res()

Prototype

ulong pfb, char *datap, length int unsigned char *addr ptr, addr_len) ; int <gcom/dlpiapi.h> *Include File(s)* Description This routine sends a TEST response. If addr_ptr is not NULL and addr_len is greater than zero, the address in addr_ptr is sent as the destination address. Otherwise, the TEST's destination address is set to NULL. **Parameters** dlpi_data stream file ID. This is the file descriptor returned by an open call on the /dev/dlpi_clone file. Poll final bit. pfb Pointer to the data for the TEST primitive. datap

length Data length.

*addr_ptr

addr_len Length of the address pointed to by addr_ptr.

Pointer to an address.

Return < 0 Failure. See "DLPI API Library Routine Error Return

Defines" on page 118 for a complete list of the possible

dlpi data,

error return values.

>= 0 Message successfully sent.

dlpi_send_uic()

Include File(s) **<gcom/dlpiapi.h>**

Description

This routine sends the specified data in a unnumbered information U-frame information request (DL_UNITDATA_REQ). It assumes there is a specific address associated with the data. This might occur if your address provided a Media Access (MAC) address, such as in a token ring environment.

If *addr_ptr* is not NULL and *addr_len* is greater than zero, the address in *addr_ptr* is sent as the destination address. Otherwise, the unnumbered information U-frame's destination address is set to NULL.

Parameters dlpi_data stream file ID that is attached and bound. This is the file

descriptor returned by an open call on the /dev/dlpi_clone

file.

*datap Pointer to the data.

data_len Length of the data pointed to by datap.

*addr ptr Pointer to an address.

addr_len Length of the address pointed to by addr_ptr.

Return < 0 Request failed. See "DLPI API Library Routine Error

Return Defines" on page 118 for a complete list of the

possible error return values.

>= 0 The data was successfully sent.

GCOM, INC. DLPI_SEND_UIC()

dlpi_send_unbind_req()

Prototype int dlpi_send_unbind_req(int dlpi_data) ;

Include File(s) **<gcom/dlpiapi.h>**

Description This optional routine sends an unbind request to DLPI. It could be used

in both connectionless and connection-oriented environments. However, $dlpi_send_unbind_req()$ is usually not called, because a simple $dlpi_close$ command is sufficient in place of the more formal unbind/detach/close process. Refer to your UNIX system administration

manual for details about the close command.

Parameters dlpi_data stream file ID. This is the file descriptor returned by an

open call on the /dev/dlpi_clone file.

Return < 0 See "DLPI API Library Routine Error Return Defines"

on page 118 for a complete list of the possible error

return values.

>= 0 Message sent.

dlpi_send_xid_req()

Include File(s) **<gcom/dlpiapi.h>**

Description This routine sends an XID request. If addr_ptr is not NULL and

addr_len is greater than zero, the address in addr_ptr is sent as the destination address. Otherwise, the XID's destination address is set to

NULL.

Parameters dlpi_data stream file ID. This is the file descriptor returned by an

open call on the /dev/dlpi_clone file.

pfb Poll final bit.

datap Pointer to the data for the TEST primitive.

length Length of data pointed to by *datap*.

*addr_ptr Pointer to an address.

addr_len Length of the address pointed to by addr_ptr.

Return < 0 Failure. See "DLPI API Library Routine Error Return

Defines" on page 118 for a complete list of the possible

error return values.

>= 0 The XID request was successfully sent.

GCOM, INC. DLPI_SEND_XID_REQ()

dlpi_send_xid_res()

Include File(s) **<gcom/dlpiapi.h>**

Description This routine sends an XID response. If addr_ptr is not NULL and

addr_len is greater than zero, the address in addr_ptr is sent as the destination address. Otherwise, the XID's destination address is set to

NULL.

Parameters dlpi_data stream file ID. This is the file descriptor returned by an

open call on the /dev/dlpi_clone file.

pfb Poll/Final bit.

*datap Pointer to the data for the TEST primitive.

length Data length.

*addr_ptr Pointer to an address.

addr_len Length of the address pointed to by addr_ptr.

Return < 0 Failure. See "DLPI API Library Routine Error Return

Defines" on page 118 for a complete list of the possible

error return values.

>= 0 Message successfully sent.

dlpi_set_log_size()

Prototype int dlpi_set_log_size (long log_size) ;

Include File(s) **<gcom/dlpiapi.h>**

Description This routine sets the maximum length of the DLPI log file. When the

logfile reaches this length, it will wrap and begin writing new data at the beginning of the file. The newest data will be followed by a line marking

the end of the log.

Parameters log_size Maximum length, in bytes, for the log file.

Return < 0 Failure. See "DLPI API Library Routine Error Return

Defines" on page 118 for a complete list of the possible

error return values.

>= 0 Message successfully sent.

dlpi_set_signal_handling()

Include File(s)

<gcom/dlpiapi.h>, <signum.h>, <gcom/dlpi.h>, <gcom/dlpi_ext.h>

Description

This routine requests that the specified signal be generated by the DLPI provider when the DLPI provider sends any of the messages indicated by *primitive_mask* up stream.

After the DLPI provider generates the signal, the signal is caught by a DLPI API signal handling procedure. This procedure reads messages from *dlpi_data* and, for each message, calls the user's function. The user's function is defined by the typedef *dlpi_sig_func_t* as follows:

The user-specified function, whose address is passed in the *func* parameter, should return zero, indicating that the API signal handler should take no action. There are no other valid return values specified at this time.

It is assumed that your application does not want any more signals if any one of the following statements are true:

- func is set to NULL
- $sig_num == 0$
- $primitive_mask == 0$

If $sig_num == SIGPOLL$, this routine requests that the stream head generate a SIGPOLL signal when an M_SIG or M_PCSIG containing a SIGPOLL reaches the stream head. DLPI generates M_PCSIGs so that the signal will be generated immediately, even when other messages are queued.

