Addendum:
Embedded ICP2432
AWS Programmer's Guide

DC 900-1557C

Simpact, Inc.
9210 Sky Park Court
San Diego, CA 92123
August 1998

SIMPACT

Simpact, Inc.
9210 Sky Park Court
San Diego, CA 92123
(619) 565-1865

Addendum: Embedded ICP2432 AWS Programmer's Guide

Printed in the United States of America

Contents

List of Tables

Preface

Purpose of Document

This document describes the installation, initialization, and programming interface required to use Simpact's Asynchronous Wire Service (AWS) protocol with the embedded ICP2432 in the Digital UNIX environment.

Reference Document

This document is an addendum to the *Asynchronous Wire Service (AWS) Programmer's Guide* (DC 900-1324) which is your main reference for the details of the AWS protocol.

Intended Audience

This document should be read by application programmers developing host software to interface with Simpact's Embedded ICP2432 AWS product. You must understand I/O and system services on your host computer.

Required Equipment

The Embedded ICP2432 AWS software runs on Simpact's ICP2432 programmable front-end processor, an intelligent communications processor (ICP) for the peripheral component interconnect (PCI) bus. The ICP2432 can be inserted into the backplane of any PCI-bus computer.

Organization of Document

Chapter 1 describes the installation, initialization, and testing procedures.

Chapter 2 is the "mini" host driver programmer's guide.

Chapter 3 describes how the host application performs AWS protocol processing in the embedded ICP2432 environment.

Revision History

The revision history of the *Addendum: Embedded ICP2432 AWS Programmer's Guide*, Simpact document DC 900-1557C, is recorded below:

Document Revision	Release Date	Description
DC 900-1557A	September 1997	Initial release
DC 900-1557B	June 1998	Minor modifications in Chapter 1 and Chapter 2
DC 900-1557C	August 1998	Add Clear Link Statistics command (Table 3–9 on page 32) and Get Statistics Report (Table 3–14 on page 37)

Customer Support

If you are having trouble with any Simpact product, call us at 1-800-275-3889 Monday through Friday between 8 a.m. and 5 p.m. Pacific time.

You can also fax your questions to us at (619)560-2838 or (619)560-2837 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 Installation, Initialization, and Testing

This chapter describes how to install the ICP2432 software, initialize the ICP2432, and test the product installation.

1.1 ICP2432 Software Installation Procedure

To install Simpact's STREAMS Digital UNIX driver for the ICP2432, there are four steps which are described in the next sections:

1. Install the product kit
2. Rebuild the UNIX kernel
3. Reboot the machine
4. Create the special device files for the ICP2432

1.1.1 Installing The Product Kit

The product kit is loaded by using the **setld** command with the **-l** option. Load the magnetic tape containing the product kit into the machine's tape drive, log into the system as **root**, and type the following command (the **#** character is the system prompt).

```
# setld -l device
```

where *device* is the device file for the tape driver containing the product kit (e.g. **/dev/nrmt0h**). The **setld** command walks you through the installation process.

When the command completes successfully, the product kit has been installed onto your system. The following files will have been created:

/usr/local/freeway/client/osf1/bin/icpload — The ICP2432 download program

/usr/local/freeway/client/osf1/bin/mkdev — The executable script file which builds the special device files for the ICP2432

/usr/local/freeway/client/test/aws/awstest.c — A sample program to verify correct installation of the hardware

/usr/local/freeway/client/test/aws/awstest.h — Header file used by **awstest.c**

/usr/local/freeway/client/test/aws/makefile — The make file for the sample program

/usr/local/freeway/icpcode/icp2432/osimpact/xio_2432.mem — The OS/Impact operating system's binary image to be downloaded onto the ICP2432

/usr/opt/ICPPCIBASE101 — The ICP2432 driver installation directory. All files in this directory and any subdirectories are part of the ICP2432 device driver.

