Asynchronous Wire Service (AWS) Programmer Guide

DC 900-1324I

Protogate, Inc.
12225 World Trade Drive, Suite R
San Diego, CA 92128
September 2011

Protogate, Inc.
12225 World Trade Drive, Suite R
San Diego, CA 92128
(858) 451-0865

AWS Programmer Guide

Printed in the United States of America

Contents

List of Figures

List of Tables

Preface

Purpose of Document

This document describes the operation and programming interface required to use Protogate's Asynchronous Wire Service (AWS) software product for Protogate's Freeway communications server or embedded ICP.

Note

In this document, the term "Freeway" can mean either a Freeway server or an embedded ICP. For the embedded ICP, also refer to the user guide for your ICP and operating system (for example, the *ICP2432 User Guide for Windows NT*).

Intended Audience

This document should be read by programmers who want to use Protogate's Freeway communications server for asynchronous wire services applications. You should understand the Freeway data link interface (DLI), as explained in the *Freeway Data Link Interface Reference Guide* for a Freeway server, or the embedded DLITE interface, as explained in the user guide for your embedded ICP and operating system.

Required Equipment

The AWS product requires the following two major hardware components to operate:

- a Freeway communications server or an embedded ICP that runs the communications software
- a client computer that runs the following:
  - TCP/IP (for a Freeway server)
  - Data link interface (DLI) or embedded DLITE interface
  - the user application program

Organization of Document

Chapter 1 is an overview of Freeway and the AWS product.

Chapter 2 describes how to use the data link interface (DLI) between the client application program and the AWS communications software running on the Freeway ICP.

Chapter 3 describes the link configuration options available on the AWS software package.

Chapter 4 describes how to configure the AWS link options using the dlicfg preprocessor program.

Appendix A lists the error codes and their causes.

Appendix B details the AWS command and response formats.

Protogate References

The following general product documentation list is to familiarize you with the available Protogate Freeway and embedded ICP products. The applicable product-specific reference documents are mentioned throughout each document (also refer to the "readme" file shipped with each product). Most documents are available on-line at Protogate's web site, www.protogate.com.

Hardware Support

- *Freeway 3112 Hardware Installation Guide* DC-900-2016
- *Freeway 3212 Hardware Installation Guide* DC-900-2017
- *Freeway 3412 Hardware Installation Guide* DC-900-2018
- *Freeway 3612 Hardware Installation Guide* DC-900-2019
- *ICP2432A Hardware Description and Theory of Operation* DC-900-1501
- *ICP2432A Hardware Installation Guide* DC-900-1502
- *ICP2432B Hardware Description and Theory of Operation* DC-900-2006
- *ICP2432B Hardware Installation Guide* DC-900-2009

Freeway Software Installation and Configuration Support

- *Freeway Message Switch User Guide* DC-900-1588
- *Freeway Release Addendum: Client Platforms* DC-900-1555
- *Freeway User Guide* DC-900-1333
- *Loopback Test Procedures* DC-900-1533

Embedded ICP Installation and Programming Support

- *ICP2432 User Guide for Digital UNIX* DC-900-1513
- *ICP2432 User Guide for OpenVMS Alpha* DC-900-1511
- *ICP2432 User Guide for OpenVMS Alpha (DLITE Interface)* DC-900-1516
- *ICP2432 User Guide for Solaris STREAMS* DC-900-1512
- *ICP2432 User Guide for Windows NT* DC-900-1510
- *ICP2432 User Guide for Windows NT (DLITE Interface)* DC-900-1514

Application Program Interface (API) Programming Support

- *Freeway Data Link Interface Reference Guide* DC-900-1385
- *Freeway Transport Subsystem Interface Reference Guide* DC-900-1386
- *QIO/SQIO API Reference Guide* DC-900-1355

Socket Interface Programming Support

- *Freeway Client-Server Interface Control Document* DC-900-1303

Toolkit Programming Support

- *Freeway Server-Resident Application (SRA) Programmer Guide* DC-900-1325
- *OS/Protogate Programmer's Guide* DC-900-2008
- *Protocol Software Toolkit Programmer Guide (ICP2432B)* DC-900-2007

Protocol Support

- *ADCCP NRM Programmer Guide* DC-900-1317
- *Asynchronous Wire Service (AWS) Programmer Guide* DC-900-1324
- *AUTODIN Programmer Guide* DC-908-1558
- *Bit-Stream Protocol Programmer Guide* DC-900-1574
- *BSC Programmer Guide* DC-900-1340
- *BSCDEMO User Guide* DC-900-1349
- *BSCTRAN Programmer Guide* DC-900-1406
- *DDCMP Programmer Guide* DC-900-1343
- *FMP Programmer Guide* DC-900-1339
- *Military/Government Protocols Programmer Guide* DC-900-1602
- *N/SP-STD-1200B Programmer Guide* DC-908-1359
- *SIO STD-1300 Programmer Guide* DC-908-1559
- *X.25 Call Service API Guide* DC-900-1392
- *X.25/HDLC Configuration Guide* DC-900-1345
- *X.25 Low-Level Interface* DC-900-1307

Document Conventions

This document follows the most significant byte first (MSB) and most significant word first (MSW) conventions for bit-numbering and byte-ordering. In all packet transfers between the client applications and the ICPs, the ordering of the byte stream is preserved.

The term "Freeway" refers to any of the Freeway server models (for example, Freeway 500/3100/3200/3400 PCI-bus servers, Freeway 1000 ISA-bus servers, or Freeway 2000/4000/8800 VME-bus servers). References to "Freeway" also may apply to an embedded ICP product using DLITE (for example, the embedded ICP2432 using DLITE on a Windows NT system).

Physical "ports" on the ICPs are logically referred to as "links." However, since port and link numbers are usually identical (that is, port 0 is the same as link 0), this document uses the term "link."

Program code samples are written in the "C" programming language.

Revision History

The revision history of the *AWS Programmer Guide*, Protogate document DC 900-1324I, is recorded below:

Revision	Release Date	Description
DC 900-1324A	June 1994	Preliminary release
DC 900-1324B	October 1994	Full release: Minor modifications and updated error codes Update file names for software release 2.1 Change the usICPStatus field to iICPStatus and change the usProtModifier field to iProtModifier (page 38)
DC 900-1324C	May 1995	Minor modifications throughout document Add new dlControl function to Table 2–4 on page 37 Add electrical interface option (Section 3.27 on page 74) Add Windows NT to Chapter 4 and Appendix B

Revision	Release Date	Description
DC 900-1324D	August 1997	Minor modifications throughout document Add browser interface for configuration Add new dlpErrString function to Table 2–4 on page 37 Corrections to Table 3–3 on page 72 and Table 3–4 on page 73 Minor changes to Chapter 4 and Appendix B
DC 900-1324E	June 1998	Minor modifications throughout document Update Section 1.1 through Section 1.4 to discuss embedded ICPs Remove browser interface support Add new dlSyncSelect function to Table 2–4 on page 37 Minor modifications to Section 2.4.3 on page 49, Table 2–7 on page 51, and Section 2.5.2 on page 52 Minor changes to Chapter 4 and Appendix B
DC 900-1324F	August 1998	Add Clear Statistics command (Section 2.4.1.6 on page 44) and Statistics Report request (Section 2.4.2.4 on page 47) Add and modify codes in Table 2–5 on page 40 and Table 2–7 on page 51
DC 900-1324G	September 1999	Minor modifications throughout for embedded ICP users Update electrical interface information (Table 3–1 on page 56) Remove loopback test appendix. This information is included in the *Loopback Test Procedures* document (for a Freeway server) or the user guide for your embedded ICP and operating system (for example, the *ICP2432 User Guide for Windows NT*). Add command and response formats (Appendix B)
DC 900-1324H	November 2003	Update contact information for Protogate.
DC 900-1324I	September 2011	Add Send Crypto Resync command (Section 2.4.1.7 on page 44). Add Transmit Expiration (Section 3.8 on page 61 and Section 2.5.2.4 on page 52). Update Table A–2 on page 87 by adding Crypto resync and Transmit expiration; reworked the error/status value column to specify the error/status bit number. Update reference documents.

Customer Support

If you are having trouble with any Protogate product, call us at (858) 451-0865 Monday through Friday between 8 a.m. and 5 p.m. Pacific time.

You can also fax your questions to us at (877) 473-0190 any time. Please include a cover sheet addressed to “Customer Service.”

We are always interested in suggestions for improving our products. You can use the report form in the back of this manual to send us your recommendations.

Chapter

1 Introduction

1.1 Product Overview

Protogate provides a variety of wide-area network (WAN) connectivity solutions for real-time financial, defense, telecommunications, and process-control applications. Protogate's Freeway server offers flexibility and ease of programming using a variety of LAN-based server hardware platforms. Now a consistent and compatible embedded intelligent communications processor (ICP) product offers the same functionality as the Freeway server, allowing individual client computers to connect directly to the WAN.

Both Freeway and the embedded ICP use the same data link interface (DLI). Therefore, migration between the two environments simply requires linking your client application with the proper library. Various client operating systems are supported (for example, UNIX, VMS, and Windows NT).

Protogate protocols that run on the ICPs are independent of the client operating system and the hardware platform (Freeway or embedded ICP).

1.1.1 Freeway Server

Protogate's Freeway communications servers enable client applications on a local-area network (LAN) to access specialized WANs through the DLI. The Freeway server can be any of several models (for example, Freeway 3112, 3212, 3412, or 3612). The Freeway server is user programmable and communicates in real time. It provides multiple data links and a variety of network services to LAN-based clients. Figure 1–1 shows the Freeway configuration.

To maintain high data throughput, Freeway uses a multi-processor architecture to support the LAN and WAN services. The LAN interface is managed by a single-board computer, called the server processor. It uses the commercially available FreeBSD operating system to provide a full-featured base for the LAN interface and layered services needed by Freeway.

Freeway can be configured with multiple WAN interface processor boards, each of which is a Protogate ICP. Each ICP runs the communication protocol software using Protogate's real-time operating system.

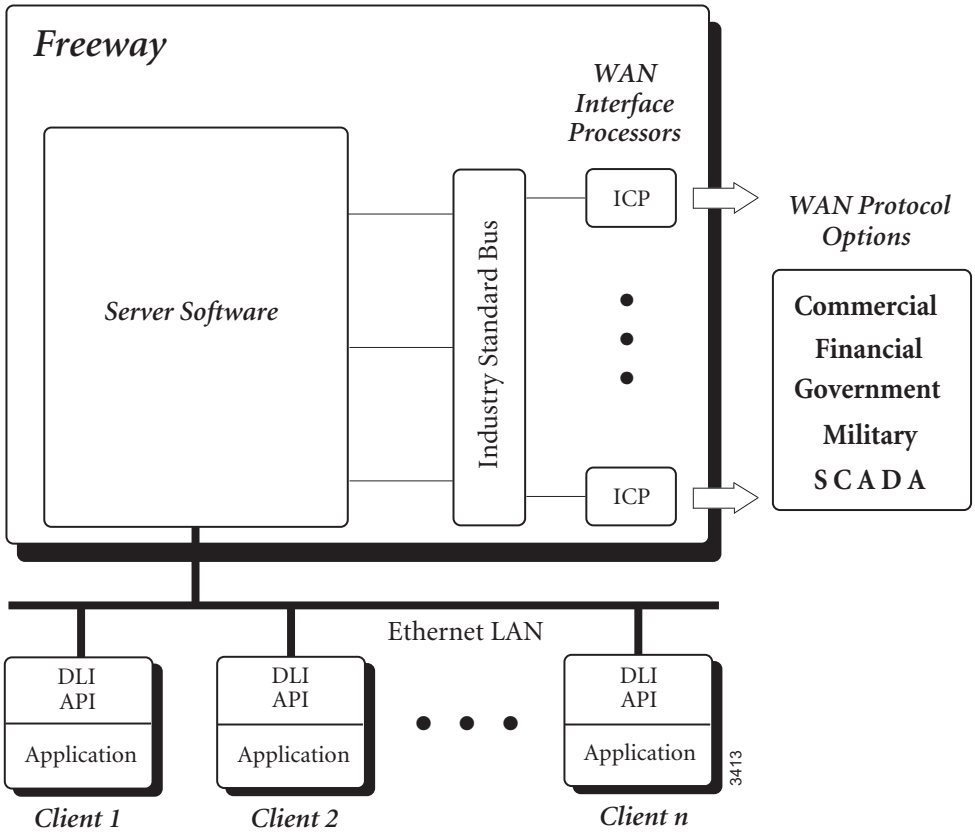

Figure 1–1: Freeway Configuration

1.1.2 Embedded ICP

The embedded ICP connects your client computer directly to the WAN (for example, using Protogate's ICP2432B PCIbus board). The embedded ICP provides client applications with the same WAN connectivity as the Freeway server, using the same data link interface (via the DLITE embedded interface). The ICP runs the communication protocol software using Protogate's real-time operating system. Figure 1–2 shows the embedded ICP configuration.

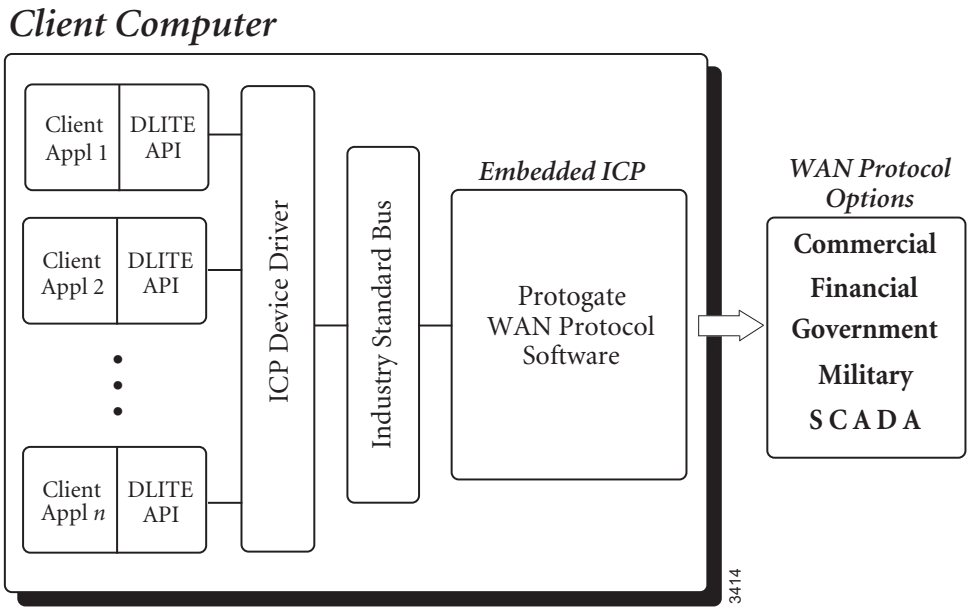

Figure 1–2: Embedded ICP Configuration

Summary of product features:

- Provision of WAN connectivity either through a LAN-based Freeway server or directly using an embedded ICP
- Elimination of difficult LAN and WAN programming and systems integration by providing a powerful and consistent data link interface
- Variety of off-the-shelf communication protocols available from Protogate which are independent of the client operating system and hardware platform
- Support for multiple WAN communication protocols simultaneously
- Support for multiple ICPs (two, four, or eight communication lines per ICP)
- Wide selection of electrical interfaces including EIA-232, EIA-449, EIA-530, and V.35
- Creation of customized server-resident and ICP-resident software, using Protogate's software development toolkits
- Freeway server standard support for Ethernet and Fast Ethernet LANs running the transmission control protocol/internet protocol (TCP/IP)
- Freeway server management and performance monitoring with the simple network management protocol (SNMP), as well as interactive menus available through a local console, telnet, or rlogin

1.2 Freeway Client-Server Environment

The Freeway server acts as a gateway that connects a client on a local-area network to a wide-area network. Through Freeway, a client application can exchange data with a remote data link application. Your client application must interact with the Freeway server and its resident ICPs before exchanging data with the remote data link application.