The primitive can consist of any of the following, ORed together in the *primitive_mask* parameter:

- 1 << DL DISCONNECT IND
- 1 << DL_RESET_IND
- 1 << DL_CONNECT_IND
- 1 << DL_RESET_CON
- 1 << DL_CONNECT_CON

Parameters

dlpi_data stream file ID. This is the file descriptor returned by an

open call on the /dev/dlpi_clone file.

func User application's signal message handler function.

sig_num Signal to be generated by the DLPI provider as it sends

to the user a message specified in the *primitive_mask* argument. If *sig_num* is zero, no further signals are

generated by the DLPI provider.

primitive_mask The set of DLPI protocol messages that cause the DLPI

provider to generate a signal as it sends the message to

the user.

Return Unused. Zero is recommended.

dlpi_set_unnum_frame_handler()

Include File(s) **<gcom/dlpiapi.h>**

Description

This routine specifies an application routine to be called when an XID, TEST or unnumbered information U-frame is read by any of the following API routines:

- *dlpi_connect_req()*, which is called by *dlpi_connect()*
- *dlpi_connect_wait()*, which is called by *dlpi_listen()*
- dlpi_read_data()
- dlpi_rcv()
- *dlpi_complete_req()*

Your application's U-frame handler is defined as follows:

Parameters dlpi_data The stream from which to receive a protocol message.

handler Specify NULL to disable

dlpi_set_unnum_frame_handler(). Otherwise, handler specifies an application routine. The specified handler applies to all DLPI streams that the process is using. However, there is not a table that associates FIDs to

handlers.

Return < 0 Failure.

>= 0 The TEST request was successfully sent.

dlpi_test_req()

Prototype int dlpi_test_req (int dlpi_data,

unsigned long pfb,
char *datap,
int length);

Include File(s) **<gcom/dlpiapi.h>**

Description This routine sends a DL_TEST_REQ.

Parameters dlpi_data

s file ID. This is the file descriptor returned by a call to *dlpi_open()*.

pfb If non-zero, poll final bit will be set in the request.

datap Pointer to a user-supplied buffer of data to send with the

request.

length Length of the buffer at datap.

Return < 0 Request failed. See "DLPI API Library Routine Error

Return Defines" on page 118 for a complete list of the

possible error return values.

>= 0 Sent.

dlpi_test_res()

Prototype int dlpi_test_res (int dlpi_data, unsigned long pfb, char *datap, int length); *Include File(s)* <gcom/dlpiapi.h> Description This routine sends a DL_TEST_RES, the response to a DL_TEST_IND. **Parameters** dlpi_data streams file ID. This is the file descriptor returned by a call to *dlpi_open()*. If non-zero, poll final bit will be set in the request. pfb Pointer to a user-supplied buffer of data to send with the datap

request.

Length of the buffer at datap. length

Request failed. See "DLPI API Library Routine Error Return < 0

Return Defines" on page 118 for a complete list of the

possible error return values.

Sent. >=0

>=0

Sent.

dlpi_uic_req()

Prototype int dlpi_uic_req (int dlpi_data, char *datap, int length) ; *Include File(s)* <gcom/dlpiapi.h> This routine sends the user-supplied data via an unnumbered info Description request. **Parameters** dlpi_data streams file ID. This is the file descriptor returned by a call to *dlpi_open()*. Pointer to a user-supplied buffer of data to send with the datap request. Length of the buffer at datap. length Return < 0 Request failed. See "DLPI API Library Routine Error Return Defines" on page 118 for a complete list of the possible error return values.

$dlpi_unbind_dlsap()$

Prototype	<pre>int dlpi_unbind_dlsap (int dlpi_data) ;</pre>	
Include File(s)	<gcom dlpiapi.h=""></gcom>	
Description	This routine sends a DL_UNBIND_REQ primitive to the DLPI provider and waits for the expected DL_OK_ACK response.	
Parameters	dlpi_data	stream file ID. This is the file descriptor returned by an open call on the /dev/dlpi_clone file.
Return	< 0	Request failed. See "DLPI API Library Routine Error Return Defines" on page 118 for a complete list of the possible error return values.
	>= 0	stream successfully unbound.

dlpi_xid_req()

Prototype int dlpi_xid_req (int dlpi_data, pfb, unsigned long char *datap, int length); *Include File(s)* <gcom/dlpiapi.h> Description This routine sends a DL_XID_REQ. **Parameters** dlpi_data streams file ID. This is the file descriptor returned by a call to *dlpi_open()*. If non-zero, poll final bit will be set in the request. pfb datap Pointer to a user-supplied buffer of data to send with the request. Length of the buffer at datap. length Request failed. See "DLPI API Library Routine Error Return < 0 Return Defines" on page 118 for a complete list of the possible error return values.

Sent.

>= 0

dlpi_data,

dlpi_xid_res()

Prototype

unsigned long pfb, char *datap, int length); *Include File(s)* <gcom/dlpiapi.h> Description This routine sends a DL_XID_RES, the response to a DL_XID_IND. **Parameters** dlpi_data streams file ID. This is the file descriptor returned by a call to *dlpi_open()*. If non-zero, poll final bit will be set in the request. pfb

request.

length Length of the buffer at datap.

Return < 0 Request failed. See "DLPI API Library Routine Error

Return Defines" on page 118 for a complete list of the

Pointer to a user-supplied buffer of data to send with the

possible error return values.

>= 0 Sent.

datap

int dlpi_xid_res (int

dlpi_write_data()

Include File(s) **<gcom/dlpiapi.h>**

Description This routine writes a data message to the DLPI driver on the indicated

stream. The data are located in the buffer area pointed to by buf and

consist of cnt bytes.

Parameters dlpi_data The stream to which to write data

buf The buffer from which to write the data

cnt The number of bytes to write

Return < 0 See "DLPI API Library Routine Error Return Defines" on

page 118 for a complete list of the possible error return

values.

>= 0 Data successfully sent.