1.1.2 Rebuilding the UNIX Kernel

After the product kit is installed, the UNIX kernel must be rebuilt so that it includes the new ICP2432 device driver. The following set of instructions rebuilds the UNIX kernel:

```
# doconfig -c SYSNAME
# mv /vmunix /vmunix.sav
# cp /sys/SYSNAME/vmunix /vmunix
```

where *SYSNAME* is the system name specified in uppercase. For example, if the system name is **beavis**, then the above sequence of commands would translate to:

```
# doconfig -c BEAVIS
# mv /vmunix /vmunix.sav
# cp /sys/BEAVIS/vmunix /vmunix
```

The **doconfig** command builds the new UNIX kernel, putting the new image in the file **/sys/*SYSNAME*/vmunix**. Before making this file the new system image (using the **cp** command above), it is a good idea to save the currently existing kernel image, which is what the second instruction above does. After the copy is performed, the system is ready to be rebooted with the new kernel image.

Note

When prompted by **doconfig**, it is not necessary to modify the configuration file.

1.1.3 Rebooting the Machine

Either the **reboot** or **shutdown** command can be used to reboot the system. The **reboot** command causes the machine to be rebooted immediately, whereas **shutdown** is friendlier, giving any users that are logged on an opportunity to log out before the system reboots. The following command waits five minutes before performing a controlled shutdown and reboot.

```
# shutdown -r +5 "Rebooting to install ICP2432 device driver"
```

The **-r** option causes an automatic reboot after the shutdown.

1.1.4 Creating the Special Device Files

After the system is rebooted, the special device files in the **/dev** directory need to be created in order to access the ICP2432. This is done by executing the **mkdev** script file found in the **/usr/local/freeway/client/osf1/bin** directory. This script file requires one parameter, and that is the major device number for the ICP2432. This can be found by executing the following command:

```
# strsetup -c
```

The major device number for the ICP2432 can be found in the **icp** entry in the table produced by the command. After the major device number has been determined, the special files are created with the following command (assuming the current directory is **/usr/local/freeway/client/osf1/bin**).

```
# mkdev major_device_number
```

When the script file completes, the new directory **/dev/icp0** contains the special files. The ICP2432 is now ready to be initialized.

1.2 Initializing the ICP2432

This section describes how to initialize the ICP2432. There are three steps to initialize the board, all of which are performed by the **icpload** program included in the product kit.

- Reset the ICP2432
- Download Simpact's OS/Impact operating system and protocol software onto the board
- Tell the ICP2432 to jump to the protocol software's **INIT** procedure

1.2.1 Downloading the ICP2432 Using the icpload Program

The **icpload** program resets and downloads the ICP2432. This program is part of the product kit, and can be found in the **/usr/local/freeway/client/osf1/bin** directory. The ICP2432 can be initialized by making this directory the current directory, and executing the **icpload** program with the following syntax:

```
icpload device loadfile
```

where *device* is the device name (e.g. **/dev/icp0/0**) and *loadfile* is the name of the load file for the protocol. The load file is in a separate product kit containing the protocol software. Minor device '**0**' (zero) should always be used when downloading the protocol software onto the board. In the following example, assume that Simpact's AWS protocol is being downloaded onto the ICP2432, and that the load file for the protocol is **awsload.txt** (contained in the Freeway boot directory, **/usr/local/freeway/boot**). Also assume that the current directory is set to **/usr/local/freeway/client/osf1/bin**. The following command downloads the protocol software.

```
icpload /dev/icp0/0 ../../../boot/awsload.txt
```

After the **icpload** program completes, the ICP is on-line and ready for normal operation.

1.2.2 The Protocol Load File

The load file associated with a protocol is described in greater detail in the documentation accompanying the protocol software's product kit. This is merely an overview.

The load file is a regular text file that contains a sequence of commands for the **icpload** program to process. Two commands are recognized by **icpload** (commands are case-sensitive):

LOAD — Download a file onto the ICP2432. The syntax for the **LOAD** command is:

```
LOAD file address
```

where *file* indicates the pathname of the file to download, and *address* indicates the starting hexadecimal address (*in the ICP2432's memory space*) where the file gets loaded.