One of the major Freeway server components is the message multiplexor (MsgMux) that manages the data traffic between the LAN and the WAN environments. The client application typically interacts with the Freeway MsgMux through a TCP/IP BSD-style socket interface (or a shared-memory interface if it is a server-resident application (SRA)). The ICPs interact with the MsgMux through the DMA and/or shared-memory interface of the industry-standard bus to exchange WAN data. From the client application's point of view, these complexities are handled through a simple and consistent data link interface (DLI), which provides dlOpen, dlWrite, dlRead, and dlClose functions.

Figure 1–3 shows a typical Freeway connected to a locally attached client by a TCP/IP network across an Ethernet LAN interface. Running a client application in the Freeway client-server environment requires the basic steps described in Section 1.2.1 and Section 1.4.

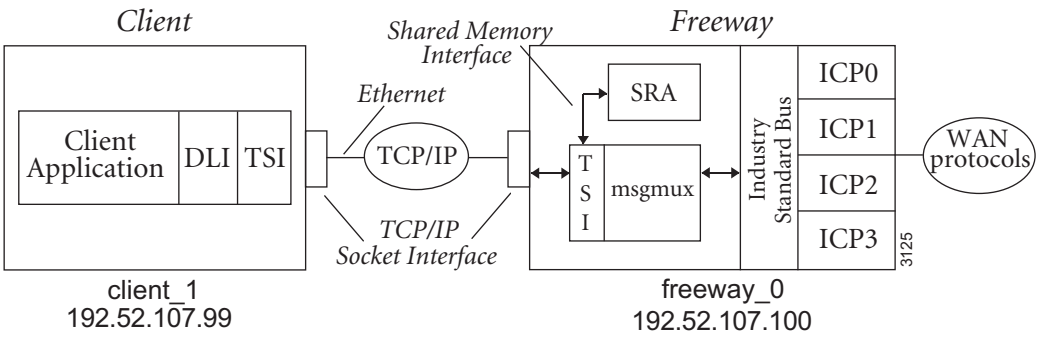

Figure 1–3: A Typical Freeway Server Environment

1.2.1 Establishing Freeway Server Internet Addresses

The Freeway server must be addressable in order for a client application to communicate with it. In the Figure 1–3 example, the TCP/IP Freeway server name is freeway2, and its unique Internet address is 192.52.107.100. The client machine where the client application resides is client1, and its unique Internet address is 192.52.107.99. Refer to the *Freeway User's Guide* to initially set up your Freeway and download the operating system, server, and protocol software.

1.3 Embedded ICP Environment

Refer to the user guide for your embedded ICP and operating system (for example, the *Freeway Embedded ICP2432 User's Guide for Windows NT*) for software installation and setup instructions. The user guide also gives additional information regarding the data link interface for embedded ICP boards (DLITE) and other embedded programming interface descriptions for your specific embedded environment. Refer back to Figure 1–2 on page 21 for a diagram of the embedded ICP environment. Running a client application in the embedded ICP environment requires the basic steps described in Section 1.4

1.4 Client Operations

1.4.1 Defining the DLI and TSI Configuration

In order for your client application to communicate with the ICP's protocol software, you must define the DLI sessions and the transport subsystem interface (TSI) connections between your client application and Freeway (or an embedded ICP). To accomplish this, you first define the configuration parameters in DLI and TSI ASCII configuration files, and then you run two preprocessor programs, dlicfg and tsicfg, to create binary configuration files (see Chapter 4). The dlInit function uses the binary configuration files to initialize the DLI environment.

1.4.2 Opening a Session

After the DLI and TSI configurations are properly defined, your client application uses the dlOpen function to establish a DLI session with an ICP link. As part of the session establishment process, the DLI establishes a TSI connection with the Freeway MsgMux through the TCP/IP BSD-style socket interface for the Freeway server, or directly to the ICP driver for the embedded ICP environment.

1.4.3 Exchanging Data with the Remote Application

After the link is enabled, the client application can exchange data with the remote application using the dlWrite and dlRead functions.

1.4.4 Closing a Session

When your application finishes exchanging data with the remote application, it calls the dlClose function to disable the ICP link, close the session with the ICP, and disconnect from the Freeway server or the embedded ICP driver.

1.5 AWS Overview

Protogate's Asynchronous Wire Service (AWS) product enables the ICP to receive and transmit asynchronous character formats. Link-level configuration options enable the software to synchronize structured message formats. Each link operates independently of the other links on the same ICP and can be configured with different communication options. A client program exchanges formatted messages with the ICP to control the communication links. Each link may be connected to any data communications equipment (DCE) or data terminal equipment (DTE) that is compatible with its configuration. The following serial link characteristics are supported:

- Asynchronous operation to 115,200 bits/second
- Isochronous operation to 115,200 bits/second

- Supports American Newspaper Publishers Association (ANPA) guidelines for 5-bit, 6-bit, 7-bit, and 8-bit wire service transmission.
- EIA-232, EIA-449, EIA-530, or V.35 electrical interface

The communications software on Freeway handles the low-level protocol interface requirements, thus freeing clients from CPU-intensive activity. The AWS software presents packets of data to your application program through the Freeway DLI.

1.5.1 Software Description

Protogate's AWS product includes the following major software components:

- A group of communications software downloadable images:
    1. Freeway server or embedded ICP software
    2. Real-time operating system (OS/Impact)
    3. AWS communications software
- DLI library for linking with client applications
- A loopback test program (awsalp.c) to check the product installation

The *Freeway User Guide* or the user guide for your particular embedded ICP and operating system (for example, the *ICP2432 User Guide for Windows NT*) describes the software installation procedures. The DLI provides an interface by which data is exchanged between the client application and Freeway; refer to the *Freeway Data Link Interface Reference Guide*.

1.5.2 Hardware Description

The configuration of Protogate's AWS product requires the following hardware:

- Freeway communications server (for example, Freeway 3112, 3212, 3412, or 3612) or an embedded ICP (for example the PCIbus ICP2432B or the PCIe bus ICP2432e)
- Ethernet connection to a client running TCP/IP (for a Freeway server)

Chapter

2 AWS DLI Functions

Note

In this document, the term "Freeway" can mean either a Freeway server or an embedded ICP. For the embedded ICP, also refer to the user guide for your ICP and operating system (for example, the *ICP2432 User Guide for Windows NT*).

This chapter describes how to use the data link interface (DLI) functions to write client applications interfacing to the Freeway AWS protocol software. You should be familiar with the concepts described in the *Freeway Data Link Interface Reference Guide*; however, some summary information is provided in Section 2.1.

If you are using an embedded ICP, you must also refer to the user guide for your specific ICP and operating system regarding the embedded DLI interface (referred to as DLITE).

The following might be helpful references while reading this chapter:

- Section 2.2 compares a typical sequence of DLI function calls using blocking versus non-blocking I/O.
- Appendix A explains error handling and provides a summary table for AWS error codes. The *Freeway Data Link Interface Reference Guide* gives complete DLI error code descriptions.

- The *Freeway Data Link Interface Reference Guide* shows a generic code example which can guide your application program development. The loopback test program (awsalp.c) distributed with the product software is another example.
- Appendix B provides detailed command and response header formats.
- The various mnemonic codes mentioned throughout this document are defined in the include files provided with this product, which are described in Table 2–1.

Table 2–1: Include Files

Description	Include File
DLI_PROT_* Codes	dliprot.h
DLI_ICP_ERR_* Codes	dlicperr.h
DLI_ICP_CMD_* Codes	dliicp.h
FW_* Codes	freeway.h

2.1 Summary of DLI Concepts

The DLI presents a consistent, high-level, common interface across multiple clients, operating systems, and transport services. It implements functions that permit your application to use data link services to access, configure, establish and terminate sessions, and transfer data across multiple data link protocols. The DLI concepts are described in detail in the *Freeway Data Link Interface Reference Guide*. This section summarizes the basic information.

2.1.1 Configuration in the Freeway Environment

Several items must be configured before a client application can run in the Freeway environment:

- Freeway server configuration
- data link interface (DLI) session configuration

- transport subsystem interface (TSI) connection configuration
- protocol-specific ICP link configuration

The Freeway server is normally configured only once, during the installation procedures described in the *Freeway User Guide*. DLI session and TSI connection configurations are defined by modifying text configuration files and running the dlicfg and tsicfg preprocessor programs. Refer to Chapter 4 of this document, as well as the *Freeway Data Link Interface Reference Guide* and the *Freeway Transport Subsystem Interface Reference Guide.*

ICP link configuration can be performed using any of the following methods:

- The dlOpen function can configure the ICP links during the DLI session establishment process using the default ICP link configuration values provided by the protocol software.
- You can specify ICP link parameters in the DLI text configuration file and then run the dlicfg preprocessor program (Chapter 4). The dlOpen function uses the resulting DLI binary configuration file to perform the link configuration during the DLI session establishment process.
- You can perform ICP link configuration within the client application (described in Section 2.4.1.2). This method is useful if you need to change link configuration without exiting the application.

2.1.2 Normal versus Raw Operation

There are two choices for the protocol DLI configuration parameter:

- A session is opened for *Normal* operation if you set protocol to a specific protocol (for example, "AWS"); then the DLI software configures the ICP links using the values in the DLI configuration file and transparently handles all headers and I/O.

- A session is opened for *Raw* operation if you set protocol to "raw"; then your application must handle all configuration, headers, and I/O details. Refer to the *Freeway Data Link Interface Reference Guide* if you need to use *Raw* operation.

Note

The embedded DLITE interface requires *Raw* operation. Refer to the user guide for your ICP and operating system; for example, the *ICP2432 User Guide for Windows NT*.

To read and write using *Normal* operation, your client application typically interacts with Freeway only for the purpose of exchanging data with the remote application.

However, if your client application needs to interact with Freeway to obtain status or reports, or to provide Freeway with protocol-specific information relating to the data exchange, *Normal* and *Raw* operation can be mixed. The client application session should be configured for *Normal* operation (allowing DLI to handle some of the headers), but the write and read requests (Section 2.4 and Section 2.5) can use *Raw* operation by including the optional arguments structure (Section 2.3) containing the protocol-specific information.

2.1.3 Blocking versus Non-blocking I/O

Note

Earlier Freeway releases used the term "synchronous" for blocking I/O and "asynchronous" for non-blocking I/O. Some parameter names reflect the previous terminology.

Non-blocking I/O applications are useful when doing I/O to multiple channels with a single process where it is not possible to "block" on any one channel waiting for I/O completion. Blocking I/O applications are useful when it is reasonable to have the calling process wait for I/O completion.

In the Freeway environment, the term blocking I/O indicates that the dlOpen, dlClose, dlRead and dlWrite functions do not return until the I/O is complete. For non-blocking I/O, these functions might return after the I/O has been queued at the client, but before the transfer to Freeway is complete. The client must handle I/O completions at the software interrupt level in the completion handler established by the dlInit or dlOpen function, or by periodic use of dlPoll to query the I/O completion status.

The asyncIO DLI configuration parameter specifies whether an application session uses blocking or non-blocking I/O. The alwaysQIO DLI configuration parameter further qualifies the operation of non-blocking I/O activity. Refer to the *Freeway Data Link Interface Reference Guide* for more information.

The effects on different DLI functions, resulting from the choice of blocking or non-blocking I/O, are explained in the *Freeway Data Link Interface Reference Guide* and throughout this chapter as they relate to AWS.

2.1.4 Buffer Management

Currently the interrelated Freeway, DLI, TSI and ICP buffers default to a size of 1024 bytes.

Caution

If you need to change a buffer size for your application, refer to the *Freeway Data Link Interface Reference Guide* for explanations of the complexities that you must consider.

2.2 Example AWS Call Sequences

Table 2–2 shows the sequence of DLI function calls to send and receive data using blocking I/O. Table 2–3 is the non-blocking I/O example. The remainder of this chapter and the *Freeway Data Link Interface Reference Guide* give further information about each function call.

Note

The example call sequences assume that the cfgLink and enable DLI configuration parameters are both set to "yes" (the defaults). This means that dlOpen configures and enables the ICP links. Figure 4–2 on page 81 shows an example DLI configuration file.

Table 2–2: DLI Call Sequence for AWS (Blocking I/O)

1. Call dlInit to initialize the DLI operating environment. The first parameter is your DLI binary configuration file name.
2. Call dlOpen for each required session (link) to get a session ID.
3. Call dlBufAlloc for all required input and output buffers.
4. Call dlWrite to send requests and data to Freeway (Section 2.4 on page 39).
5. Call dlRead to receive responses and data from Freeway (Section 2.5 on page 50).
6. Repeat Step 4 and Step 5 until you are finished writing and reading.
7. Call dlBufFree for all buffers allocated in Step 3.
8. Call dlClose for each session ID obtained in Step 2.
9. Call dlTerm to terminate your application's access to Freeway.