GCOM, INC. DLPI_WRITE_DATA()

dlpi_xray_req()

Include File(s)

<gcom/dlpiapi.h>

Description:

This routine is used with a newly-opened data stream. When processed, the request sent by this routine links the stream to a DLPI UPA. Any data traffic passing through the UPA will also be presented to this data stream. When data is presented to the X-ray stream, a descriptive header is prepended to the data, providing some insight into the nature of the data which follows. This header conforms to the format:

```
typedef struct
                               /* 1 = send, 2 = receive */
   dl_uns8
              xr_type ;
                                                        * /
   dl_uns8
             xr_rsvd1 ;
                                /* reserved
                                                        * /
   dl uns16
              xr seq ;
                               /* sequence number
   dl_uns32
                              /* Rsys time (millisecs) */
              xr_timestamp ;
                               /* Rsys process number
                                                        * /
   dl_uns16
              xr modid;
   dl uns16
               xr_upa ;
                               /* DLPI UPA number
                                                        * /
} dl_x_ray_hdr_t ;
```

The Rsys time is the number of elapsed milliseconds since the Rsystem was initialized. The process number is an internal module ID number.

Sequence numbers can be useful in keeping track of flow control. Flow control for X-ray streams works a little differently than flow control for other streams. A "normal" stream will queue data until the high water mark is reached, then exert backpressure to stop data flow on that stream until the low water mark is reached. An X-ray stream begins discarding data when the high water mark is reached, then begins re-queuing it when the low water mark is reached. Monitoring sequence numbers will allow the user to determine when data was dropped.

The data which follow this header are the raw CDI message, formatted according to the protocol rules, with all the headers included. An application with some knowledge of the underlying protocols can decipher pieces of these messages to perform data tracing.

As a practical note, when X-ray streams are going to be used, the streams configuration must have an extra CDI UPA or two beyond those needed to support the protocol drivers. These extra UPA's are used by the X-ray streams.

Parameters: fid: The file descriptor of the data stream to use as the X-ray

stream.

upa: The DLPI UPA to be X-rayed.

on_off: Non-zero to turn X-ray on, zero to turn X-ray off.

hi_wat: A high-water mark for the stream. Should probably be

relatively large (60000 or so).

lo_wat: A low-water mark for the stream. Should probably be

about 75% of the high-water mark.

Return Values: 0: The request was successfully sent.

< 0: The request was not sent; error condition.

NDEX

Index

buf parameter, defined 143 \boldsymbol{a} C AcceptConnect input flag for dlpi_rcv(), defined C programming language, knowledge requirements 163 acknowledgement C programming language, requirements 17 defined 39 caller parameter, defined 131 addr_ptr parameter, defined 184 cautions, purpose of 18 address cc command, UNIX 116 DLSAP to be bound to the STREAM 128 clone open used for dlpi_open_data() 152 addressing DLPI 33 clone open, associated w/DL_STYLE1 34 angle bracket conventions 19 close attach session via UNIX close() 49 PPA to STREAM 50 cnt parameter, defined 143 request sent via dlpi send attach req() 174 complete request 131 return values 166 configure the STREAM for the DLSAPs 133 STREAM to PPA 127 confirmation auto_flags parameter, defined 175 defined 39 AutoConnect input flag for dlpi_rcv(), defined 162 conind_nr parameter, defined 128 AutoListen input flag for dlpi_rcv(), defined 163 conn_mgnt parameter, defined 175 AutoReset input flag for dlpi_rcv(), defined 162 connect confirm, in dlpi_conn_con 121 h indication in dlpi_conn_ind 121 indication, response via AcceptConnect input bfr_len parameter, defined 161 flag 163 bind request for HDLC/SDLC 95-96 acknowledge, in dlpi_bind_ack 121 request for LAPB/D & LLC-II 73-74 DLPI STREAM via dlpi_connect() 134 request issued by dlpi connect() 134 request issued via dlpi_bind_dlsap() 128 request sent via dlpi send connect req() 176 request service 52 request, issued by dlpi_connect_req() 136 return values 166 response sent via dlpi_send_connect_res() 177 bind dlsap parameter, defined 134 Connection bind sap used to open/bind a STREAM 134 oriented data transfer, defined 65 Black, Uyless book recommendation 17 oriented service, defined 41

boldface text conventions 19

connection	transfer phase, defined 31
establishment for HDLC/SDLC 86	data link
establishment phase, defined 31	protocol book recommendation 17
establishment service 66	data link addressing components 33
indication value, conind_nr parameter 175	Data Link Connection Identifier. See DLCI
indications that can be queued 128	data link interface, defined 28
indications, incoming number of 117	data link layer, described 28
oriented data transfer, defined 85	Data Link Provider Interface. See DLPI
oriented service, defined 31	Data Link Service Access Point. See DLSAP
rejection for HDLC/SDLC 93-94	Data Link Service. See DLS
rejection in LAPB/LAPD and LLC-II 71-72	data_ptr parameter, defined 161
release for HDLC/SDLC 97-103	datap parameter, defined 184
release for LAPB/LAPD & LLC-II 75–78	defines contained in dlpiapi.h 23
release phase, defined 31	detach
request via DLPIAPI_CONNECT_IND 167	local service 51
reset, HDLC/SDLC 104–107	request sent via dlpi_send_detach_req() 178
reset, LAPB/LAPD & LLC-II 79–82	request, via dlpi_detach_ppa() 140
service, defined 65	return values 166
setup sequence completed return code 167	DISC
Connectionless data transfer, defined 57	DLS provider/user release for HDLC/SDLC secondary 103
service 41	DLS provider/user simultaneous conn. release 78
connectionless service, defined 32	DLS provider-invoked conn. release 77
control messages processed via dlpi_rcv() 159 conventions	DLS provider-invoked release for HDLC/SDLC primary 100
names for routines 20	DLS user aborts conn req after HDLC/SDLC
notes, cautions and warnings 18	SNRM(E) sent 96
text 19 correlation parameter, defined 177	DLS user aborts conn req after LAPB/D & LLC-II SABM(E) sent 74
•	DLS user simultaneous release for HDLC/SDLC 99
d	DLS user/provider release for HDLC/SDLC primary 102
data	DLS user-invoked connection release 75
connectionless transfer of 58	DLS user-invoked release for HDLC/SDLC
connection-oriented transfer 67	primary 97
message read via dlpi_read_data() 171	DLS user-invoked simultaneous conn. release
messages, build 122	76
received via dlpi_rcv() 159	discard_un_iframes parameter, defined 131
return code DLPIAPI_DATA 165	disconnect
transfer for HDLC/SDLC 88	indication, respond via AutoConnect input flag 162