INIT — Command the ICP2432 to jump to the protocol software's **INIT** procedure. The syntax for the **INIT** command is:

```
INIT address
```

where *address* indicates the starting hexadecimal address (*in the ICP2432's address space*) where the **INIT** procedure is located.

The contents of a typical load file are:

```
LOAD /usr/local/freeway/icpcode/icp2432/osimpact/xio_2432.mem     801200
LOAD /usr/local/freeway/icpcode/icp2432/protocols/aws_fw_2432.mem 818000
INIT                                                              818000
```

This loads the OS/Impact operating system (**xio_2432.mem**) and the AWS protocol software (**aws_fw_2432.mem**) onto the ICP2432, and executes the protocol software's **INIT** procedure.

Users must ensure that the pathnames of the download files are correct within the load file. Also, pathnames are case-sensitive under Digital UNIX.

1.3 Testing the ICP2432 Installation

This section describes how to verify that the ICP2432 has been properly installed and downloaded. For purposes of this discussion, assume that the test program is **awstest**. The make file to build the test program resides in the /usr/local/freeway/client/test/aws directory.

Build the test program as follows:

```
# make
```

This make file compiles and links the test program and places the executable program in the /usr/local/freeway/client/osf1/bin directory.

Run the test program as follows:

```
# awstest transfers device rec_size
```

where **transfers** is the number of records to be sent by links 0 and 1 of the ICP2432 (local loopback cables must be connected from link 0 to link 1). **Device** is the ICP device number (e.g. /dev/icp0/0 is device 0), and **rec_size** should be greater than 8 and less than 1024.

Example:

```
# awstest 100 0 64
```

Chapter 2

Host Driver Programmer's Guide

This chapter gives a brief overview of the Digital UNIX system call interface used to communicate with the ICP2432. It is not meant as a tutorial for systems programming in the Digital UNIX environment; it is assumed the reader is familiar with the functions described below.

Chapter 3 discusses the commands and responses exchanged between an application program on the host system and the protocol software on the ICP2432, the buffer structure, the header formats, and the contents of the headers. The reader should refer to the *Asynchronous Wire Service (AWS) Programmer's Guide* for information on these items, as well as the information in Chapter 3 of this document. Chapter 2 is concerned only with how user buffers are exchanged between the application and the protocol software, not the contents of those buffers.

2.1 Opening the ICP

Before a user application can exchange data with the ICP2432, it must obtain a file descriptor for the device. This is accomplished with the **open** system call, as shown in the following example.

```
int FileDescriptor;
...
if ( ( FileDescriptor = open( "/dev/icp0/0", O_RDWR ) ) < 0 )
{
   /* Error processing. */
}
```

The **open** function returns a file descriptor for the device indicated in the first parameter. If the file descriptor is less than zero, an error occurred and the device is not open.

The file descriptor is used in all subsequent system calls which request services with the ICP2432. Application programs control the ICP2432 by writing commands down to the device and reading back the responses to those commands, so the device should be opened with read-write mode, as indicated in the second parameter in the above example.

2.2 Writing to the ICP

Writing to the ICP is accomplished with the **write** system call. After the I/O buffer is constructed, it is sent to the ICP2432 as shown in the following example.

```
void    *Bfr    = I/O buffer address;
ssize_t  Count;
size_t   Size   = Size of data transfer;
...
Count = write( FileDescriptor,
               Bfr,
               Size );
if ( Count < 0 )
{
   /* Error processing. */
}
```

The parameters to **write** are as follows:

- **FileDescriptor** contains the file descriptor which was returned in the **open** call
- **Bfr** is the address of the I/O buffer
- **Size** is the number of bytes to transfer

When the function completes, the variable **Count** contains the actual number of bytes transferred. If this value is negative, then an error occurred and must be accounted for. Consult the *Asynchronous Wire Service (AWS) Programmer's Guide* for how the ICP2432 software handles the case of partial data transfer (**Count** differs from **Size** after **write** completes)… most protocols return an error notification packet, so an explicit comparison of **Count** and **Size** may not be required.