Table 2–3: DLI Call Sequence for AWS (Non-blocking I/O)

1. Call dlInit to initialize the DLI operating environment. The first parameter is your DLI binary configuration file name.
2. Call dlOpen for each required session (link) to get a session ID.
3. Call dlPoll to confirm the success of each session ID obtained in Step 2.
4. Call dlBufAlloc for all required input and output buffers.
5. Call dlRead to queue the initial read request.
6. Call dlWrite to send requests and data to Freeway (Section 2.4 on page 39).
7. Call dlRead to queue reads to receive responses and data from Freeway (Section 2.5 on page 50).
8. As I/Os complete and the I/O completion handler is invoked, call dlPoll to confirm the success of each dlWrite in Step 6 and to accept the data from each dlRead in Step 7.
9. Repeat Step 6 through Step 8 until you are finished writing and reading.
10. Call dlBufFree for all buffers allocated in Step 4.
11. Call dlClose for each session ID obtained in Step 2.
12. Call dlPoll to confirm that each session was closed in Step 11.
13. Call dlTerm to terminate your application's access to Freeway.

2.3 Overview of DLI Functions for AWS

This section summarizes the DLI functions used in writing a client application. An overview of using the DLI functions is:

- Start up communications (dlInit, dlOpen, dlBufAlloc)
- Send requests and data using dlWrite
- Receive responses using dlRead
- For blocking I/O, use dlSyncSelect to query read availability status for multiple sessions
- For non-blocking I/O, handle I/O completions at the software interrupt level in the completion handler established by the dlInit or dlOpen function, or by periodic use of dlPoll to query the I/O completion status
- Monitor errors using dlpErrString
- If necessary, reset and download the protocol software to the ICP using dlControl
- Shut down communications (dlBufFree, dlClose, dlTerm)

Table 2–4 summarizes the DLI function syntax and parameters, listed in the most likely calling order. Refer to the *Freeway Data Link Interface Reference Guide* for details.

Table 2–4: DLI Functions: Syntax and Parameters (Listed in Typical Call Order)

DLI Function	Parameter(s)	Parameter Usage
int dlInit	(char *cfgFile, char *pUsrCb, int (*fUsrIOCH)(char *pUsrCb));	DLI binary configuration file name Optional I/O complete control block Optional IOCH and parameter
int dlOpen[a]	(char *cSessionName, int (*fUsrIOCH) (char *pUsrCB, int iSessionID));	Session name in DLI config file Optional I/O completion handler Parameters for IOCH
int dlPoll	(int iSessionID, int iPollType, char **ppBuf, int *piBufLen, char *pStat, DLI_OPT_ARGS **ppOptArgs);	Session ID from dlOpen Request type Poll type dependent buffer Size of I/O buffer (bytes) Status or configuration buffer Optional arguments
int dlpErrString	(int dlErrNo);	DLI error number (global variable dlerrno)
char *dlBufAlloc	(int iBufLen);	Minimum buffer size
int dlRead	(int iSessionID, char **ppBuf, int iBufLen, DLI_OPT_ARGS *pOptArgs);	Session ID from dlOpen Buffer to receive data Maximum bytes to be returned Optional arguments structure
int dlWrite	(int iSessionID, char *pBuf, int iBufLen, int iWritePriority, DLI_OPT_ARGS *pOptArgs);	Session ID from dlOpen Source buffer for write Number of bytes to write Write priority (normal or expedite) Optional arguments structure
int dlSyncSelect	(int iNbrSessID, int sessIDArray[], int readStatArray[]);	Number of session IDs Packed array of session IDs Array containing read status for IDs
char *dlBufFree	(char *pBuf);	Buffer to return to pool
int dlClose	(int iSessionID, int iCloseMode);	Session ID from dlOpen Mode (normal or force)
int dlTerm	(void);	
int dlControl	(char *cSessionName, int iCommand, int (*fUsrIOCH) (char *pUsrCB, int iSessionID));	Session name in DLI config file Command (*e.g.* reset/download) Optional I/O completion handler Parameters for IOCH

[a] It is critical for the client application to receive the dlOpen completion status before making any other DLI requests; otherwise, subsequent requests will fail. After the dlOpen completion, however, you do not have to maintain a one-to-one correspondence between DLI requests and dlRead requests.

2.3.1 DLI Optional Arguments

Section 2.4 and Section 2.5 describe the dlWrite and dlRead functions for an AWS application. Both functions can use the optional arguments parameter to provide the protocol-specific information required for *Raw* operation (Section 2.1.2). The "C" definition of the optional arguments structure is shown in Figure 2–1.

```
typedef struct      _DLI_OPT_ARGS
{
    unsigned short  usFWPacketType;  /* FW_CONTROL or FW_DATA                */
    unsigned short  usFWCommand;     /* FW_ICP_WRITE, FW_ICP_WRITE_EXP       */
                                     /*     or FW_ICP_READ                   */
    unsigned short  usFWStatus;
    unsigned short  usICPClientID;
    unsigned short  usICPServerID;
    unsigned short  usICPCommand;    /* Required for start/stop cmds         */
    short           iICPStatus;      /* ICP return error code (dlRead)       */
    unsigned short  usICPParms[3];
    unsigned short  usProtCommand;   /* Required field (dlWrite)             */
    short           iProtModifier;
    unsigned short  usProtLinkID;
    unsigned short  usProtCircuitID; /* Used for translation tables          */
    unsigned short  usProtSessionID;
    unsigned short  usProtSequence;
    unsigned short  usProtXParms[2];
}   DLI_OPT_ARGS;
```

Figure 2–1: "C" Definition of DLI Optional Arguments Structure

2.4 Overview of AWS Requests using dlWrite

For AWS the dlWrite function supports three dlWrite categories: commands, information requests, and data transfer, which are discussed in detail in Section 2.4.1 through Section 2.4.3. Whether you use blocking or non-blocking I/O, each dlWrite request must be followed by a dlRead request to receive the command confirmation, information requested, or acknowledgment of the data transfer. Section 2.5 discusses these different responses received using dlRead.

In a typical AWS application, most dlWrite requests use *Raw* operation; that is, the optional arguments structure (page 38) is required to specify protocol-specific information.

If your application is limited to interacting with Freeway only to exchange data with the remote application, you can read and write using *Normal* operation.

Caution

Take care not to confuse the terms "*Normal* operation" and "normal data." Normal data can be sent using either *Normal* or *Raw* operation.

Table 2–5 shows the AWS DLI request codes for different categories of the dlWrite function. Each request is explained in the following sections.

An unsuccessful dlWrite function can return one of the following error codes in the dlRead pOptArgs.iICPStatus field (see Appendix A for error handling):

DLI_ICP_ERR_INBUF_OVERFLOW	Input buffer overflow
DLI_ICP_ERR_OUTBUF_OVERFLOW	Output buffer overflow

Table 2–5: Categories for AWS dlWrite Requests

Category	DLI Request Codes	Usage	Reference Section
Commands to ICP	DLI_ICP_CMD_BIND	Enable ICP link	Section 2.4.1.3
	DLI_ICP_CMD_UNBIND	Disable ICP link	Section 2.4.1.4
	DLI_PROT_CFG_LINK	Configure ICP link	Section 2.4.1.2
	DLI_PROT_CLR_STATISTICS	Clear ICP link statistics	Section 2.4.1.6
	DLI_PROT_SEND_CRYPTO_RESYNC	Send crypto Resync pulse	Section 2.4.1.7
	DLI_PROT_SET_BUF_SIZE	Set ICP message buffer size	Section 2.4.1.1
	DLI_PROT_SET_SIG	Set/Clear discrete signals	Section 2.4.1.5
Report Requests	DLI_PROT_GET_LINK_CFG	Request ICP link configuration report	Section 2.4.2.1
	DLI_PROT_GET_ICP_REPORT	Request ICP configuration report	Section 2.4.2.2
	DLI_PROT_GET_SIG_REPORT	Request discrete signals status report	Section 2.4.2.3
	DLI_PROT_GET_STATISTICS	Request statistics report	Section 2.4.2.4
Data Transfer	DLI_PROT_SEND_NORM_DATA	Transmit normal data	Section 2.4.3.1

2.4.1 Commands using Raw dlWrite

Section 2.4.1.1 through Section 2.4.1.6 explain how to issue specific commands to the AWS software using the dlWrite function. Call dlRead to receive the command confirmation response (the dlRead pOptArgs.usProtCommand field will be set by the DLI).

2.4.1.1 Set ICP Message Buffer Size Command

The ICP message buffer size applies to all links on the ICP. The DLI sets the ICP message buffer size as part of the configuration process during dlOpen. The default ICP message buffer size of 1024 (or the bufferSize parameter value, page 83, specified in the DLI configuration file) is used in the following situations:

- If the buffer size is not changed after the very first dlOpen is issued, immediately after the AWS software is downloaded. (The dlOpen function uses the value specified in the bufferSize parameter, page 83).

- If you specify an invalid buffer size for the Set ICP Message Buffer Size command

If your application must set the ICP message buffer size itself (see the caution previously mentioned in Section 2.1.4), it must set the cfgLink and enable DLI configuration parameters to "no" (Section 4.2 on page 80) and perform the following procedure:

First, download the AWS software and send a dlOpen request for one link on the ICP. Second, send a dlWrite function with the pOptArgs.usProtCommand field set to DLI_PROT_SET_BUF_SIZE. Set the iBufLen parameter to 2, and set the write buffer to the maximum data size (in bytes) for any single transfer to or from the ICP. The valid range is 64 to 8192 bytes and must be less than or equal to the maxBufSize parameter in the TSI configuration file (the default value is 1024).

2.4.1.2 Configure ICP Link Command

You can define the ICP link configuration by setting parameters in the DLI text configuration file and running the dlicfg preprocessor program as described in Chapter 4;

however, if your application must perform link configuration, set both the cfgLink and enable DLI configuration parameters to "no" for that link and then perform the configuration as follows:

Use the dlWrite function with the pOptArgs.usProtCommand field set to DLI_PROT_CFG_LINK to set the link configuration options. The buffer pointed to by the pBuf parameter contains a configuration option/value pair for each configuration parameter. The first 32-bit option word contains the configuration option. The second 32-bit option word contains the configuration value. The string of configuration pairs is terminated by a 32-bit word containing zero. The iBufLen field equals the number of bytes in the configuration string including the zero terminator word. Figure 2–2 gives an example of the link configuration string.

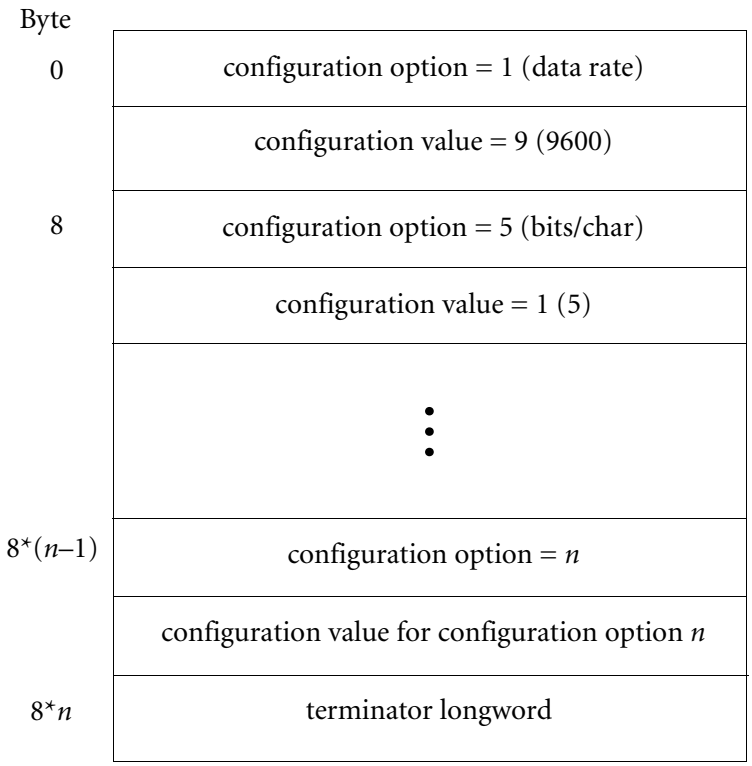

Figure 2–2: Link Configuration Report Format

Each option number corresponds to a software-selectable option of the AWS software. The configuration value is used to set that option. Table 3–1 on page 56 lists the available options and settings for the AWS protocol.

2.4.1.3 Enable ICP Link Command

The bind link command activates (enables) the ICP link. The DLI enables the link for you during the dlOpen command; however, if your application must perform a bind itself, it must set the enable parameter to "no" and perform the following:

Use the dlWrite function with the pOptArgs.usProtCommand field set to DLI_ICP_CMD_BIND and the pOptArgs.usICPCommand field set to DLI_ICP_CMD_BIND to send the bind link command. Set the iBufLen parameter to zero. When the ICP receives this message, it allocates receive buffers and activates the link.

2.4.1.4 Disable ICP Link Command

The unbind link command deactivates (disables) the ICP link. The DLI disables the link for you during the dlClose command; however, if your application must perform an unbind itself, it must set the enable parameter to "no" and perform the following:

Use the dlWrite function with the pOptArgs.usProtCommand field set to DLI_ICP_CMD_UNBIND and the pOptArgs.usICPCommand field set to DLI_ICP_CMD_UNBIND to send the unbind link command. Set the iBufLen parameter to zero. When the ICP receives this message, it aborts serial communications and releases all buffers in use by the link.

A call to dlClose (described in the *Freeway Data Link Interface Reference Guide*) also stops the link, but terminates the session as well. This is the normal method to terminate a session at the end of a client application. The Disable ICP Link command is useful for temporarily stopping the link without terminating the session (for example, to reconfigure the link using the Configure ICP Link command). The link is restarted by issuing the Enable ICP Link command.

2.4.1.5 Set/Clear Discrete Line(s) Command

The set/clear discrete line command asserts or deasserts the Data Terminal Ready (DTR) and Request To Send (RTS) discrete lines (see Figure 2–3).

Use the dlWrite function with the pOptArgs.usProtCommand field set DLI_PROT_SET_SIG to set or clear DTR and RTS for the link. Set the iBufLen parameter to 2, and set the write buffer to the 16-bit word describing the action to be taken. Set the corresponding bit in the lower byte of the write buffer command word to assert the DTR or RTS discrete lines. Set the corresponding bit in the upper byte of the write buffer command word to deassert the lines. The DTR value is two (bit one), and the RTS value is 16 (bit four).

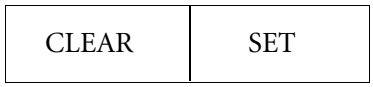

Figure 2–3: Set/Clear Discrete Line(s) Word

2.4.1.6 Clear Link Statistics Command

The clear link statistics command clears the link statistics that are obtained using the DLI_PROT_GET_STATISTICS request (Section 2.4.2.4 on page 47).