indication, respond via AutoListen input flag 163	dlpi.h contains primitive definitions 161
indications logged 119	dlpi.log
request for HDLC/SDLC 95–96	default log file name 110
request for LAPB/D & LLC-II 73–74	dlpi.log, default log file name 117
request response via	DLPI_logging options, defined 119
DLPIAPI_DISC_COMPLETED 166	dlpi_attach_ppa()
request send via dlpi_send_disconnect_req() 179	attach service 50 described 127
request sent via dlpi_disconnect_req() 142	dlpi_bind_ack
DL_ DLPI primitives, partial listing of 131	bind ack copied to via dlpi_bind_dlsap() 128
DL_ primitives for dlpi_set_signal_handling() 192	dlpi_bind_ack global variable 121
DL_BIND_ACK, copy in dlpi_bind_ack 121	dlpi_bind_dlsap()
DL_CONNECT_IND in dlpi_conn_ind 121	bind service 52
dl_connect_ind_t structure contains correlation	described 128
parameter 177	dlpi_clear_zombies()
DL_DISC_ reason codes, defined 142	described 129
DL_IDLE state w/unbind 53	dlpi_clone, dlpi_data returned via 127
DL_INFO_ACK	dlpi_close 25, 49, 61
returns requested information 47	dlpi_close() 51, 53, 130
DL_STYLE1	dlpi_complete_req()
described 146	attach service 50
DL_STYLE1 DLS provider, defined 33	bind service 52
DL_STYLE2	called upon XID, TEXT or UI U-frame 193
described 146	connection establishment 66
DL_STYLE2 DLS provider, defined 34	connection establishment for HDLC/SDLC 87
DL_UNATTACHED state w/attach 50	described 131
DL_UNBOUND	detach service 51
state w/ bind request 52	DLS provider connection reject for
state w/detach 51	HDLC/SDLC 94
DLP user	DLS provider connection rejection 72
abort conn req after HDLC/SDLC SNRM(E) sent 96	DLS provider/user release for HDLC/SDLC secondary 103
DLPI	DLS provider/user simultaneous conn. release 78
alternate definition 30	DLS provider-invoked conn. reset 82
API, purpose 23	DLS provider-invoked conn. reset 62 DLS user aborts conn req after HDLC/SDLC
architecture 28	SNRM(E) sent 96
described 23	DLS user aborts conn req after LAPB/D &
primitives, partial listing of 131	LLC-II SABM(E) sent 74
provider info via dlpi_get_info 145	DLS user connect reset for HDLC/SDLC
dlpi 130	primary 105
DLPI API library 23	DLS user connect reset for HDLC/SDLC