2.3 Reading from the ICP

Reading from the ICP is accomplished with the **read** system call, as shown in the following example.

```
void    *Bfr    = I/O buffer address;
ssize_t  Count;
size_t   Size   = Size of I/O buffer;
...
memset( Bfr,        /* Clear the I/O buffer (not required, but nice). */
        (char) 0,
        Size );
Count = read( FileDescriptor,
              Bfr,
              Size );
if ( Count < 0 )
{
   /* Error processing. */
}
```

The parameters to **read** are the same as for **write**... with the exception that the **Size** parameter indicates the size of the entire I/O buffer instead of the requested transfer size.

When the function completes, the variable **Count** contains the actual number of bytes transferred. If this value is negative, then an error occurred and must be accounted for.

2.4 Closing the ICP

When an application has finished using the ICP2432, it must close any open file handles it has to the device. This is accomplished with the **close** system call as shown in the following example.

```
if ( close( FileDescriptor ) < 0 )
{
   /* Error processing. */
}
```

Again, **FileDescriptor** is the file descriptor that was returned by the **open** call when the device was first opened. If **close** returns a negative value, then an error occurred and must be accounted for; although at that point there is not much that an application can

do. (It is still good practice to check the return value, however. For example, it could help detect programmatic errors during development of the application.)

2.5 IOCTL Commands to the ICP

The Simpact ICP2432 driver supports three IOCTL functions. However, all of these are used during board initialization to reset and download the ICP2432, and are of no interest to the general user. The **icpload** program included in the product kit is used to download the ICP2432. If knowledge of these IOCTL commands is imperative for application development, contact Simpact's Customer Support (see page 8).

Chapter

3 AWS Embedded ICP2432 Protocol Processing

3.1 Introduction

A host application that uses the ICP2432 AWS protocol in an embedded environment must use the ICP2432 driver, and is responsible for processing all write and read requests. The *Asynchronous Wire Service (AWS) Programmer's Guide* describes most of the features and requirements of AWS. Note that since the host application does not call any of the LAN-based (non-embedded) Freeway API functions (such as `dlOpen`), any references to these functions must be ignored. In using the *Asynchronous Wire Service (AWS) Programmer's Guide* as a reference, be aware that the *Raw* operation mode of communication with the ICP is used, that the host application is responsible for processing command acknowledgments (equivalent to DLI configuration parameter "`localAck = no`"), and that the following commands must be implemented by the host application:

- `DLI_ICP_CMD_ATTACH`
- `DLI_ICP_CMD_DETACH`

In addition to normal data that is sent to AWS, the embedded ICP2432 implementation requires that a header section of data, preceding normal data, be included with every message packet sent to the ICP. This header section is made up of two portions, the ICP Header containing eight 16-bit words, and a Protocol Header, also containing eight 16-bit words. These headers contain data that is normally included in the Optional Arguments as described for *Raw* operation of AWS. The tables (in Section 3.3 and Section 3.4) describe which values the ICP and Protocol Headers must contain for each different protocol command. Table elements that have names that are the same as the

name of the Optional Argument elements are identical in function as described in the *Asynchronous Wire Service (AWS) Programmer's Guide*. Names of elements that start with "fill" can be ignored. There are two new values that must exist in these headers that are not defined in the Optional Arguments, `iICPSize` and `iProtSize`. The `iICPSize` field is always the size of the Protocol Header (16) plus the value in `iProtSize`. The `iProtSize` field is the number of bytes that immediately follows the Protocol Header. Many protocol command messages have no data, and thus `iProtSize` should be zero (0). Command messages that provide data to the protocol, such as to send normal data (`DLI_PROT_SEND_NORM_DATA`), must set `iProtSize` to the number of bytes of data to be transferred. Refer to the *Asynchronous Wire Service (AWS) Programmer's Guide* for a description of the message data fields associated with each command. The size of the message buffer that is provided to the ICP2432 driver must be `iICPSize` plus the size of the ICP Header (16). Thus, if the size of data that is to be sent to the ICP is 20, then `iProtSize` is 20, `iICPSize` is 36, and the driver buffer message size is 52.