Use the dlWrite function with the pOptArgs.usProtCommand field set to DLI_PROT_CLR_STATISTICS. The function returns the same code (DLI_PROT_CLR_STATISTICS). If successful the pOptArgs.iProtModifier field is 0. If unsuccessful the pOptArgs.iProtModifier field is 32768.

2.4.1.7 Send Crypto Resync Command

The send crypto resync command requests the link to issue a crypto resync pulse. This pulse is effected by raising the Data Terminal Ready modem signal for the number of milliseconds specified by the link's Crypto resync configuration option (Section 3.17 on

page 68). The crypto resync is not performed until the current message transmission, if any, is completed.

Use the dlWrite function with the pOptArgs.usProtCommand field set to DLI_PROT_SEND_CRYPTO_RESYNC. The ICP does not issue a response.

2.4.2 Information Requests using Raw dlWrite

Section 2.4.2.1 through Section 2.4.2.4 explain how to issue specific information requests to the AWS software using the dlWrite function. You must then make a *Raw* dlRead request to receive the report information (the dlRead pOptArgs.usProtCommand field will be set by the DLI to reflect the type of report, and the iBufLen parameter indicates the size of the message).

Caution

The dlWrite iBufLen parameter must specify a buffer size large enough for the requested report; otherwise, the dlRead function truncates the text to the size indicated by the dlWrite iBufLen parameter.

2.4.2.1 Request ICP Link Configuration Report

Use the dlWrite function with the pOptArgs.usProtCommand field set to DLI_PROT_GET_LINK_CFG to request the current configuration option settings for a link. The dlRead configuration report response (the pOptArgs.usProtCommand field will be set to DLI_PROT_GET_LINK_CFG by the DLI) consists of a sequence of configuration option/value pairs for each configuration parameter pointed to by the pBuf parameter. The first 32-bit option word contains the configuration option. The second 32-bit value word contains the configuration option value. The report format is identical to that used by the dlWrite Configure Link command (Section 2.4.1.2).

2.4.2.2 Request ICP Configuration Report

When the ICP receives this command, it returns the ICP configuration report using the same function code (DLI_PROT_GET_ICP_REPORT) and a data size field of 10. The number of ICP links field can be used to determine how many serial links are available. The returned format is shown in Figure 2–4.

Word	
0	Buffer size (in bytes)
2	Total number of buffers in ICP
4	Number of buffers in use
6	Number of buffers in free queue
8	Number of ICP links

Figure 2–4: ICP Configuration Report Form

2.4.2.3 Request Discrete Signals Status Report

When the ICP receives this command, it returns a discrete status report using the same function code (DLI_PROT_GET_SIG_REPORT). The report consists of an ICP header that has a data size field of one. The data area field contains the discrete status.

The discrete line is asserted if the corresponding bit is set in the report byte. The Clear To Send (CTS) value is 16 (bit four) and the Data Carrier Detect (DCD) value is 32 (bit five).

2.4.2.4 Request Statistics Report

When the ICP receives this command, it returns a statistics report using the same function code (DLI_PROT_GET_STATISTICS). The report consists of 32 bytes (16 words) of statistics. The data area field contains the statistics report shown in Table 2–6.

Table 2–6: Statistics Report Format

Word	Description
0	DCD up
1	CTS up
2	Break errors
3	Receive overrun errors
4	Block check errors
5	Parity errors
6	Framing errors
7	Transmit underruns
8–9	Characters sent
10–11	Characters received
12–13	Frames sent
14–15	Frames received

2.4.3 Data Transfer using Raw dlWrite

If your application must send data packets using *Raw* operation or must process the DLI_PROT_RESP_LOCAL_ACK packet, you need to understand this section.

If the DLI localAck configuration parameter is set to "no" (see the *Freeway Data Link Interface Reference Guide*), the client application must make a dlRead request to receive the data acknowledge response for each dlWrite data transfer request (the pOptArgs.usProtCommand field will be set to DLI_PROT_RESP_LOCAL_ACK by the DLI). One DLI_PROT_RESP_LOCAL_ACK response is sent to the client computer after each data block has been successfully transmitted to the remote computer, and can be treated by the client program as the remote acknowledgment. Data acknowledgments are also used to report transmission errors (as a response to a dlRead request as described in Section 2.5.2). The client application can use the data acknowledge response to monitor the success or failure of transmitted data messages, or for regulating the number of outbound messages that the AWS software has pending transmission at any one time.

If the DLI localAck configuration parameter is set to "yes" (which is the default), the data acknowledge response is implied by a successful dlWrite.

2.4.3.1 Send Normal Data

If your application needs to perform a *Raw* dlWrite, use the dlWrite function with the pOptArgs.usProtCommand field set to DLI_PROT_SEND_NORM_DATA to transmit a data message.

2.5 Overview of AWS Responses using Raw dlRead

Table 2–7 shows the valid AWS codes sent to your application in response to a *Raw* dlRead request; the returned dlRead pOptArgs.usProtCommand field indicates the response code. If the dlRead return value is zero or positive, it indicates the number of bytes read; if it is less than zero, an error has occurred. AWS error codes associated with the responses are returned in the pOptArgs.iICPStatus field or the pOptArgs.iProtModifier field (see Appendix A).

Note

The use of *Normal* dlRead requests (that is, without the optional arguments parameter) is not recommended for AWS since reports and acknowledgments would be indistinguishable from data received from the remote application.

The following types of data can be returned from the ICP:

- Normal received data
- Acknowledgments (if the localAck parameter is set to "no")
- Command confirmation responses
- Reports in response to dlWrite information requests

2.5.1 Received Data

2.5.1.1 Received Data Response

The received data response (the DLI sets the dlRead pOptArgs.usProtCommand field to DLI_PROT_RECV_DATA) signifies that the ICP has generated a message with data from a serial link or status report data. The buffer pointed to by the pBuf parameter contains data associated with the message, and dlRead returns the number of bytes contained in the message. You must specify the maximum size required (in bytes) for the data area field when you configure the ICP buffers.

Table 2–7: AWS Response Codes

Category	DLI Response Code	Usage	Section
Received Data	DLI_PROT_RECV_DATA	Normal received data	Section 2.5.1.1
	DLI_PROT_RECV_DATA_LOST	Data lost – client not reading	Section 2.5.1.2
Acknowledgments	DLI_ICP_CMD_BIND	Enable ICP link acknowledge	Section 2.5.2.2
	DLI_ICP_CMD_UNBIND	Disable ICP link acknowledge	Section 2.5.2.3
	DLI_PROT_RESP_LOCAL_ACK	Acknowledgment that a transmission buffer has been sent (if the pOptArgs.iICPStatus parameter = 1). If it is less than zero, an error has occurred.	Section 2.5.2.1
	DLI_PROT_RESP_NACK	Transmit negative acknowledge	Section 2.5.2.4
Command Confirmations	DLI_PROT_SET_BUF_SIZE	Set ICP Message Buffer Size confirmation	Section 2.4.1.1
	DLI_PROT_CFG_LINK	Configure link confirmation	Section 2.4.1.2
	DLI_PROT_CLR_STATISTICS	Clear statistics confirmation	Section 2.4.1.6
Reports	DLI_PROT_GET_LINK_CFG_REP	ICP link configuration report	Section 2.4.2.1
	DLI_PROT_GET_ICP_REPORT	ICP configuration report	Section 2.4.2.2
	DLI_PROT_GET_SIG_REPORT	Discrete signals status report	Section 2.4.2.3
	DLI_PROT_GET_STATISTICS	Statistics report	Section 2.4.2.4

2.5.1.2 Data Lost Response

The data lost response (the DLI sets the dlRead pOptArgs.usProtCommand field to DLI_PROT_RECV_DATA_LOST) signifies that the client is not reading queued receive messages at a rate compatible with the incoming data. To avoid a buffer depletion problem, the ICP discards received messages when the client queue is not serviced. The dlRead function returns the number of bytes in the message (the buffer pointed to by the pBuf parameter), and pOptArgs.usProtLinkID is set to the link that received the data.

2.5.2 Acknowledgments

Table 2–7 on page 51 lists the possible acknowledgment codes returned in the dlRead pOptArgs.usProtCommand field. All of the responses can also return an error code in the dlRead pOptArgs.iICPStatus field to indicate failure.

2.5.2.1 Transmit Acknowledge Response

The transmit acknowledge response (the DLI sets the dlRead pOptArgs.usProtCommand field to DLI_PROT_RESP_LOCAL_ACK) signifies that a transmit message completed successfully. The dlRead function returns the number of bytes successfully transmitted.

2.5.2.2 Bind Link Acknowledge Response

The bind link acknowledge response (the DLI sets the dlRead pOptArgs.usProtCommand field to DLI_ICP_CMD_BIND) signifies that a link was successfully enabled. The dlRead function returns a data length of zero.

2.5.2.3 Unbind Link Acknowledge Response

The unbind link acknowledge response (the DLI sets the dlRead pOptArgs.usProtCommand field to DLI_ICP_CMD_UNBIND) signifies that a link was successfully disabled. The dlRead function returns a data length of zero.

2.5.2.4 Transmit Negative Acknowledge Response

The transmit negative acknowledge response (the DLI sets the dlRead pOptArgs.usProtCommand field to DLI_PROT_RESP_NACK) signifies that the transmission was aborted (or never begun). The reason is specified in the pOptArgs.iProtModifier field (see Table A–2 on page 87). Transmit-related reasons include CTS Timeout, Transmit Expiration, and Secure Mode Violation. When any message is negatively acknowledged, then all subsequent messages currently queued for transmission are also negatively acknowledged (allowing the client to re-queue them in the originally intended order of transmission). The dlRead function returns the number of characters (if any) sent on

the serial line before the transmission was aborted (this will be 0 when the transmission was never started).

2.5.3 Command Confirmations

Table 2–7 on page 51 lists the possible command confirmation codes returned in the dlRead pOptArgs.usProtCommand field. These responses can return an error code in the dlRead pOptArgs.iICPStatus field to indicate failure.

2.5.4 Reports in Response to dlWrite Information Requests

After issuing a dlWrite information request (Section 2.4.2 on page 46), a dlRead request must be issued to receive the report information. These ICP reports can help diagnose software problems. The reports are listed in Table 2–7 on page 51.

Note

Rejected Commands: When the client application sends an illegal command (for example, an invalid command code), the ICP sends the command back with the rejected message bit (bit 15) set in the pOptArgs.iProtModifier field.

Chapter

3 AWS Link Configuration Options

This chapter describes the various link configuration options that can be set using the DLI configuration file described in Section 4.2 on page 80. Alternatively, the link options can be set using the dlWrite Configure Link command as described in Section 2.4.1.2 on page 41.

Table 3–1 lists all the available options in numerical order along with the allowed settings and defaults. The defaults are in effect immediately after the protocol software is downloaded to the ICP. They remain in effect until you either modify the DLI configuration file and redownload the ICP, or send a dlWrite Configure Link command.

If your application uses a DLI binary configuration file containing protocol-specific parameters that differ from the defaults, and you do a normal dlOpen and the cfgLink configuration parameter is set to "yes", then the new values will take effect when your application calls dlOpen. For a description of cfgLink, refer to the *Freeway Data Link Interface Reference Guide*.

Note

Link configuration options can be set only when the link being configured is stopped.

Some of the possible error conditions are discussed in the following sections, as they relate to using each configuration option. Appendix A explains AWS error handling and gives a list of errors.

Table 3–1: AWS Default Options and Settings

Option	Number	Value	Default (✓)	Setting
Standard data rate (b/s)	1	1		50 (asynchronous only)
		2		56 (56.8)
		3		74 (74.2)
		4		300
		5		1050
		6	✓	1200
		7		2400
		8		4800
		9		9600
		10		19200
		11		38400
		12		57600
		13		76800
		14		115200
Clock Source	2	0		External
		1	✓	Internal
Start Count	3	*n*	0	0–4
End Count	4	*n*	0	0–4
Bits/char	5	1		5 bits/char
		2		6 bits/char
		3		7 bits/char
		4	✓	8 bits/char
Parity	6	1	✓	None
		2		Even
		3		Odd
		4		Mark
		5		Space
Stop bits	7	1	✓	1 stop bit
		2		1.5 stop bits
		3		2 stop bits
Transmit Expiration	8	0	0	no transmit expiration
		n		1-32767 milliseconds
Receive Timeout	9	0	6 (300 ms)	Disable
		n		Number of 50-ms increments, where 0 ð *n* < 8192

Table 3–1: AWS Default Options and Settings *(Cont'd)*

Option	Number	Value	Default (✓)	Setting
Queue Limit	10	0		Disable
		n	4	Number of buffers to queue for client before discarding data, where 0 < *n* < 4096
Termination Count	11	*n*	ICP buffer size	Receive buffer termination count, where 0 < *n* < 8192
Message format	12	1	✓	Unformatted
		2		Control character set termination
		3		Control character set framing
		4		Character sequence framing
		5		Character sequence synchronize start
Start-of-message Longword	13	*n*	0x02000000	32-bit mask or 1–4 character sequence, where 0x00 < *n* < 0xFFFFFFFF
End-of-message Longword	14	*n*	0x03000000	32-bit mask or 1–4 character sequence, where 0x00 < *n* < 0xFFFFFFFF
Alternate data rate	15	0	✓	Disable
		n		User-defined time constant to define non-standard data rate, where 0 < *n* < 65535
Secure mode	16	1	✓	Disable
		2		Enable
Crypto resync	17	0	✓	Disable
		n		Crypto pulse length in milliseconds, where 0 < *n* < 0x8000
CTS timeout	22	0	✓	Disable
		n		Number of 50-ms increments, where 0 < *n* < 8192
RTS delay	23	0	✓	Disable
		n		Number of 50-ms increments, where 0 < *n* < 8192
Modem Control	24	1	✓	Disable
		2		Full duplex
		3		Half duplex
Format transmit messages	25	1	✓	Disable
		2		Enable

Table 3–1: AWS Default Options and Settings *(Cont'd)*

Option	Number	Value	Default (✓)	Setting
Receive block check	29	1	✓	None
		2		LRC-8
		3		CRC-16
Receive framing characters	30	1	✓	Disable
		2		Enable
Baudot code	31	1	✓	Disable
		2		Enable
Software handshake	32	1	✓	Disable
		2		XON/XOFF
		3		XON/XANY
Isochronous mode	34	1	✓	Disable
		2		Enable
Electrical Interface (ICP2432 only)	40	0	✓	EIA-232
		1		Reserved
		2		EIA-530
		3		V.35
		4		EIA-449

3.1 Standard Data Rate Option (1)

The data rate can be set by the client for installations where the AWS software must generate the data clocking signal. If external clocking is provided by a modem or modem eliminator, the configuration of the data rate is not required. However, when transmitting data, the data rate setting is used to calculate the amount of time to wait before aborting a transmission. Therefore, if AWS is being used to transmit data, the data rate should be set to match, or be slower than, the modem clock rate.