secondary 106	described 136, 140
DLS user connection reject for HDLC/SDLC 93	DLS provider connection reject for HDLC/SDLC 94
DLS user connection rejection 71	DLS provider connection rejection 72
DLS user connection retraction 73	DLS user connection reject for HDLC/SDLC
DLS user retracts connection request 95	93
DLS user simultaneous connect reset for HDLC/SDLC 107	DLS user connection rejection 71 used by dlpi_connect() 134
DLS user simultaneous release for HDLC/SDLC 99	dlpi_connect_wait() called upon XID, TEXT or UI U-frame 193
DLS user/provider release for HDLC/SDLC primary 102	dlpi_ctl_buf
DLS user-invoked conn. reset 80	size of DLPI_CTL_BUF_SIZE 117
DLS user-invoked connection release 75	dlpi_ctl_buf global variable 121
DLS user-invoked release for HDLC/SDLC	DLPI_CTL_BUF_SIZE constant 117
primary 97 DLS user-invoked release for HDLC/SDLC	DLPI_CTL_BUF_SIZE, passed in dlpi_ctl_buf 121
secondary 98	dlpi_ctl_cnt global variable 122
DLS user-invoked simultaneous conn. release	dlpi_data parameter, defined 127
76	dlpi_data_buf
info reporting service 46	size of DLPI_DATA_BUF_SIZE 117
simultaneous DLS user connection reset 81	dlpi_data_buf global variable 122
dlpi_configure_dlsaps()	DLPI_DATA_BUF_SIZE constant 117
dlpi_configure_dlsaps() defined 133	DLPI_DATA_BUF_SIZE constant 117 DLPI_DATA_BUF_SIZE in dlpi_data_buf 122
defined 133	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122
defined 133 dlpi_conn_con global variable 121	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122 dlpi_data_cnt global variable 122
defined 133 dlpi_conn_con global variable 121 dlpi_conn_ind global variable 121	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122 dlpi_data_cnt global variable 122 dlpi_decode_ctl(), described 138
defined 133 dlpi_conn_con global variable 121 dlpi_conn_ind global variable 121 dlpi_connect()	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122 dlpi_data_cnt global variable 122 dlpi_decode_ctl(), described 138 dlpi_decode_disconnect_reason(), described 139
defined 133 dlpi_conn_con global variable 121 dlpi_conn_ind global variable 121 dlpi_connect() connection establishment 66	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122 dlpi_data_cnt global variable 122 dlpi_decode_ctl(), described 138 dlpi_decode_disconnect_reason(), described 139 DLPI_DEFAULT_PPA constant 117
defined 133 dlpi_conn_con global variable 121 dlpi_conn_ind global variable 121 dlpi_connect() connection establishment 66 connection establishment for HDLC/SDLC 87	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122 dlpi_data_cnt global variable 122 dlpi_decode_ctl(), described 138 dlpi_decode_disconnect_reason(), described 139 DLPI_DEFAULT_PPA constant 117 dlpi_detach_ppa()
defined 133 dlpi_conn_con global variable 121 dlpi_conn_ind global variable 121 dlpi_connect() connection establishment 66 connection establishment for HDLC/SDLC 87 connection rejection 71 described 134 DLS provider connection reject for	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122 dlpi_data_cnt global variable 122 dlpi_decode_ctl(), described 138 dlpi_decode_disconnect_reason(), described 139 DLPI_DEFAULT_PPA constant 117 dlpi_detach_ppa() detach_service 51
defined 133 dlpi_conn_con global variable 121 dlpi_conn_ind global variable 121 dlpi_connect() connection establishment 66 connection establishment for HDLC/SDLC 87 connection rejection 71 described 134 DLS provider connection reject for HDLC/SDLC 94	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122 dlpi_data_cnt global variable 122 dlpi_decode_ctl(), described 138 dlpi_decode_disconnect_reason(), described 139 DLPI_DEFAULT_PPA constant 117 dlpi_detach_ppa() detach service 51 dlpi_discon_req()
defined 133 dlpi_conn_con global variable 121 dlpi_conn_ind global variable 121 dlpi_connect() connection establishment 66 connection establishment for HDLC/SDLC 87 connection rejection 71 described 134 DLS provider connection reject for HDLC/SDLC 94 DLS provider connection rejection 72	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122 dlpi_data_cnt global variable 122 dlpi_decode_ctl(), described 138 dlpi_decode_disconnect_reason(), described 139 DLPI_DEFAULT_PPA constant 117 dlpi_detach_ppa() detach service 51 dlpi_discon_req() described 141
defined 133 dlpi_conn_con global variable 121 dlpi_conn_ind global variable 121 dlpi_connect() connection establishment 66 connection establishment for HDLC/SDLC 87 connection rejection 71 described 134 DLS provider connection reject for HDLC/SDLC 94	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122 dlpi_data_cnt global variable 122 dlpi_decode_ctl(), described 138 dlpi_decode_disconnect_reason(), described 139 DLPI_DEFAULT_PPA constant 117 dlpi_detach_ppa() detach service 51 dlpi_discon_req() described 141 dlpi_disconnect_req()
defined 133 dlpi_conn_con global variable 121 dlpi_conn_ind global variable 121 dlpi_connect() connection establishment 66 connection establishment for HDLC/SDLC 87 connection rejection 71 described 134 DLS provider connection reject for HDLC/SDLC 94 DLS provider connection rejection 72 DLS user connection reject for HDLC/SDLC	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122 dlpi_data_cnt global variable 122 dlpi_decode_ctl(), described 138 dlpi_decode_disconnect_reason(), described 139 DLPI_DEFAULT_PPA constant 117 dlpi_detach_ppa() detach service 51 dlpi_discon_req() described 141 dlpi_disconnect_req() described 142 DLS provider/user release for HDLC/SDLC
defined 133 dlpi_conn_con global variable 121 dlpi_conn_ind global variable 121 dlpi_connect() connection establishment 66 connection establishment for HDLC/SDLC 87 connection rejection 71 described 134 DLS provider connection reject for HDLC/SDLC 94 DLS provider connection rejection 72 DLS user connection reject for HDLC/SDLC 93 DLS user connection rejection 71 dlpi_connect_host() 134 dlpi_connect_req()	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122 dlpi_data_cnt global variable 122 dlpi_decode_ctl(), described 138 dlpi_decode_disconnect_reason(), described 139 DLPI_DEFAULT_PPA constant 117 dlpi_detach_ppa() detach service 51 dlpi_discon_req() described 141 dlpi_disconnect_req() described 142 DLS provider/user release for HDLC/SDLC secondary 103 DLS user simultaneous release for HDLC/SDLC 99 DLS user/provider release for HDLC/SDLC
defined 133 dlpi_conn_con global variable 121 dlpi_conn_ind global variable 121 dlpi_connect() connection establishment 66 connection establishment for HDLC/SDLC 87 connection rejection 71 described 134 DLS provider connection reject for HDLC/SDLC 94 DLS provider connection rejection 72 DLS user connection reject for HDLC/SDLC 93 DLS user connection rejection 71 dlpi_connect_host() 134 dlpi_connect_req() called upon XID, TEXT or UI U-frame 193	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122 dlpi_data_cnt global variable 122 dlpi_decode_ctl(), described 138 dlpi_decode_disconnect_reason(), described 139 DLPI_DEFAULT_PPA constant 117 dlpi_detach_ppa() detach service 51 dlpi_discon_req() described 141 dlpi_disconnect_req() described 142 DLS provider/user release for HDLC/SDLC secondary 103 DLS user simultaneous release for HDLC/SDLC 99 DLS user/provider release for HDLC/SDLC primary 102
defined 133 dlpi_conn_con global variable 121 dlpi_conn_ind global variable 121 dlpi_connect() connection establishment 66 connection establishment for HDLC/SDLC 87 connection rejection 71 described 134 DLS provider connection reject for HDLC/SDLC 94 DLS provider connection rejection 72 DLS user connection reject for HDLC/SDLC 93 DLS user connection rejection 71 dlpi_connect_host() 134 dlpi_connect_req()	DLPI_DATA_BUF_SIZE in dlpi_data_buf 122 dlpi_data_cnt global variable 122 dlpi_decode_ctl(), described 138 dlpi_decode_disconnect_reason(), described 139 DLPI_DEFAULT_PPA constant 117 dlpi_detach_ppa() detach service 51 dlpi_discon_req() described 141 dlpi_disconnect_req() described 142 DLS provider/user release for HDLC/SDLC secondary 103 DLS user simultaneous release for HDLC/SDLC 99 DLS user/provider release for HDLC/SDLC