Note

The AWS protocol takes care of network byte ordering for the Protocol Header and data fields (and thus of any necessary byte swapping of 16-bit words that must occur if the host is a Little-Endian processor), on both messages sent and received. However, it is the responsibility of the host application to insure that messages that are sent or received process the ICP Header in network order; that is, it must byte swap the eight 16-bit words in the ICP Header if the host is a Little-Endian processor (i.e. a VAX).

A host application that operates in the embedded environment must process its own "session management." That is, it must define the relationship between links, ICP nodes, and sessions. For AWS, each link that is to be accessed has a node assigned by the host application, normally link number plus 3 (it must be at least 3 and less than 127). In addition, the host application must save a "session id" returned by the ICP on a suc-

cessful “attach” to a link/node. All subsequent writes to the ICP must contain this session ID in the Protocol Header.

3.2 AWS Protocol Processing for the Embedded ICP2432

3.2.1 Session Attach

The client application must process a `DLI_ICP_CMD_ATTACH` command and subsequent response for each node and link that is be accessed. The session ID (`usProtSessionID`) field in the Protocol Header must be saved and placed in every subsequent Protocol Header command message. See Table 3–3 on page 26.

3.2.2 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 3–7 on page 30.

3.2.3 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 3–8 on page 31.

3.2.4 Link Enabling

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 3–5 on page 28.

3.2.5 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 `DLI_ICP_CMD_WRITE`. See Table 3–15 on page 38.

The client application is normally implemented to allow asynchronous receipt of both the acknowledgment to the `DLI_PROT_SEND_NORM_DATA`, which is `DLI_PROT_RESP_LOCAL_ACK`; and receipt of data received at the data link, which is `DLI_PROT_RECV_DATA`. Note that for both messages, the `usICPCommand` field of the received ICP Header is `DLI_ICP_CMD_READ`. See Table 3–15 on page 38 and Table 3–16 on page 39.

There are other protocol commands (and subsequent responses) that can be sent to or received from the ICP. See the *Asynchronous Wire Service (AWS) Programmer's Guide* and the following command tables for a further description.

3.2.6 Link Disabling

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 3–6 on page 29.

3.2.7 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 3–4 on page 27.

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.

3.3 Command and Response Header Tables

Table 3–1 lists the AWS command /response codes. The required header field values for both the ICP Header and the Protocol Header are detailed in the referenced tables.

Table 3–1: AWS Command/Response Codes

Command/Response Code	Reference Table
DLI_ICP_CMD_ATTACH	Table 3–3 on page 26
DLI_ICP_CMD_DETACH	Table 3–4 on page 27
DLI_ICP_CMD_BIND	Table 3–5 on page 28
DLI_ICP_CMD_UNBIND	Table 3–6 on page 29
DLI_PROT_SET_BUF_SIZE	Table 3–7 on page 30
DLI_PROT_CFG_LINK	Table 3–8 on page 31
DLI_PROT_CLR_STATISTICS	Table 3–9 on page 32
DLI_PROT_SET_SIG	Table 3–10 on page 33
DLI_PROT_GET_ICP_REPORT	Table 3–11 on page 34
DLI_PROT_GET_LINK_CFG	Table 3–12 on page 35
DLI_PROT_GET_SIG_REPORT	Table 3–13 on page 36
DLI_PROT_GET_STATISTICS_REPORT	Table 3–14 on page 37
DLI_PROT_SEND_NORM_DATA	Table 3–15 on page 38

Section 3.4 on page 39 describes the header information for the responses relating to data receipt. The response codes are listed in Table 3–2.