The data rate on a link can be set from 50 through 115,200 bits/second. When using data rates above 19,200 bits/second, be careful not to overload the communications server processor. Freeway supports up to 16 links per ICP, and the maximum link data rates are:

- 4–8 links at 19,200 b/s
- 16 links for a 16-port ICP at 9,600 b/s

Note

The 50 b/s data rate is valid for asynchronous only.

To set this option using the DLI configuration file, use the dataRate parameter; for example, dataRate = 1200. See Table 4–2 on page 82.

3.2 Clock Source Option (2)

The clock source option determines the source of the data clock signals for a link. For both asynchronous and isochronous operations, data clocking can be provided by the AWS software or received from an external source. Note that the Freeway server clock jumpers must be set to match the software clock selection (except when in normal asynchronous mode using internal clocking; in this case, the jumper settings are not important).

The standard asynchronous mode of transmission/reception uses internal receive and transmit clocking. The default for the clock source option is internal clocking to be compatible with standard asynchronous operation.

To set this option using the DLI configuration file, use the clockSource parameter; for example, clockSource = "internal". See Table 4–2 on page 82.

3.2.1 External

External clocking is usually used only in conjunction with the isochronous mode option (Section 3.26); very few applications require external clocking in asynchronous mode.

When the *external* clock setting is used, the clock generator is disabled, and data clocking must be supplied by an external source such as a modem or modem eliminator.

Receive clocking is input through the receiver timing signal (EIA-232 pin 17), and transmit clocking is input through the transmitter timing signal (EIA-232 pin 15).

3.2.2 Internal

When the *internal* clock setting is used, the clock signal is generated at the rate specified in the data rate option (Section 3.1).

In isochronous mode, the generated clock signal is used for transmit clocking and is output on the terminal timing signal (EIA-232 pin 24). Receive clocking is input through the receiver timing signal (EIA-232 pin 17). The transmitter timing signal (EIA-232 pin 15) is not used.

In asynchronous mode, the generated clock signal is used for both transmit and receive clocking and is not provided on any output signal (because standard asynchronous mode does not use clock lines).

3.3 Start Count Option (3)

The start count option defines the number of characters in the start-of-message sequence for character-oriented formats. The value can specify zero to four characters.

To set this option using the DLI configuration file, use the startCount parameter; for example, startCount = 0. See Table 4–2 on page 82.

3.4 End Count Option (4)

The end count option defines the number of characters in the end-of-message sequence for character-oriented formats. The value can specify zero to four characters.

To set this option using the DLI configuration file, use the endCount parameter; for example, endCount = 0. See Table 4–2 on page 82.

3.5 Bits per Character Option (5)

The bits per character option defines the number of bits per character. The value can specify five, six, seven, or eight bits per character.

To set this option using the DLI configuration file, use the bitsPerChar parameter; for example, bitsPerChar = "8bits". See Table 4–2 on page 82.

3.6 Parity Option (6)

The parity option defines the parity sense. The value can specify *no parity, even, odd, mark*, or *space*.

To set this option using the DLI configuration file, use the parity parameter; for example, parity = "none". See Table 4–2 on page 82.

3.7 Stop Bits Option (7)

The stop bits option defines the number of stop bits per character. The value can specify 1, 1.5, or 2 stop bits.

To set this option using the DLI configuration file, use the stopBits parameter; for example, stopBits = "one". See Table 4–2 on page 82.

3.8 Transmit Expiration Option (8)

The transmit expiration option defines whether the link applies a time limit to the opportunity to transmit messages, and if so, how long this window of opportunity lasts. For a given message, the window begins upon message receipt from the client. If the ICP is unable to begin to transmit the message before the end of this time window, it will reject the message as "stale". The rejection is a Transmit Negative Acknowledgement Response (see Section 2.5.2.4 on page 52). The option value can be in the range 1-65535

(with units of milliseconds). It is set to 0 when no transmit expiration is wanted. The default is 0.

To set this option using the DLI configuration file, use the transmitExpiration parameter; for example, transmitExpiration = 5000. See Table 4–2 on page 82.

3.9 Receive Timeout Option (9)

The receive timeout option defines the maximum time to wait for the next character of an asynchronous message. The value specifies the timeout period in 50-millisecond increments. When the timeout occurs, the receive buffer is terminated and sent to the client.

To set this option using the DLI configuration file, use the recvTimeout parameter; for example, recvTimeout = 6. See Table 4–2 on page 82.

3.10 Queue Limit Option (10)

The queue limit option defines the maximum number of buffers a link may queue for the client. Queue limiting prevents a single link from using all the buffers on the ICP. When a link's queue limit is exceeded, receive message function codes are changed to the data lost function code. Subsequent received data is lost because the ICP does not queue buffers for the client from a link that has exceeded its queue limit. A value of zero disables queue limiting for that link.

To set this option using the DLI configuration file, use the qLimit parameter; for example, qLimit = 4. See Table 4–2 on page 82.

3.11 Termination Count Option (11)

The termination count option defines the number of characters that the ICP will buffer for a link before terminating the buffer. The value must be less than or equal to the configured ICP buffer size.

To set this option using the DLI configuration file, use the termCount parameter; for example, termCount = 1024. See Table 4–2 on page 82.

3.12 Message Format Option (12)

The message format option defines the receive message format. AWS uses this value to determine receive message processing. It can specify *unformatted*, *control character set termination*, *control character set framing*, *character sequence framing*, or *character sequence synchronize start*. Receive message buffers are queued for the client whenever an end-of-message is detected or a buffer is filled with characters. Error conditions such as a receive timeout or DCD error when using full- or half-duplex modem control can cause a partially filled buffer to be queued for the client.

To set this option using the DLI configuration file, use the msgFormat parameter; for example, msgFormat = "unformatted". See Table 4–2 on page 82.

3.12.1 Unformatted

When the message format option is set to *unformatted*, AWS accepts all incoming characters. The received data is queued for the client when the buffered character count reaches the configured buffer size. The configured buffer size is specified by the Set ICP Message Buffer Size command (Section 2.4.1.1 on page 41) and may be logically limited by the termination count option (Section 3.11 on page 62).

3.12.2 Control Character Set Termination

When the message format option is set to *control character set termination*, receive messages are terminated by a single ASCII control character (0 to 31). The EOM longword option (Section 3.14 on page 66) defines the set of terminating characters. The correspondence is between the bit number and the binary value of the character; the character is a terminator if the bit is set. For example, if bit zero and bit 13 are set in the EOM longword, null (0) or carriage return (13) are message terminators. If all bits are set, any binary value from 0 through 31 is a message terminator.

3.12.3 Control Character Set Framing

When the message format option is set to *control character set framing*, receive messages are framed by a single ASCII control characters (0 to 31). The SOM longword option (Section 3.13 on page 65) defines the set of characters that start a message. The EOM longword option (Section 3.14 on page 66) defines the set of terminating characters. The correspondence is between the bit number and the binary value of the character. For example, if bit one and bit two are set in the SOM longword, start-of-header (SOH) (1) or start-of-text (STX) (2) signify the start of a message. If bit 3 and bit 4 are set in the EOM longword, end-of-text (ETX) (3) or end-of-transmission (EOT) (4) signify the end of a message.

3.12.4 Character Sequence Framing

When the message format option is set to *character sequence framing*, receive messages are framed by characters ordered in a unique one- to four-byte sequence. AWS uses the following four configuration options to implement character sequence framing:

Start count (3)	The number of characters that indicate the beginning of a receive message
SOM longword (13)	The ASCII character values to be used for comparison
End count (4)	The number of characters that indicate the end of a receive message
EOM longword (14)	The ASCII character values to be used for comparison

The example below shows the start and end of a message framed by a single character.

Message format: STX … DATA … ETX
Start count: 1
SOM longword: 2
End count: 1
EOM longword: 3

The example below shows the start and end of a message framed by four characters.

Message format: zczc … DATA … nnnn
Start count: 4
SOM longword: 7A637A63 (hex)
End count: 4
EOM longword: 6E6E6E6E (hex)

3.12.5 Character Sequence Synchronize Start

When the message format option is set to *character sequence synchronize start,* receive messages are synchronized at the start of the first message by characters ordered in a unique one- to four-byte sequence. AWS uses the following three configuration options to implement character sequence synchronize start:

Start count (3)	The number of characters that indicate the beginning of receive data.
SOM longword (13)	The ASCII character values to be used for comparison.
End count (4)	Set to zero.

Once synchronized, blocks of characters are buffered up and transferred to the reading client machine (i.e., no further synchronization is required for subsequent blocks). This mode continues until the line is shut down or a configured timeout value expires indicating that incoming data has stopped. When either of these occurs, the partial buffer (if any) is forwarded to the client with a timeout status, and the synchronization criteria will apply to any subsequent data received.

3.13 Start-of-Message Longword Option (13)

The start-of-message (SOM) longword option is interpreted according to the message format. For *character sequence framed* and *character sequence synchronize start* formats, SOM defines the character sequence that delimits the start of a receive message. For *control character set framed* formats, SOM is used as the bit-mask longword. Enter in hex format.

To set this option using the DLI configuration file, use the startOfMessage parameter; for example, startOfMessage = 0x02000000. See Table 4–2 on page 82.

3.14 End-of-Message Longword Option (14)

The end-of-message (EOM) longword option is interpreted according to the message format. For *character sequence framed* formats, EOM defines the character sequence that delimits the end of a receive message. For *control character set termination* and *control character set framed* formats, EOM is used as the bit-mask longword. Enter in hex format.

To set this option using the DLI configuration file, use the endOfMessage parameter; for example, endOfMessage = 0x03000000. See Table 4–2 on page 82.

3.15 Alternate Data Rate Option (15)

The user-defined data rate option overrides the data rate option (Section 3.1). The value for this option can be determined from the formula:

$$V = \frac{1}{R} \times \left(\frac{f}{32}\right) - 2$$

where

V is the time constant and must be an integer value
R is the data rate
f is the PCLK frequency

The Freeway PCLK frequency equals 7.3728 x 10^6.

An example follows for 14400 bits per second:

$$V = 16 - 2$$

$$V = \frac{7372800}{14400 \times 32} - 2$$

$$V = 14$$

Note

This option applies only to asynchronous lines.

To set this option using the DLI configuration file, use the altDataRate parameter; for example, altDataRate = 14. See Table 4–2 on page 82.

3.16 Secure Mode Option (16)

The secure mode option enables or disables the secure mode feature for a link. The secure mode defines a client-to-ICP protocol that only transmits messages with a Cyclic Redundancy Check (CRC-16) error check appended at the end of the data. When the secure mode is *enabled* for a link, the following actions take place:

- The client program appends a CRC-16 polynomial to the end of all transmit message data. The data size in the message header includes the two bytes for the CRC-16.
- Before queuing the message for transmission, the ICP runs a CRC-16 error check on the data. If the ICP-generated CRC-16 matches the client-generated CRC-16, the message is queued for transmission. A CRC-16 error (and therefore a secure mode violation) is signified by the transmit negative acknowledge message sent to the client.
- In all cases the buffer is zeroed when it is released to the free buffer pool.

- The two-byte CRC-16 polynomial is not transmitted.

To set this option using the DLI configuration file, use the secureMode parameter; for example, secureMode = "disable". See Table 4–2 on page 82.

3.17 Crypto Resync Option (17)

The crypto resync option specifies the number of milliseconds that the DTR modem signal will be turned on to effect a crypto resync pulse when the Send Crypto Resync Command (Section 2.4.1.7 on page 44) is issued by the client. A value of 0 (the default) means that crypto resync function is disabled. Valid values for the pulse duration are 1 through 32767.

To set this option using the DLI configuration file, use the crypto parameter; for example, crypto = 20. See Table 4–2 on page 82.

3.18 CTS Timeout Option (22)

The clear to send (CTS) timeout option monitors the CTS modem signal. Full- or half-duplex modem support is required to enable the CTS timeout function. The CTS timeout value specifies the acceptable number of seconds CTS may be deasserted before aborting the transmission. The CTS timeout is reported in the header status field of a transmit negative acknowledge. The number of characters sent before aborting the transmission is reported in the header transfer count field. A transfer count of zero signifies that the transmission never started (no CTS). Setting the CTS timeout option to zero is the same as disabling the link.

To set this option using the DLI configuration file, use the ctsTimeout parameter; for example, ctsTimeout = 0. See Table 4–2 on page 82.

3.19 RTS Delay Option (23)

The ready to send (RTS) delay option keeps RTS asserted for an additional period after a transmission has completed. The RTS delay is used only with half-duplex modem control. The RTS delay value specifies the number of seconds to wait after the last byte has been sent before deasserting RTS. The RTS delay keeps RTS active to account for propagation delays caused by modems in half-duplex circuits. Setting the RTS delay option to zero is the same as disabling the link.

To set this option using the DLI configuration file, use the rtsDelay parameter; for example, rtsDelay = 0. See Table 4–2 on page 82.

3.20 Modem Control Option (24)

The modem control option enables modem control for the communication link. The modem control options are: *disabled*, *full duplex*, and *half duplex*. The data carrier detect (DCD) and CTS modem input signals are ignored when modem control is disabled. The DTR output signal is asserted while the link is enabled, regardless of the modem control type. The RTS output signal is deasserted while the link is enabled when modem control is disabled.

DCD is required to enable the receiver when modem control is enabled (modem control = *full duplex* or *half duplex*). The communication software reports an error when DCD is lost while a message is being received and modem control is enabled.

CTS is required to enable the transmitter when modem control is enabled. RTS is asserted while the link is enabled and configured for full duplex modem control. RTS is asserted only while a transmission is in progress for half duplex modem control. See the CTS timeout and RTS delay options for additional modem signal control functions. Table 3–2 shows the modem control options and signals supported.

To set this option using the DLI configuration file, use the modemControl parameter; for example, modemControl = “none”. See Table 4–2 on page 82.