DLS user-invoked release for HDLC/SDLC secondary 98	described 156 sample code 112
DLS user-invoked simultaneous conn. release	dlpi_put_both()
76	described 157
dlpi_get_a_msg()	dlpi_put_proto()
described 143	described 158
dlpi_get_info_strm()	dlpi_rcv()
described 145	attach service 50
info reporting service 46	bind service 52
dlpi_get_style_strm()	
described 146	called upon XID, TEXT or UI U-frame 193 connection establishment 66
dlpi_init()	
described 147	connection establishment for HDLC/SDLC 87
receives DLPI_logging options 119	connection mode data xfer 67
use before opening connectionless session 58	data xfer for HDLC/SDLC 88
dlpi_init_FILE(), described 148	described in reference appendix 159
dlpi_listen()	detach service 51
connection establishment 66	DLS provider connection reject for HDLC/SDLC 94
connection establishment for HDLC/SDLC 87	DLS provider connection rejection 72
described 149	DLS provider/user release for HDLC/SDLC
DLS user aborts conn req after HDLC/SDLC	secondary 103
SNRM(E) sent 96	DLS provider/user simultaneous conn. release
DLS user aborts conn req after LAPB/D &	78
LLC-II SABM(E) sent 74	DLS provider-invoked conn. release 77
sample code 110	DLS provider-invoked conn. reset 82
dlpi_listen_host 149	DLS provider-invoked release for
dlpi_listen_host() 149	HDLC/SDLC primary 100
DLPI_LOG_NAME constant 117	DLS provider-invoked release for HDLC/SDLC secondary 101
DLPI_N_CONINDS constant 117	DLS user aborts conn req after HDLC/SDLC
dlpi_open 25, 130, 152	SNRM(E) sent 96
dlpi_open() 130	DLS user aborts conn req after LAPB/D &
dlpi_open_data()	LLC-II SABM(E) sent 74
described 152	DLS user connect reset for HDLC/SDLC
open local service 48	primary 105
dlpi_open_log()	DLS user connect reset for HDLC/SDLC secondary 106
described 153	DLS user connection reject for HDLC/SDLC
dlpi_perror()	93
described 154	DLS user connection rejection 71
dlpi_print_msg()	DLS user connection retraction 73
described 155	DLS user retracts connection request 95
dlpi_printf()	
	DLS user simultaneous connect reset for

HDLC/SDLC 107	91
DLS user simultaneous release for HDLC/SDLC 99	UI U-frame transfer for HDLC/SDLC secondary 92
DLS user/provider release for HDLC/SDLC	XID connection mode service 68
primary 102	XID service 59
DLS user-invoked conn. reset 80	XID transfer for HDLC/SDLC 89
DLS user-invoked connection release 75	dlpi_reset_req()
DLS user-invoked release for HDLC/SDLC primary 97	DLS user connect reset for HDLC/SDLC primary 105
DLS user-invoked release for HDLC/SDLC secondary 98	DLS user connect reset for HDLC/SDLC secondary 106
DLS user-invoked simultaneous conn. release 76	DLS user simultaneous connect reset for HDLC/SDLC 107
info reporting service 46	DLS user-invoked conn. reset 80
simultaneous DLS user connection reset 81	simultaneous DLS user connection reset 81
TEST connection mode service 69	dlpi_reset_res()
TEST service 60	described 173
TEST transfer for HDLC/SDLC 90	DLS provider-invoked conn. reset 82
U-frame (data) exchange 58	DLS user connect reset for HDLC/SDLC
U-frame xfer connection service 70	primary 105
UI U-frame transfer for HDLC/SDLC primary	DLS user-invoked conn. reset 80
91	dlpi_send_attach_req()
UI U-frame transfer for HDLC/SDLC secondary 92	attach service 50
XID connection mode service 68	described 174
XID service 59	dlpi_send_bind_req()
XID transfer for HDLC/SDLC 89	bind service 52
dlpi_rcv_msg()	described 175
described 169	dlpi_send_connect_req()
dlpi_read_data()	connection establishment 66
called upon XID, TEXT or UI U-frame 193	connection establishment for HDLC/SDLC 87
connection mode data xfer 67	connection rejection 71
data xfer for HDLC/SDLC 88	described 176
described 171	DLS provider connection reject for HDLC/SDLC 94
overwrites dlpi_ctl_buf 121	DLS provider connection rejection 72
sample code 112 TEST connection mode service 69	DLS user aborts conn req after HDLC/SDLC SNRM(E) sent 96
TEST service 60 TEST transfer for HDLC/SDLC 90	DLS user aborts conn req after LAPB/D & LLC-II SABM(E) sent 74
U-frame (data) exchange 58	DLS user connect reset for HDLC/SDLC secondary 106
U-frame xfer connection service 70 UI U-frame transfer for HDLC/SDLC primary	DLS user connection reject for HDLC/SDLC 93

DLS user simultaneous connect reset for
HDLC/SDLC 107
DLS user-invoked conn. reset 80
simultaneous DLS user connection reset 81
dlpi_send_reset_res()
described 177, 182
DLS provider-invoked conn. reset 82
DLS user connect reset for HDLC/SDLC primary 105
DLS user-invoked conn. reset 80
dlpi_send_stats_req()
described 183
dlpi_send_test_req()
described 184
TEST connection mode service 69
TEST service 60
TEST transfer for HDLC/SDLC 90
dlpi_send_test_res()
described 185
dlpi_send_uic()
described 186
U-frame (data) exchange 58
U-frame xfer connection service 70
UI U-frame transfer for HDLC/SDLC primary 91
UI U-frame transfer for HDLC/SDLC
secondary 92
dlpi_send_unbind_req()
described 187
unbind service 53
dlpi_send_xid_req()
described 188, 189
XID connection mode service 68
XID service 59
XID transfer for HDLC/SDLC 89
dlpi_send_xid_res()
XID connection mode service 68
XID service 59
XID transfer for HDLC/SDLC 89
dlpi_service_mode global variable 122
dlpi_set_log_size()