Table 3–2: AWS Response Codes

Response Code	Reference Table
DLI_PROT_RECV_DATA	Table 3–16 on page 39
DLI_PROT_RECV_DATA_LOST	Table 3–17 on page 40
DLI_PROT_RESP_NACK	Table 3–18 on page 41

Table 3–3: 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
	ilCPSize	16	16
	usICPCommand	DLI_ICP_CMD_ATTACH	DLI_ICP_CMD_ATTACH
	ilCPStatus	*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 the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. This returned Session ID must be provided on all subsequent writes for this link/node (see Note 4)
4. Node (node number = link number + 3)

Table 3–4: 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
	ilCPSize	16	16
	usICPCommand	DLI_ICP_CMD_ATTACH	DLI_ICP_CMD_ATTACH
	ilCPStatus	*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_ATTACH	DLI_ICP_CMD_ATTACH
	iProtModifier	N/A	*See Note 4.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	*See Note 3.*	*See Note 3.*
	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 the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. The usProtSessionID that was returned on the DLI_ICP_CMD_ATTACH command.
4. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)

Table 3–5: 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
	`ilCPSize`	16	N/A
	`usICPCommand`	`DLI_ICP_CMD_BIND`	`DLI_ICP_CMD_BIND`
	`ilCPStatus`	*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 4.*
	`usProtLinkID`	Link Number	Link Number
	`fill3`	N/A	N/A
	`usProtSessionID`	*See Note 3.*	*See Note 3.*
	`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 the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. The `usProtSessionID` that was returned on the `DLI_ICP_CMD_ATTACH` command.
4. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)

Table 3–6: 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
	ilCPSize	16	N/A
	usICPCommand	DLI_ICP_CMD_UNBIND	DLI_ICP_CMD_UNBIND
	ilCPStatus	*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 4.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	*See Note 3.*	*See Note 3.*
	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 the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. The usProtSessionID that was returned on the DLI_ICP_CMD_ATTACH command.
4. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)

Table 3–7: 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
	ilCPSize	18	18
	usICPCommand	DLI_ICP_CMD_WRITE	DLI_ICP_CMD_WRITE
	ilCPStatus	*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 4.*
	usProtLinkID	0	0
	fill3	N/A	N/A
	usProtSessionID	*See Note 3.*	*See Note 3.*
	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 the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. The usProtSessionID that was returned on the DLI_ICP_CMD_ATTACH command.
4. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)

Table 3–8: 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
	ilCPSize	16 (size of Prot_header) plus iProtSize	N/A
	usICPCommand	DLI_ICP_CMD_WRITE	DLI_ICP_CMD_WRITE
	ilCPStatus	*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 4.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	*See Note 3.*	*See Note 3.*
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	*See Note 5.*	*See Note 5.*

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. The usProtSessionID that was returned on the DLI_ICP_CMD_ATTACH command.
4. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
5. Specify the number of configuration word pairs times two plus 2; this is the total number of bytes in the configuration data area.

Table 3–9: 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
	ilCPSize	16	N/A
	usICPCommand	DLI_ICP_CMD_WRITE	DLI_ICP_CMD_WRITE
	ilCPStatus	*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 4.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	*See Note 3.*	*See Note 3.*
	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 the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. The usProtSessionID that was returned on the DLI_ICP_CMD_ATTACH command.
4. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)

Table 3–10: 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	
	ilCPSize	18	
	usICPCommand	DLI_ICP_CMD_WRITE	
	ilCPStatus	*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	*See Note 2.*	
	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)
2. The usProtSessionID that was returned on the DLI_ICP_CMD_ATTACH command.

Table 3–11: 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_WRITE
	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 4.*
	usProtLinkID	N/A	N/A
	fill3	N/A	N/A
	usProtSessionID	*See Note 3.*	*See Note 3.*
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	0	*See Note 5.*

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. The usProtSessionID that was returned on the DLI_ICP_CMD_ATTACH command.
4. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
5. The number of bytes in the ICP configuration report (10)

Table 3–12: 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
	ilCPSize	16	16 + iProtSize
	usICPCommand	DLI_ICP_CMD_WRITE	DLI_ICP_CMD_WRITE
	ilCPStatus	*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_LINK_CFG	DLI_PROT_GET_LINK_CFG
	iProtModifier	N/A	*See Note 4.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	*See Note 3.*	*See Note 3.*
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	0	*See Note 5.*