Table 3–2: Modem Control Options and Modem Signals

Modem Control	DTR	RTS	CTS	DCD
None	Always *on* while link is enabled	*Off*	Ignored	Ignored
Full Duplex		Always *on* while link is enabled	Required to enable transmitter; may be monitored by CTS timeout	Required to enable receiver; DCD lost is reported in the receive message buffer header
Half Duplex		*On* while transmission is in progress; may use RTS delay		

3.21 Format Transmit Messages Option (25)

The format transmit messages option enables or disables automatic transmit message framing. If the link message format (Section 3.12) is configured for *character sequence framed* formats, and format transmit messages is *enabled*, the ICP automatically sends the start-of-message sequence and end-of-message sequence to frame transmit messages. If the message format is configured for *character sequence synchronize start*, and format transmit messages is *enabled*, the ICP automatically sends the start-of-message sequence with each message.

To set this option using the DLI configuration file, use the formatXmitMsgs parameter; for example, formatXmitMsgs = "disable". See Table 4–2 on page 82.

3.22 Receive Block Check Option (29)

The values for receive block checking are *none*, *LRC-8* or *CRC-16*. If the link message format (Section 3.12) is configured for *control character set framing* or *character sequence framing*, and receive block checking is enabled, a one-byte LRC-8 or a two-byte CRC-16 block check character received with the message is passed to the client application as part of the message, depending on the setting for this option.

To set this option using the DLI configuration file, use the recvBlockCheck parameter; for example, recvBlockCheck = "none". See Table 4–2 on page 82.

Note

The ICP does *not* verify receive block check characters but simply passes them on to the client application, which may process them.

3.23 Receive Framing Characters Option (30)

If the link message format (Section 3.12) is configured for *control character set framing* or *character sequence framing* and the receive framing option is *enabled*, the character(s) framing the received message are passed to the client application as part of the message. If the message format is configured for *character sequence synchronize start*, the sync characters are passed to the client application with the first synchronized message.

To set this option using the DLI configuration file, use the recvFramingChars parameter; for example, recvFramingChars = "disable". See Table 4–2 on page 82.

3.24 Baudot Code Option (31)

The Baudot code conversion option enables or disables the ASCII-to-Baudot and Baudot-to-ASCII code conversion. This option is to be used when the serial line data is five-bit Baudot characters. The code conversion supports the CCITT International Alphabet No. 2 (ITA #2). There is no code conversion for ASCII characters that do not map to the translation tables. The ASCII lower case alphabetic characters are converted to upper case.

When this option is *enabled*, the AWS software converts ASCII data from the client application to Baudot before transmitting the data on the serial line. The AWS software converts Baudot data received from the serial line to ASCII before sending the data to the client application.

To set this option using the DLI configuration file, use the baudotCode parameter; for example, baudotCode = "disable". See Table 4–2 on page 82.

This option can be used only for five-bit (bits/char[5] = 1) Baudot encoded serial data. Table 3–3 and Table 3–4 show the code conversion for letters and figures mode.

Table 3–3: Baudot Code Table (Letters Mode)

ASCII	Baudot	ASCII	Baudot
null	0	T	10
E	1	Z	11
line feed	2	L	12
A	3	W	13
space	4	H	14
S	5	Y	15
I	6	P	16
U	7	Q	17
return	8	O	18
D	9	B	19
R	A	G	1A
J	B	(FIG)[1]	1B
N	C	M	1C
F	D	X	1D
C	E	V	1E
K	F	(LET)[a]	1F

[1] Baudot mode change characters

3.25 Software Handshake Option (32)

The software handshake option defines the type of software flow control on the serial line. This option has three choices: *disable*, *XON/XOFF*, and *XOFF/XANY*. The default configuration is *disable* (no software flow control). The software flow control feature can be used only for ASCII seven- and eight-bit data.

The *XON/XOFF* option supports recognition of XON and XOFF characters to stop and start data transmission from an ICP serial data link. A transmission in progress is interrupted and new transmissions are suspended after receipt of the XOFF character. An

Table 3–4: Baudot Code Table (Figures Mode)

ASCII	Baudot	ASCII	Baudot
null	0	5	10
3	1	"	11
line feed	2	)	12
–	3	2	13
space	4	#	14
bell	5	6	15
8	6	0	16
7	7	1	17
return	8	9	18
$	9	?	19
4	A	&	1A
\	B	(FIG)[1]	1B
,	C	.	1C
!	D	/	1D
:	E	;	1E
(	F	(LET)[a]	1F

[1] Baudot mode change characters

interrupted transmission is resumed and new transmissions are allowed after receipt of the XON character. The XOFF and XON characters are not buffered for the client.

The *XOFF/XANY* option supports recognition of the XOFF character to stop data transmission and any character to restart transmission from an ICP serial data link. A transmission in progress is interrupted and new transmissions are suspended after receipt of the XOFF character. An interrupted transmission is resumed and new transmissions are allowed after receipt of any character. The XOFF character is not buffered for the client. The XANY character (first character received after XOFF) is buffered and processed according to the link message format configuration.

To set this option using the DLI configuration file, use the softHandshake parameter; for example, softHandshake = "disable". See Table 4–2 on page 82.

3.26 Isochronous Mode Option (34)

The isochronous mode option enables or disables isochronous communications. Isochronous communication allows you to send or receive asynchronous format data by synchronous means (i.e., an external clock source).

If you select isochronous mode, configure the clock source option (Section 3.2) to *external* and have an external clock source (a modem or modem eliminator). Isochronous mode must be *disabled* if you select a standard data rate of 50 bits/second.

To set this option using the DLI configuration file, use the isochronous parameter; for example, isochronous = "disable". See Table 4–2 on page 82.

3.27 Electrical Interface Option (40)

The electrical interface option allows the electrical interface for each link to be set. It applies to ICPs that have a configurable electrical interface; otherwise it is ignored. The valid choices are EIA-232 (default), EIA-449, EIA-530, and V.35.

To set this option using the DLI configuration file, use the elecInterface parameter; for example, elecInterface = "EIA232". See Table 4–2 on page 82.

Chapter

4 AWS Link Configuration Using dlicfg

Note

In this document, the term "Freeway" can mean either a Freeway server or an embedded ICP. For the embedded ICP, also refer to the user guide for your ICP and operating system (for example, the *ICP2432 User Guide for Windows NT*).

4.1 Configuration Overview

Section 2.1.1 on page 30 summarized your choices for performing ICP link configuration. This chapter describes the AWS link configuration process using the DLI text configuration file as input to the dlicfg preprocessor program to produce a binary configuration file which is used by the dlInit and dlOpen functions. For embedded ICPs, only a DLI configuration file is used (not a TSI configuration file).

If you use the DLI configuration file to define link configuration, and later need to change a link parameter value, you must shut down your application, modify the DLI text configuration file, rerun dlicfg, and then restart your application using the updated binary configuration file (you do not have to rebuild your application). If you need to make changes to link configuration frequently, consider using the Configure Link command (Section 2.4.1.2 on page 41) in your application.

Even if you choose not to use the DLI configuration file to define the AWS links, you still must configure DLI sessions (and TSI connections for a Freeway server). You should be familiar with the protocol-independent configuration procedures described in the

Freeway Data Link Interface Reference Guide and the *Freeway Transport Subsystem Interface Reference Guide*.

During your client application development and testing, you might need to perform DLI configuration repeatedly (as well as TSI configuration for a Freeway server). The DLI and TSI configuration files provided with the product are listed in Table 4–1.

Table 4–1: Configuration File Names

	Freeway Server	**Embedded ICP**
DLI:	awsaldcfg	awsacfg awsscfg
TSI:	awsaltcfg	TSI not applicable for embedded ICP

The DLI and TSI configuration procedures are summarized as follows. Keep in mind that TSI configuration does not apply to an embedded ICP environment.

1. For a Freeway server, create or modify a TSI text configuration file specifying the configuration of the TSI connections (for example, awsaltcfg in the freeway/client/test/aws directory).

2. Create or modify a DLI text configuration file specifying the DLI session configuration and optional ICP link configurations for all ICPs and serial communication links in your system (for example, awsaldcfg in the freeway/client/ test/aws directory).

3. If you have a UNIX or Windows NT system, skip this step. If you have a VMS system, run the makefc.com command file from the [FREEWAY.CLIENT.TEST.AWS] directory to create the foreign commands used for dlicfg and tsicfg.

@MAKEFC <*tcp-sys*>

where <*tcp-sys*> is your TCP/IP package:

MULTINET (for a Multinet system)

TCPWARE (for TCPware system)

UCX (for a UCX system)

VMS example: @MAKEFC UCX

4. For a Freeway server, go to the freeway/client/test/aws directory and execute tsicfg with the text file from Step 1 as input. This creates the TSI binary configuration file in the same directory as the location of the text file (unless a different path is supplied with the optional filename). If the optional filename is not supplied, the binary file is given the same name as your TSI text configuration file plus a .bin extension.

   tsicfg *TSI-text-configuration-filename [TSI-binary-configuration-filename]*

   VMS example: tsicfg awsaltcfg

   UNIX example: freeway/client/*op-sys*/bin/tsicfg awsaltcfg

   NT example: freeway\client*op-sys*\bin\tsicfg awsaltcfg

5. From the freeway/client/test/aws directory (or the freeway/client/[nt_dlite or sol_dlite]/aws directory for an embedded ICP), execute dlicfg with the text file from Step 2 as input. This creates the DLI binary configuration file in the same directory as the location of the text file (unless a different path is supplied with the optional filename). If the optional filename is not supplied, the binary file is given the same name as your DLI text configuration file plus a .bin extension.

   dlicfg *DLI-text-configuration-filename [DLI-binary-configuration-filename]*

   VMS example: dlicfg awsaldcfg

   UNIX example: freeway/client/*op-sys*/bin/dlicfg awsaldcfg

   NT example: freeway\client*op-sys*\bin\dlicfg awsaldcfg

Note

You must rerun dlicfg or tsicfg whenever you modify the text configuration file so that the DLI or TSI functions can apply the changes. On all but VMS systems, if a binary file already exists with the same name in the directory, the existing file is renamed by appending the .BAK extension. If the renamed file duplicates an existing file in the directory, the existing file is removed by the configuration preprocessor program.

6. If you have a UNIX system, move the binary configuration files that you created in Step 4 and Step 5 into the appropriate freeway/client/*op-sys*/bin directory where *op-sys* indicates the operating system: for example, dec, hpux, sgi, solaris, or sunos (or sol_emb for an embedded ICP).

   UNIX example: mv awsaldcfg.bin /usr/local/freeway/client/sunos/bin
   mv awsaltcfg.bin /usr/local/freeway/client/sunos/bin

7. If you have a VMS system, run the move.com command file from the [FREEWAY.CLIENT.TEST.AWS] directory. This moves the binary configuration files you created in Step 4 and Step 5 into the bin directory for your TCP/IP package.

   @MOVE *filename* <*tcp-sys*>
   where *filename* is the name of the binary configuration file and <*tcp-sys*> is the TCP/IP package:
   MULTINET (for a Multinet system)
   TCPWARE (for TCPware system)
   UCX (for a UCX system)
   VMS example: @MOVE AWSALDCFG.BIN UCX

8. If you have a Windows NT system, move the binary configuration files that you created in Step 4 and Step 5 into the appropriate freeway\client*op-sys*\bin directory where *op-sys* indicates the operating system: axp_nt or int_nt (for a Freeway server); axp_nt_emb or int_nt_emb (for an embedded ICP).

NT example: copy awsaldcfg.bin \freeway\client\axp_nt\bin

copy awsaltcfg.bin \freeway\client\axp_nt\bin

When your application calls the dlInit function, the DLI and TSI binary configuration files generated in Step 4 and Step 5 are used to configure the DLI sessions and TSI connections. Figure 4–1 shows the configuration process.

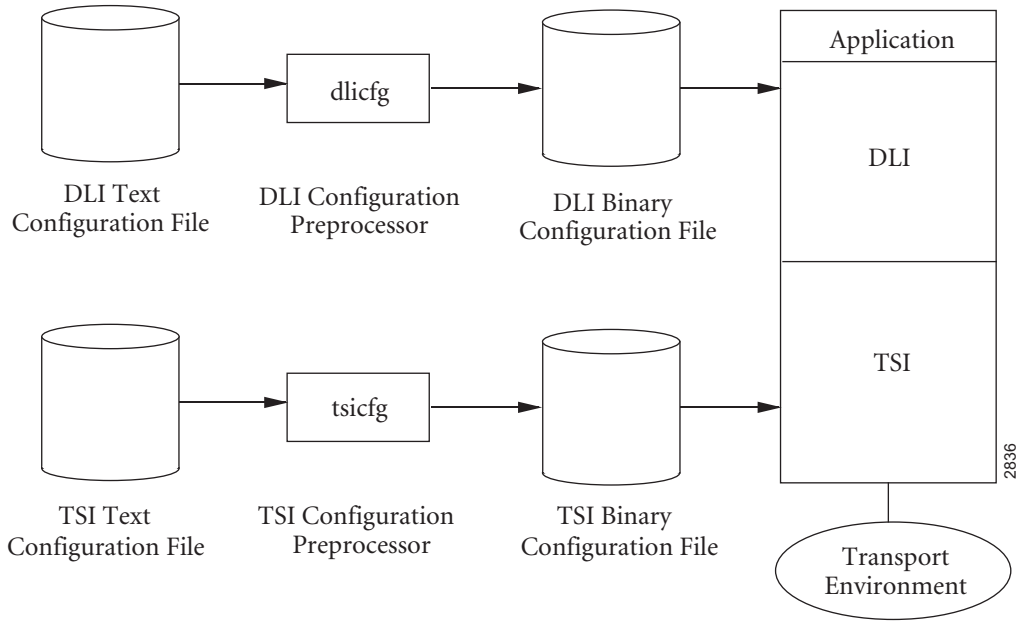

Figure 4–1: DLI and TSI Configuration Process

4.2 DLI Session Configuration

The DLI text configuration file used by the dlicfg program consists of the following sections:

- A "main" section which specifies the DLI configuration for non-session-specific operations (described in the *Freeway Data Link Interface Reference Guide*)
- One or more additional sections, each specifying a protocol-specific session associated with a particular Freeway serial communication link (port). Each link can be configured independently of the other links.

The protocol-specific session parameters can be divided into two groups:

- Client-related parameters are described in the *Freeway Data Link Interface Reference Guide*. For example, each session has an associated TSI connection name which you also specify in your TSI configuration file, though multiple sessions can use the same TSI connection.
- Protocol-specific link parameter values which are different from the defaults are shown in Table 4–2. These parameters are optional in the DLI text configuration file and are supported only in DLI *Normal* operation. The dlWrite Configure Link command described in Section 2.4.1.2 on page 41 also can perform protocol-specific configuration.