described 190	DLPIAPI_CONNECT_IND return code for
dlpi_set_signal_handling()	dlpi_rcv() 167
described 191	DLPIAPI_DATA return code for dlpi_rcv() 165
dlpi_set_unnum_frame_handler() described 193	DLPIAPI_DETACHED return code for dlpi_rcv() 166
	DLPIAPI_DISC_COMPLETED return code for
dlpi_sig_func_t structure definition 191	dlpi_rcv() 166
dlpi_sig_func_t, signal handling prototype 120	DLPIAPI_DISC_IND return code for dlpi_rcv() 166
dlpi_test_req()	
described 194	DLPIAPI_ERROR_ACK return code for dlpi_rcv() 167
dlpi_test_res()	DLPIAPI_INFO_ACK return code for dlpi_rcv()
described 195	167
dlpi_token parameter, defined 177	DLPIAPI_OK_ACK return code for dlpi_rcv()
dlpi_uic_req()	167
described 196	DLPIAPI_OTHER return code for dlpi_rcv() 168
dlpi_unbind_dlsap()	DLPIAPI_RESET_COMPLETE return code for
described 197	dlpi_rcv() 167
unbind service 53	DLPIAPI_RESET_INDICATION return code for dlpi_rcv() 166
dlpi_write_data()	•
connection mode data xfer 67	DLPIAPI_TEST_CON return code for dlpi_rcv() 168
data xfer for HDLC/SDLC 88	DLPIAPI_TEST_IND return code for dlpi_rcv()
described 200	168
dlpi_xid_req()	DLPIAPI_UIC_IND return code for dlpi_rcv()
described 198	168
dlpi_xid_res()	DLPIAPI_UNBOUND return code for dlpi_rcv()
described 199	166
dlpiapi.a	DLPIAPI_UNUSABLE return value, described 146
described 23	DLPIAPI_XID_CON return code for dlpi_rcv()
linking 116	168
dlpiapi.h	DLPIAPI_XID_IND return code for dlpi_rcv()
contains defines, globals and prototypes 116	168
described 23	DLS
including 110	provider, described 28
including it in your application 116	user, defined 28
DLPIAPI_ errors, defined 118	DLS provider
DLPIAPI_ return codes, partial list w/ definitions	connection reject for HDLC/SDLC 94
161	connection rejection 72
DLPIAPI_ATTACHED return code for dlpi_rcv() 166	DLS user release for HDLC/SDLC primary 102
DLPIAPI_BOUND return code for dlpi_rcv() 166 DLPIAPI_CONNECT_COMPLETE return code	DLS user release for HDLC/SDLC secondary 103
for dlpi_rcv() 167	DLS user-invoked conn. release 78

invoked connection release 77	DLS provider/user release for HDLC/SDLC
invoked connection reset 82	secondary 103
invoked release for HDLC/SDLC primary 100	DLS provider-invoked release for HDLC/SDLC primary 100
invoked release for HDLC/SDLC secondary 101	DLS provider-invoked release for HDLC/SDLC secondary 101
style 1 and style 2 33	DLS user connect reset for HDLC/SDLC
DLS user	secondary 106
abort conn req after LAPB/D & LLC-II SABM(E) sent 74	DLS user connection reject for HDLC/SDL0 93
connect reset for HDLC/SDLC secondary 106	DLS user connection rejection 71
connection rejection 71	DLS user/provider release for HDLC/SDLC
connection rejection for HDLC/SDLC 93	primary 102 DLS user-invoked release for HDLC/SDLC secondary 98
connection reset 80	
connection reset for HDLC/SDLC primary	
105	duplication of frame relay frames 57
DLS provider release for HDLC/SDLC primary 102	
DLS provider release for HDLC/SDLC secondary 103	e
DLS provider-invoked conn. release 78	enter vs. type 19
invoked connection release 75	error return defines 118
invoked release for HDLC/SDLC primary 97	
invoked release for HDLC/SDLC secondary 98	f
retracts connection request 73, 95	fcntl() UNIX system call 162
simultaneous connect reset for HDLC/SDLC 107	file descriptor returned 48
simultaneous connection reset 81	
simultaneous invoked connection release 76	flags parameter, defined 162
simultaneous release for HDLC/SDLC 99	fmt parameter, defined 156 fork options, N/A 123
TEST transfer for HDLC/SDLC 90	•
TEXT transfer for LAPB/D & LLC-II 69	fork_options parameter, defined 150
XID U-frame transfer for HDLC/SDLC 89	frame delivery without duplication 57
XID U-frame transfer for LAPB/D & LLC-II 68	frame relay possible frame duplication in a WAN 57
DLSAP	func parameter, described 192
(dis)associate w/ a STREAM 52	
address to be bound to the STREAM 128	
configure for the STREAM 133	\boldsymbol{g}
dlsap parameter, defined 128	Gcom Protocol Appliance 24
DLSAP, defined 34	Gcom Remote API
DM	architecture 24
Divi	client server model 24

running the RAPI server 25	l
using the RAPI library 25 Geom_lstn program, described 110–112	LAPB/LAPD connection-oriented data transfer 65–82
getmsg call issued via dlpi_get_a_msg() 143	length parameter, defined 184
getmsg failed 118	lgth parameter, defined 158
getmsg()	link
to access DLS provider 28	header files 116
global variables, defined 121	listen
	for an incoming connection 149
h	LLC-II connection-oriented data transfer 65–82
	local service 47–53
handler parameter, described 193	defined 31
hang-up notification 118	local_dlsap parameter, defined 133
HDLC connection-oriented data transfer 85–107	log
	default options 119
i	default, DLPI_LOG_NAME 117
	file specified via dlpi_init_FILE() 148
I-frame	options, defined 119
connection mode data xfer 67	log_FILE parameter, defined 148
data xfer for HDLC/SDLC 88	log_name parameter, defined 147
indication	log_optns parameter, defined 147
defined 39	loopback program, Gcom_lstn 110
information	
acknowledge, response via ReturnInfo input flag 163	m
request response via DLPIAPI_INFO_ACK 167	M_PCSIG 191
request sent via dlpi_send_info_req() 180	M_PROTO
information retrieved via dlpi_get_info() 145	length in dlpi_ctl_cnt 122
initialize	received in dlpi_ctl_buf 121
application & DLPI API Library 147	M_SIG 191
DLPI API library 110	major device, PPA assignment 33
error 118	management requests handled via dlpi_rcv() 159
software w/passed log file 148	message
italic text conventions 19	build data 122 written to log 119
_	minor device, PPA assignment 33
\boldsymbol{k}	Multilink Procedure. See MLP