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. The usProtSessionID that was returned on the DLI_ICP_CMD_ATTACH command.
4. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
5. The number of bytes in the link configuration report

Table 3–13: 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
	ilCPSize	16	16 + iProtSize
	usICPCommand	DLI_ICP_CMD_WRITE	DLI_ICP_CMD_WRITE
	ilCPStatus	*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 4.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	*See Note 3.*	*See Note 3.*
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	0	*See Note 5.*

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. The usProtSessionID that was returned on the DLI_ICP_CMD_ATTACH command.
4. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
5. The size of the discrete status report (1)

Table 3–14: AWS DLI_PROT_GET_STATISTICS_REPORT Command and Response

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	fill1	N/A	N/A
	fill2	N/A	N/A
	ilCPSize	16	16 + iProtSize
	usICPCommand	DLI_ICP_CMD_WRITE	DLI_ICP_CMD_WRITE
	ilCPStatus	*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_REPORT	DLI_PROT_GET_STATISTICS_REPORT
	iProtModifier	N/A	*See Note 4.*
	usProtLinkID	Link Number	Link Number
	fill3	N/A	N/A
	usProtSessionID	*See Note 3.*	*See Note 3.*
	fill4	N/A	N/A
	fill5	N/A	N/A
	iProtSize	0	*See Note 5.*

Notes:

1. Zero (0) for Big Endian clients (i.e. SunOS)
   0x4000 for Little Endian clients (i.e. DEC)
2. DLI return status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. The usProtSessionID that was returned on the DLI_ICP_CMD_ATTACH command.
4. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
5. The size of the statistics report (32).

Table 3–15: AWS `DLI_PROT_SEND_NORM_DATA` Command and Response

Header	Field	Command Value (Write)	Response Value (Read)
ICP_header	`fill1`	N/A	N/A
	`fill2`	N/A	N/A
	`ilCPSize`	16 + `iProtSize`	N/A
	`uslCPCommand`	`DLI_ICP_CMD_WRITE`	`DLI_ICP_CMD_WRITE`
	`ilCPStatus`	*See Note 1.*	*See Note 2.*
	`uslCPParms[0]`	0	0
	`uslCPParms[1]`	N/A	N/A
	`uslCPParms[2]`	N/A	N/A
Prot_header	`usProtCommand`	`DLI_PROT_SEND_NORM_DATA`	`DLI_PROT_SEND_NORM_DATA`
	`iProtModifier`	N/A	*See Note 4.*
	`usProtLinkID`	Link Number	Link Number
	`fill3`	N/A	N/A
	`usProtSessionID`	*See Note 3.*	*See Note 3.*
	`fill4`	N/A	N/A
	`fill5`	N/A	N/A
	`iProtSize`	Size of data to be sent	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 the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. The `usProtSessionID` that was returned on the `DLI_ICP_CMD_ATTACH` command.
4. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)

3.4 Response Header Tables

Table 3–16 through Table 3–18 describe the responses associated with the receipt of data.

Table 3–16: AWS DLI_PROT_RECV_DATA Response

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

Notes:

1. DLI return status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
2. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)

Table 3–17: AWS DLI_PROT_RECV_DATA_LOST Response

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

Notes:

1. DLI return status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
2. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. Data size of the message received. There have been messages (prior to this data message) that were lost.

Table 3–18: AWS DLI_PROT_RESP_NACK Response

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

Notes:

1. DLI return status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
2. Protocol status (See the *Asynchronous Wire Service (AWS) Programmer's Guide, Appendix A*)
3. Size of data that was sent before the transmission was aborted.

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 Simpact at 9210 Sky Park Court, San Diego, CA 92123, or fax it to (619) 560-2838.

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: _____

_____

_____

_____

_____

_____

Simpact, Inc.
Customer Service
9210 Sky Park Court
San Diego, CA 92123