For a Freeway server, Figure 4–2 shows the "main" section and two AWS sessions. The DLI client-related parameters are shown in typewriter type. The protocol-specific parameters are shown in **bold typewriter type**. You need to include only those parameters whose values differ from the defaults. Chapter 3 describes the link configuration options in detail.

The syntax for the AWS link configuration parameters is shown in Table 4–2, along with the defaults. The parameter names are case independent but are shown in upper and lower case for readability.

```
main                                    // DLI "main" section:                  //
{
    asyncIO = "no";                     // Wait for I/O completion              //
    tsiCfgName = "awsaltcfg.bin";       // TSI binary config file               //
}

ICP0link0                               // First session name:                  //
{                                       // Client-related parameters:           //
    asyncIO = "no";                     //     Use blocking I/O                 //
    boardNo = 0;                        //     First ICP is zero                //
    portNo = 0;                         //     First ICP link is zero           //
    protocol = "AWS";                   //     Session protocol                 //
    transport = "client1";              //     TSI connection name specified    //
                                        //         in TSI configuration file    //
    // Optional protocol parameters (different from defaults)://
    dataRate = 4800;                    //  4800 bits/second                    //
    qLimit = 10;                        // 10-buffer queue limit                //
}

ICP0link1                               // Second session name:                 //
{                                       // Client-related parameters:           //
    asyncIO = "no";                     //     Use blocking I/O                 //
    boardNo = 0;                        //     First ICP is zero                //
    portNo = 1;                         //     Second ICP link is one           //
    protocol = "AWS";                   //     Session protocol                 //
    transport = "client1";              //     TSI connection name specified    //
                                        //         in TSI configuration file    //

    // Optional protocol parameters (different from defaults)://
    termcount = 512;                    // 512-byte termination count           //
}
```

Figure 4–2: DLI Configuration File for Two Links (Freeway Server)

Table 4–2: AWS ICP Link Parameters and Defaults for Using dlicfg

dlicfg Option Name	Default	Valid Values
dataRate	1200	50, 56, 74, 300, 1050, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 76800 or 115200
clockSource	“internal”	“external” or “internal”
startCount	0	0–4
endCount	0	0–4
bitsPerChar	“8bits”	“5bits” “6bits” “7bits” “8bits”
parity	“none”	“none” “odd” “even” “mark” “space”
stopBits	“one”	“one” “oneandhalf” “two”
transmitExpiration	0	0, 1-65535
recvTimeout	6	0–8192
qLimit	4	0–4096
termCount	1024	0–8192 (default = ICP message buffer size)
msgFormat	“unformatted”	“unformatted” “ctrlcharsetterm” “ctrlcharsetframing” “charseqframing” “charseqstartsync.”
startOfMessage	0x02000000	0x00–0xFFFFFFFF
endOfMessage	0x03000000	0x00–0xFFFFFFFF
altDataRate	0	0–230400
secureMode	“disable”	“disable” or “enable”
crypto	0	0-32767
ctsTimeout	0	0–8192
rtsDelay	0	0–8192

Table 4–2: AWS ICP Link Parameters and Defaults for Using dlicfg

dlicfg Option Name	Default	Valid Values
modemControl	"none"	"none" "fullduplex" "halfduplex"
formatXmitMsgs	"disable"	"disable" or "enable"
recvBlockCheck	"none"	"none" "lrc8" "crc16"
recvFramingChars	"disable"	"disable" or "enable"
baudotCode	"disable"	"disable" or "enable"
softHandshake	"disable"	"disable" "xonxoff" "xonxany"
isochronous	"disable"	"disable" or "enable"
elecInterface	"EIA232"	"EIA232" "unbEIA449" "EIA530" "V35"
bufferSize[1]	1024	64–8192

[1] The bufferSize parameter allows the DLI to configure the ICP message buffer size (equivalent to the Set ICP Message Buffer Size command in Section 2.4.1.1 on page 41).

For an embedded ICP using the DLITE interface, Figure 4–3 shows the "main" section and two AWS sessions. You need to include only those parameters whose values differ from the defaults. Since the DLITE interface supports only *Raw* operation, protocol-specific parameters are not included. For more information on the DLITE interface, refer to the user guide for your embedded ICP and operating system (for example, the *ICP2432 User Guide for Windows NT (DLITE Interface)*)

```
main                                    // DLI "main" section:                  //
{
      asyncIO = "yes";                  // Use non-blocking I/O                 //
      tsiCfgName = "."                  // tsiCfgName unused for DLITE      //
                        // Exception: For NT = Location of log/trace svc  //
// The following two parameters are for DLITE only:                          //
      maxBuffers = 1024;
      maxBufSize = 1200;
}

ICP0link0                               // First session name:                  //
{                                       // Client-related parameters:           //
      alwaysQIO = "yes";                //      Queue I/Os even if complete     //
      asyncIO = "yes";                  //      Use non-blocking I/O            //
      cfgLink = "no";                   //      Client configures links         //
      enable = "no";                    //      Client enables links            //
      localAck = "no";                  //      Client processes transmit ack   //
      boardNo = 0;                      //      First ICP is zero               //
      portNo = 0;                       //      First ICP link is zero          //
      protocol = "raw";                 //      DLITE requires Raw operation    //
      maxBufSize = 1200;                //      Used by DLITE                   //

// Protocol parameters not supported for "Raw" operation  //
}

ICP0link1                               // Second session name:                 //
{                                       // Client-related parameters:           //
      alwaysQIO = "yes";                //      Queue I/Os even if complete     //
      asyncIO = "yes";                  //      Use non-blocking I/O            //
      cfgLink = "no";                   //      Client configures links         //
      enable = "no";                    //      Client enables links            //
      localAck = "no";                  //      Client processes transmit ack   //
      boardNo = 0;                      //      First ICP is zero               //
      portNo = 1;                       //      First ICP link is zero          //
      protocol = "raw";                 //      DLITE requires Raw operation    //
      maxBufSize = 1200;                //      Used by DLITE                   //

// Protocol parameters not supported for "Raw" operation  //
}
```

Figure 4–3: DLI Configuration File for Two Embedded ICP Links (DLITE Interface)

Appendix

A Error Codes

There are several methods used by the DLI and AWS software to report errors, as described in the following sections:

A.1 DLI Error Codes

The error code can be returned directly by the DLI function call in the global variable dlerrno. Typical errors are those described in the *Freeway Data Link Interface Reference Guide*.

A.2 ICP Global Error Codes

Table A–1 lists the AWS ICP-related errors that can be returned in the global variable iICPStatus. The DLI constants are defined in the file dlicperr.h.

A.3 ICP Error Status Codes

The AWS ICP-related errors and status information listed in Table A–1 can also be returned in the dlRead pOptArgs.iICPStatus field of the response, which is a duplicate of the iIPCStatus global variable. The DLI sets the dlRead pOptArgs.usProtCommand field to the same value as the dlWrite request that caused the error.

Table A–1: ICP Error Status Codes used by AWS

Code	Mnemonic	Meaning
0	DLI_ICP_ERR_NO_ERR	A data block has been successfully transmitted or received on the line or a command has been successfully executed.
–101	DLI_ICP_ERR_BAD_NODE	DLI internal error. An invalid node number was passed to the ICP from the DLI.
–102	DLI_ICP_ERR_BAD_LINK	The link number from the client program is not a legal value.
–103	DLI_ICP_ERR_NO_CLIENT	The maximum number of clients are registered for the link.
–105	DLI_ICP_ERR_BAD_CMD	The command from the client program is not a legal value.
–119	DLI_ICP_ERR_BAD_SESSID	The session number is invalid.
–122	DLI_ICP_ERR_BAD_PARMS	The values used for the function call are illegal.
–145	DLI_ICP_ERR_INBUF_OVERFLOW	Input buffer overflow
–146	DLI_ICP_ERR_OUTBUF_OVERFLOW	Output buffer overflow

A.4 Protocol Error/Status Codes

Table A–2 lists the AWS protocol error/status conditions that are reported via bit numbers in the dlRead pOptArgs.iProtModifier field. It is possible for more than one of these bits to be set.

Table A–2: AWS Protocol Error Status Codes

pOptArgs.iProtModifier Field Value (bit)	**Description**
0	(reserved)
1	Receive framing error
2	Receive parity error
3	Receive overrun error
4	End-of-message frame
5	Receive buffer full
6	Receive timeout
7	Loss of carrier (DCD)
8	Broken message
9	Secure-mode violation
10	CTS timeout
11	Crypto resync
12	Transmit expiration
13	(reserved)
14	(reserved)
15	Rejected message

Appendix

B AWS Detailed Command and Response Formats

The command and response header formats in this appendix are to aid programmers who are writing an application program under one of the following conditions:

1. If you are writing to Protogate's data link interface (DLI) using *Raw* operation, also refer to the *Freeway Data Link Interface Reference Guide*. If you are using the embedded DLITE interface, also refer to the user guide for your particular ICP and operating system; for example, the *ICP2432 User Guide for Windows NT*.

2. If you are writing a non-DLI application using a Protogate driver interface, also refer to the user guide for your particular ICP and operating system; for example, the *ICP2432 User Guide for Windows NT*.

3. If you are writing a non-DLI application using a socket interface, also refer to the *Freeway Client-Server Interface Control Document*.

B.1 AWS Protocol Processing

B.1.1 Session Attach

The client application must process a DLI_ICP_CMD_ATTACH command and subsequent response for each link that is accessed. A protocol session ID number is returned in the usProtSessionID field in the Protocol Header of a successful DLI_ICP_CMD_ATTACH response. This returned ID number must be saved and placed in every subsequent Protocol Header command message. Note that the protocol session ID is different from the DLI session ID used in the dlRead and dlWrite calls. See Table B–2 on page 93.

B.1.2 Set Buffer Size

The client application must process a DLI_PROT_SET_BUF_SIZE command and subsequent response to the ICP to set the maximum buffer size on data that is transmitted and/or received on the protocol link. This command message should be sent at the completion of all link attaches, and before any other command messages are sent to the ICP. Note that the usICPCommand field of the ICP Header is DLI_ICP_CMD_WRITE. See Table B–6 on page 97.

B.1.3 Link Configuration

The client application must process a DLI_PROT_CFG_LINK command and subsequent response to the ICP to configure each link. Note that the usICPCommand field of the ICP Header is DLI_ICP_CMD_WRITE. See Table B–7 on page 98.

B.1.4 Link Enabling (Bind)

The client application must process a DLI_ICP_CMD_BIND command and subsequent response to the ICP to enable a link for transfer or receipt of data. See Table B–4 on page 95.

B.1.5 Data Transfer

The client application is now ready to process any data that is to be transmitted or received. To transmit data, the client must send data using the DLI_PROT_SEND_NORM_DATA protocol command. Note that the usICPCommand field of the ICP Header is always DLI_ICP_CMD_WRITE. See Table B–15 on page 106.

The client application must now be ready to receive both acknowledgments to transmitted data (DLI_PROT_RESP_LOCAL_ACK) and data received from the data link which is received as DLI_PROT_RECV_DATA. See Table B–15 on page 106 and Table B–17 on page 108.

Other protocol commands (and subsequent responses) that can be sent to or received from the ICP are shown in the format tables in Section B.2 and Section B.3.

B.1.6 Link Disabling (Unbind)

When the client application is ready to terminate data transfer of a link, it processes a DLI_ICP_CMD_UNBIND command and subsequent response to the ICP to disable the specified link. See Table B–5 on page 96.

B.1.7 Session Detach

When the client application is ready to terminate processing, it processes a DLI_ICP_CMD_DETACH command and subsequent response for each node and link that is be detached from session management. See Table B–3 on page 94.

After a session and link have been detached, the same, or another client application is free to attach a session to the link. Without normal session detaches, any link that is attached on the ICP is not re-usable, and the ICP must be re-downloaded to make the link again available.

B.2 Command and Response Header Tables

Table B–1 lists the codes AWS for commands originated by the client and the associated responses to the client. The required header field values for both the ICP Header and the Protocol Header are detailed in the referenced tables.

Table B–1: AWS Command/Response Codes

Command/Response Code	Reference Table
DLI_ICP_CMD_ATTACH	Table B–2 on page 93
DLI_ICP_CMD_DETACH	Table B–3 on page 94
DLI_ICP_CMD_BIND	Table B–4 on page 95
DLI_ICP_CMD_UNBIND	Table B–5 on page 96
DLI_PROT_SET_BUF_SIZE	Table B–6 on page 97
DLI_PROT_CFG_LINK	Table B–7 on page 98
DLI_PROT_CLR_STATISTICS	Table B–8 on page 99
DLI_PROT_SET_SIG	Table B–9 on page 100
DLI_PROT_GET_ICP_REPORT	Table B–10 on page 101
DLI_PROT_SEND_CRYPTO_RESYNC	Table B–11 on page 102
DLI_PROT_GET_LINK_CFG	Table B–12 on page 103
DLI_PROT_GET_SIG_REPORT	Table B–13 on page 104
DLI_PROT_GET_STATISTICS	Table B–14 on page 105
DLI_PROT_SEND_NORM_DATA	Table B–15 on page 106

Table B–2: AWS DLI_ICP_CMD_ATTACH Command and Response

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	N/A
	fill2	N/A	N/A
	iICPSize	16	16
	usICPCommand	DLI_ICP_CMD_ATTACH	DLI_ICP_CMD_ATTACH
	iICPStatus	*See Note 1.*	*See Note 2.*
	usICPParms[0]	*See Note 4.*	*See Note 4.*
	usICPParms[1]	0	0
	usICPParms[2]	N/A	N/A
Prot_header	usProtCommand	DLI_ICP_CMD_ATTACH	DLI_ICP_CMD_ATTACH
	iProtModifier	N/A	*See Note 2.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	0	*See Note 3.*
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	0	N/A

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See Appendix A)
3. This returned Session ID must be provided on all subsequent writes for this protocol session.
4. Node number used for reading responses for this protocol session. This field is handled automatically by DLI or DLITE. If you are not using DLI or DLITE, your application or the device driver must fill in the proper read node number.