212 DLPI API GUIDE APRIL 2003

kernel-level interfaces 28

n	attach STREAM to 127
11 11 1/0	default, DLPI_DEFAULT_PPA 117
non-blocking I/O	defined 33
set 118	detach service 51
supported via dlpi_rcv() 160	ppa parameter, defined 127
notes, purpose of 18	prefix parameter, defined 138
Novell documentation 17	PRIM_type field 121
	primitive_mask argument 192
0	primitive_mask parameter, described 192
	printf()
open	functionality w/dlpi_printf() 156
DLPI STREAM via dlpi_connect() 134	protocol message
error 118	decode DLPI via dlpi_decode_ctl() 138
session via dlpi_open_data() call 48	protocol messages, log 119
STREAM to DLPI driver via dlpi_open_data()	prototype declarations contained in dlpiapi.h 23
152	prototype message
open()	write via dlpi_put_proto() 158
to access DLS user 30	putmsg failed 118
open() system call accepted 48	putmsg()
OSI Reference Model, knowledge requirements 17	to access DLS provider 28
out_code parameter, defined 161	
	r
p	1
I	reason code parameter, defined 142
path	recommended reading 17
setting up to other DL users 33	reference model, presented 37
peer_sap parameter, defined 136	reject
pfb parameter, defined 184	connection establishment for LAPB/D & LLC-
Physical Point of Attachment. See PPA	II 71
Poll	connection for HDLC/SDLC 93
DLS provider-invoked release for	rejection
HDLC/SDLC secondary 101	connection, called DLS user 71
DLS user connect reset for HDLC/SDLC secondary 106	connection, DLS provider 72
DLS user-invoked release for HDLC/SDLC	release connection
secondary 98	HDLC/SDLC 97–103
Poll UI U-frame transfer for HDLC/SDLC	LAPB/LAPD and LLC-II 75–78
secondary 92	Remote API 24
PPA	remote_dlsap parameter, defined 133
assign via attach service 50	request
assigned via DL_STYLE1 33	defined 39

rejected via DLPIAPI_ERROR_ACK 167	DLS provider-invoked conn. reset 82
request parameter, defined 131	DLS user aborts conn req after LAPB/D &
requirements, knowledge 17	LLC-II SABM(E) sent 74
reset	DLS user connection rejection 71
indication received via	DLS user-invoked conn. reset 80
DLPIAPI_RESET_INDICATION 166	simultaneous DLS user connection reset 81
indication, respond via AutoReset input flag	SAP, defined 34
162	screen display 19
indications, logged 119	SDLC connection-oriented data transfer 85–107
request connection, HDLC/SDLC 104–107	service
request connection, LAPB/D & LLC-II 79–82	extensions to DLPI 43
request sent via dlpi_reset_req() 172	local 47–53
request sent via dlpi-send_reset_req() 181	local, defined 47
response sent via dlpi_reset_res() 173	supported DLPI 41
response sent via dlpi_send_reset_res() 182	service provider
sequence completed via DLPIAPI_RESET_COMPLETE 167	defined 37
reset connection	service user, defined 37
LAPB/LAPD & LLC-II 79–82	service, defined 37
reset service, defined 79	service_mode parameter, defined 175
response	session
defined 39	close a connectionless 61
retraction	close via UNIX close() 49
connection establishment 73–74	via dlpi_open_data() 48
RetryOnSignal input flag for dlpi_rcv(), defined	sig_num 191
164	sig_num parameter, described 192
ReturnInfo input flag for dlpi_rcv(), defined 163	signal
ReturnTEST input flag for dlpi_rcv(), defined 164	handling via dlpi_set_signal_handling() 191
ReturnUIC input flag for dlpi_rcv(), defined 164 ReturnXID input flag for dlpi_rcv(), defined 164	signal for system call via RetryOnSignal input flag 164
routine	signal handling
DLPI/SDLC dlpiapi.a file 23	logged 119
naming conventions 20	prototype 120
RR	SIGPOLL 191
connection mode data xfer 67	SNRM
data xfer for HDLC/SDLC 88	DLS user connect reset for HDLC/SDLC secondary 106
	SNRM(E)
S	connection establishment for HDLC/SDLC 87
SABM(E)	DLS user aborts conn req after HDLC/SDLC SNRM(E) sent 96
connection establishment 66	DLS user connect reset for HDLC/SDLC primary 105

DLS user connection reject for HDLC/SDLC	HDLC/SDLC 99
93	DLS user-invoked conn. reset 80
DLS user simultaneous connect reset for HDLC/SDLC 107	DLS user-invoked connection release 75
stderr, messages written to 119	DLS user-invoked release for HDLC/SDLC primary 97
STREAM	DLS user-invoked simultaneous conn. release 76
configures for the DLSAPs passed 133	
style 2 attach service 50	U-frame
style type returned via dlpi_get_style() 146	connection mode service 68–70
	input flags to receive 164
t	receiving via dlpi_rcv() 159
ı	transfer for HDLC/SDLC 89–92
terminology conventions 19	U-frames
TEST	connectionless transfer of 58
application to call when read 193	return codes signaling 168
connection mode service 69	UI
request sent via dlpi_send_test_req() 184	U-frame sent via dlpi_send_uic() 186
response sent via dlpi_send_test_res() 185	U-frame transfer for HDLC/SDLC primary 91
service (command/response) 60	U-frame transfer for HDLC/SDLC secondary 92
text conventions 19	U-frame, application to call when read 193
type vs. enter 19	unbind
	request sent via dlpi_send_unbind_req() 187
u	request, send via dlpi_unbind_dlsap() 197
	return values 166
UA	service 53
connection establishment 66	Univel documentation 17
connection establishment for HDLC/SDLC 87	UNIX
DLS provider/user simultaneous conn. release 78	cc command 116
DLS provider-invoked conn. release 77	error messages, logged 119
DLS provider-invoked conn. reset 82	fcntl() system call 162
DLS user aborts conn req after HDLC/SDLC SNRM(E) sent 96	unnum_frame_t U-frame handler structure 193 Unnumbered information U-frame
DLS user aborts conn req after LAPB/D &	transfer 70
LLC-II SABM(E) sent 74	Uyless Black's book recommendation 17
DLS user connect reset for HDLC/SDLC primary 105	
DLS user connect reset for HDLC/SDLC secondary 106	V
DLS user simultaneous connect reset for HDLC/SDLC 107	verbose logging example 110
DLS user simultaneous release for	

W

warnings, purpose of 18
write
data message via dlpi_write_data() 200
prototype message to DLPI driver via
dlpi_put_proto() 158

\boldsymbol{x}

XID

application to call when read 193 connection mode service 68 request sent via dlpi_send_xid_req() 188 response send via dlpi_send_xid_res() 189 U-frame service (command/response) 59 U-frame transfer for HDLC/SDLC 89