Table B–3: AWS DLI_ICP_CMD_DETACH Command and Response

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	N/A
	fill2	N/A	N/A
	iICPSize	16	16
	usICPCommand	DLI_ICP_CMD_DETACH	DLI_ICP_CMD_DETACH
	iICPStatus	*See Note 1.*	*See Note 2.*
	usICPParms[0]	0	0
	usICPParms[1]	N/A	N/A
	usICPParms[2]	N/A	N/A
Prot_header	usProtCommand	DLI_ICP_CMD_DETACH	DLI_ICP_CMD_DETACH
	iProtModifier	N/A	*See Note 3.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	Protocol Session ID	Protocol Session ID
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	0	0

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See Appendix A)
3. Protocol status (See Appendix A)

Table B–4: AWS DLI_ICP_CMD_BIND Command and Response

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	N/A
	fill2	N/A	N/A
	iICPSize	16	N/A
	usICPCommand	DLI_ICP_CMD_BIND	DLI_ICP_CMD_BIND
	iICPStatus	*See Note 1.*	*See Note 2.*
	usICPParms[0]	0	0
	usICPParms[1]	N/A	N/A
	usICPParms[2]	N/A	N/A
Prot_header	usProtCommand	DLI_ICP_CMD_BIND	DLI_ICP_CMD_BIND
	iProtModifier	N/A	*See Note 3.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	Protocol Session ID	Protocol Session ID
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	0	0

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See Appendix A)
3. Protocol status (See Appendix A)

Table B–5: AWS DLI_ICP_CMD_UNBIND Command and Response

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	N/A
	fill2	N/A	N/A
	iICPSize	16	N/A
	usICPCommand	DLI_ICP_CMD_UNBIND	DLI_ICP_CMD_UNBIND
	iICPStatus	*See Note 1.*	*See Note 2.*
	usICPParms[0]	0	0
	usICPParms[1]	N/A	N/A
	usICPParms[2]	N/A	N/A
Prot_header	usProtCommand	DLI_ICP_CMD_UNBIND	DLI_ICP_CMD_UNBIND
	iProtModifier	N/A	*See Note 3.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	Protocol Session ID	Protocol Session ID
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	0	0

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See Appendix A)
3. Protocol status (See Appendix A)

Table B–6: AWS DLI_PROT_SET_BUF_SIZE Command and Response

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	N/A
	fill2	N/A	N/A
	iICPSize	18	18
	usICPCommand	DLI_ICP_CMD_WRITE	DLI_ICP_CMD_READ
	iICPStatus	*See Note 1.*	*See Note 2.*
	usICPParms[0]	0	0
	usICPParms[1]	N/A	N/A
	usICPParms[2]	N/A	N/A
Prot_header	usProtCommand	DLI_PROT_SET_BUF_SIZE	DLI_PROT_SET_BUF_SIZE
	iProtModifier	N/A	*See Note 3.*
	usProtLinkID	0	0
	fill3	N/A	N/A
	usProtSessionID	Protocol Session ID	Protocol Session ID
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	2	2

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See Appendix A)
3. Protocol status (See Appendix A)

Table B–7: AWS DLI_PROT_CFG_LINK Command and Response

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	N/A
	fill2	N/A	N/A
	iICPSize	16 (size of Prot_header) plus iProtSize	N/A
	usICPCommand	DLI_ICP_CMD_WRITE	DLI_ICP_CMD_READ
	iICPStatus	*See Note 1.*	*See Note 2.*
	usICPParms[0]	0	0
	usICPParms[1]	N/A	N/A
	usICPParms[2]	N/A	N/A
Prot_header	usProtCommand	DLI_PROT_CFG_LINK	DLI_PROT_CFG_LINK
	iProtModifier	N/A	*See Note 3.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	Protocol Session ID	Protocol Session ID
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	*See Note 4.*	*See Note 4.*

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See Appendix A)
3. Protocol status (See Appendix A)
4. Specify the number of configuration word pairs times 8 plus 4; this is the total number of bytes in the configuration data area.

Table B–8: AWS DLI_PROT_CLR_STATISTICS Command and Response

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	N/A
	fill2	N/A	N/A
	iICPSize	16	N/A
	usICPCommand	DLI_ICP_CMD_WRITE	DLI_ICP_CMD_READ
	iICPStatus	*See Note 1.*	*See Note 2.*
	usICPParms[0]	0	0
	usICPParms[1]	N/A	N/A
	usICPParms[2]	N/A	N/A
Prot_header	usProtCommand	DLI_PROT_CLR_STATISTICS	DLI_PROT_CLR_STATISTICS
	iProtModifier	N/A	*See Note 3.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	Protocol Session ID	Protocol Session ID
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	0	0

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See Appendix A)
3. Protocol status (See Appendix A)

Table B–9: AWS DLI_PROT_SET_SIG Command

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	*Note: This command has no response associated with it.*
	fill2	N/A	
	iICPSize	18	
	usICPCommand	DLI_ICP_CMD_WRITE	
	iICPStatus	*See Note 1.*	
	usICPParms[0]	0	
	usICPParms[1]	N/A	
	usICPParms[2]	N/A	
Prot_header	usProtCommand	DLI_PROT_SET_SIG	
	iProtModifier	N/A	
	usProtLinkID	Link Number	
	fill3	N/A	
	usProtSessionID	Protocol Session ID	
	fill4	N/A	
	fill5	N/A	
	iProtSize	2	

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)

Table B–10: AWS DLI_PROT_GET_ICP_REPORT Command and Response

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	N/A
	fill2	N/A	N/A
	iICPSize	16	16 + iProtSize
	usICPCommand	DLI_ICP_CMD_WRITE	DLI_ICP_CMD_READ
	iICPStatus	*See Note 1.*	*See Note 2.*
	usICPParms[0]	0	0
	usICPParms[1]	N/A	N/A
	usICPParms[2]	N/A	N/A
Prot_header	usProtCommand	DLI_PROT_GET_ICP_REPORT	DLI_PROT_GET_ICP_REPORT
	iProtModifier	N/A	*See Note 3.*
	usProtLinkID	N/A	N/A
	fill3	N/A	N/A
	usProtSessionID	Protocol Session ID	Protocol Session ID
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	0	*See Note 4.*

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See Appendix A)
3. Protocol status (See Appendix A)
4. The number of bytes in the ICP configuration report (10)

Table B–11: AWS DLI_PROT_SEND_CRYPTO_RESYNC command

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	*Note: This command has no response associated with it.*
	fill2	N/A	
	iICPSize	16	
	usICPCommand	DLI_ICP_CMD_WRITE	
	iICPStatus	*See Note 1.*	
	usICPParms[0]	0	
	usICPParms[1]	N/A	
	usICPParms[2]	N/A	
Prot_header	usProtCommand	DLI_PROT_SEND_CRYPTO_RESYNC	
	iProtModifier	N/A	
	usProtLinkID	N/A	
	fill3	N/A	
	usProtSessionID	Protocol Session ID	
	fill4	N/A	
	fill5	N/A	
	iProtSize	0	

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)

Table B–12: AWS DLI_PROT_GET_LINK_CFG Command and Response

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	N/A
	fill2	N/A	N/A
	iICPSize	16	16 + iProtSize
	usICPCommand	DLI_ICP_CMD_WRITE	DLI_ICP_CMD_READ
	iICPStatus	*See Note*	*See Note 2.*
	usICPParms[0]	0	0
	usICPParms[1]	N/A	N/A
	usICPParms[2]	N/A	N/A
Prot_header	usProtCommand	DLI_PROT_GET_LINK_CFG	DLI_PROT_GET_LINK_CFG
	iProtModifier	N/A	*See Note 3.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	Protocol Session ID	Protocol Session ID
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	0	*See Note 4.*

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See Appendix A)
3. Protocol status (See Appendix A)
4. The number of bytes in the link configuration report

Table B–13: AWS DLI_PROT_GET_SIG_REPORT Command and Response

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	N/A
	fill2	N/A	N/A
	iICPSize	16	16 + iProtSize
	usICPCommand	DLI_ICP_CMD_WRITE	DLI_ICP_CMD_READ
	iICPStatus	*See Note 1.*	*See Note 2.*
	usICPParms[0]	0	0
	usICPParms[1]	N/A	N/A
	usICPParms[2]	N/A	N/A
Prot_header	usProtCommand	DLI_PROT_GET_SIG_REPORT	DLI_PROT_GET_SIG_REPORT
	iProtModifier	N/A	*See Note 3.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	Protocol Session ID	Protocol Session ID
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	0	*See Note 4.*

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See Appendix A)
3. Protocol status (See Appendix A)
4. The size of the discrete status report (1)

Table B–14: AWS DLI_PROT_GET_STATISTICS Command and Response

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	N/A
	fill2	N/A	N/A
	iICPSize	16	16 + iProtSize
	usICPCommand	DLI_ICP_CMD_WRITE	DLI_ICP_CMD_READ
	iICPStatus	*See Note 1.*	*See Note 2.*
	usICPParms[0]	0	0
	usICPParms[1]	N/A	N/A
	usICPParms[2]	N/A	N/A
Prot_header	usProtCommand	DLI_PROT_GET_STATISTICS	DLI_PROT_GET_STATISTICS
	iProtModifier	N/A	*See Note 3.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	Protocol Session ID	Protocol Session ID
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	0	*See Note 4.*

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See Appendix A)
3. Protocol status (See Appendix A)
4. The size of the statistics report (32).

Table B–15: AWS DLI_PROT_SEND_NORM_DATA Command

Header	Field	Command Value (Write)
ICP_header	fill1	N/A
	fill2	N/A
	iICPSize	16 + iProtSize
	usICPCommand	DLI_ICP_CMD_WRITE
	iICPStatus	*See Note 1.*
	usICPParms[0]	0
	usICPParms[1]	N/A
	usICPParms[2]	N/A
Prot_header	usProtCommand	DLI_PROT_SEND_NORM_DATA *See Note 2.*
	iProtModifier	N/A
	usProtLinkID	Link Number
	fill3	N/A
	usProtSessionID	Protocol Session ID
	fill4	N/A
	fill5	N/A
	iProtSize	Size of data to be sent

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)

2. This command has no immediate response; a matching DLI_PROT_RESP_LOCAL_ACK (Table B–19 on page 110) or DLI_PROT_RESP_NACK (Table B–20 on page 111) is eventually sent by the ICP.

B.3 Response Header Tables

Table B–16 lists the AWS response codes for data-related responses that originate in the ICP. These responses are either unsolicited (created by receipt of data) or associated with an earlier DLI_PROT_SEND_NORM_DATA command.

Table B–16: AWS Response Codes

Response Code	Reference Table	Response Type
DLI_PROT_RECV_DATA	Table B–17 on page 108	unsolicited
DLI_PROT_RECV_DATA_LOST	Table B–18 on page 109	unsolicited
DLI_PROT_RESP_LOCAL_ACK	Table B–19 on page 110	response to earlier DLI_PROT_SEND_NORM_DATA
DLI_PROT_RESP_NACK	Table B–20 on page 111	response to earlier DLI_PROT_SEND_NORM_DATA

On the following pages, Table B–17 through Table B–20 describe the detailed header fields of the above responses.

Table B–17: AWS DLI_PROT_RECV_DATA Response

Header	Field	Response Value (Read)
ICP_header	fill1	N/A
	fill2	N/A
	iICPSize	16 + iProtSize
	usICPCommand	DLI_ICP_CMD_READ
	iICPStatus	***See Note 1.***
	usICPParms[0]	0
	usICPParms[1]	N/A
	usICPParms[2]	N/A
Prot_header	usProtCommand	DLI_PROT_RECV_DATA
	iProtModifier	***See Note 2.***
	usProtLinkID	Link Number
	fill3	N/A
	usProtSessionID	Protocol Session ID
	fill4	N/A
	fill5	N/A
	iProtSize	Size of data received

Notes:

1. DLI returns status (See Appendix A)
2. Protocol status (See Appendix A)

Table B–18: AWS DLI_PROT_RECV_DATA_LOST Response

Header	Field	Response Value (Read)
ICP_header	fill1	N/A
	fill2	N/A
	iICPSize	16 + iProtSize
	usICPCommand	DLI_ICP_CMD_READ
	iICPStatus	*See Note 1.*
	usICPParms[0]	0
	usICPParms[1]	N/A
	usICPParms[2]	N/A
Prot_header	usProtCommand	DLI_PROT_RECV_DATA_LOST
	iProtModifier	*See Note 2.*
	usProtLinkID	Link Number
	fill3	N/A
	usProtSessionID	Protocol Session ID
	fill4	N/A
	fill5	N/A
	iProtSize	*See Note 3.*

Notes:

1. DLI return status (See Appendix A)
2. Protocol status (See Appendix A)
3. Data size of the message received. There have been messages (prior to this data message) that were lost.

Table B–19: AWS DLI_PROT_RESP_LOCAL_ACK Response

Header	Field	Response Value (Read)
ICP_header	fill1	N/A
	fill2	N/A
	iICPSize	N/A
	usICPCommand	DLI_ICP_CMD_READ
	iICPStatus	***See Note 1.***
	usICPParms[0]	0
	usICPParms[1]	N/A
	usICPParms[2]	N/A
Prot_header	usProtCommand	DLI_PROT_RESP_LOCAL_ACK
	iProtModifier	***See Note 2.***
	usProtLinkID	Link Number
	fill3	N/A
	usProtSessionID	Protocol Session ID
	fill4	N/A
	fill5	N/A
	iProtSize	0

Notes:

1. DLI return status (See Appendix A)
2. Protocol status (See Appendix A)

Table B–20: AWS DLI_PROT_RESP_NACK Response

Header	Field	Response Value (Read)
ICP_header	fill1	N/A
	fill2	N/A
	iICPSize	16 + iProtSize
	usICPCommand	DLI_ICP_CMD_READ
	iICPStatus	*See Note 1.*
	usICPParms[0]	0
	usICPParms[1]	N/A
	usICPParms[2]	N/A
Prot_header	usProtCommand	DLI_PROT_RESP_NACK
	iProtModifier	*See Note 2.*
	usProtLinkID	Link Number
	fill3	N/A
	usProtSessionID	Protocol Session ID
	fill4	N/A
	fill5	N/A
	iProtSize	*See Note 3.*

Notes:

1. DLI return status (See Appendix A)
2. Protocol status (See Appendix A)
3. Data size of the message received. There have been messages (prior to this data message) that were lost.

Index

Customer Report Form

We are constantly improving our products. If you have suggestions or problems you would like to report regarding the hardware, software or documentation, please complete this form and mail it to Protogate at 12225 World Trade Drive, Suite R, San Diego, CA 92128, or fax it to (877) 473-0190.

If you are reporting errors in the documentation, please enter the section and page number.

Your Name: _____

Company: _____

Address: _____

_____

_____

Phone Number: _____

Product: _____

Problem or
Suggestion: _____

_____

_____

_____

_____

_____

Protogate, Inc.
Customer Service
12225 World Trade Drive, Suite R
San Diego, CA 92128