ICP2432 User's Guide for OpenVMS Alpha

DC 900-1511B

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

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

ICP2432 User's Guide for OpenVMS Alpha

Printed in the United States of America

Contents

List of Figures

List of Tables

Preface

Purpose of Document

This document describes how to use the ICP2432 intelligent communications processor (ICP) in a peripheral component interconnect (PCI) bus computer running the VMS operating system.

Intended Audience

This document is intended primarily for VMS system managers and applications programmers.

Organization of Document

Chapter 1 is an overview of the product.

Chapter 2 describes how to install the ICP2432 and protocol software in a VMS system.

Chapter 3 describes the application interface to the ICP2432 device driver.

Chapter 4 describes the format of packets written to or read from the ICP.

Chapter 5 describes the `icpload.exe` utility.

Document Conventions

The term "ICP," as used in this document, refers to the physical ICP2432, whereas the term "device" refers to all of the VMS software constructs (device driver, I/O database, and so on) that define the device to the system, in addition to the ICP2432 itself.

Document Revision History

The revision history of the *ICP2432 User's Guide for OpenVMS Alpha*, Simpact document DC 900-1511B, is recorded below:

Document Revision	Release Date	Description
DC 900-1511A	January 1998	Original release
DC 900-1511B	August 1998	Added auto-configuration support Added Single Step Debugger support

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 Product Overview

The Simpact ICP2432 data communications product allows PCIbus computers running the VMS operating system to transfer data to other computers or terminals over standard communications circuits. The remote site need not have identical equipment. The protocols used comply with various corporate, national, and international standards.

The ICP2432 product consists of the software and hardware required for user applications to communicate with remote sites. Figure 1–1 is a block diagram of a typical system configuration. Application software in the VMS system communicates with the ICP2432 by means of the Simpact-supplied device driver.

The ICP controls the communications links for the user applications. The user application writes commands and data to the ICP in the form of packets. The user application also reads responses and data from the ICP in the form of packets. All packets conform to the format described in Chapter 4.

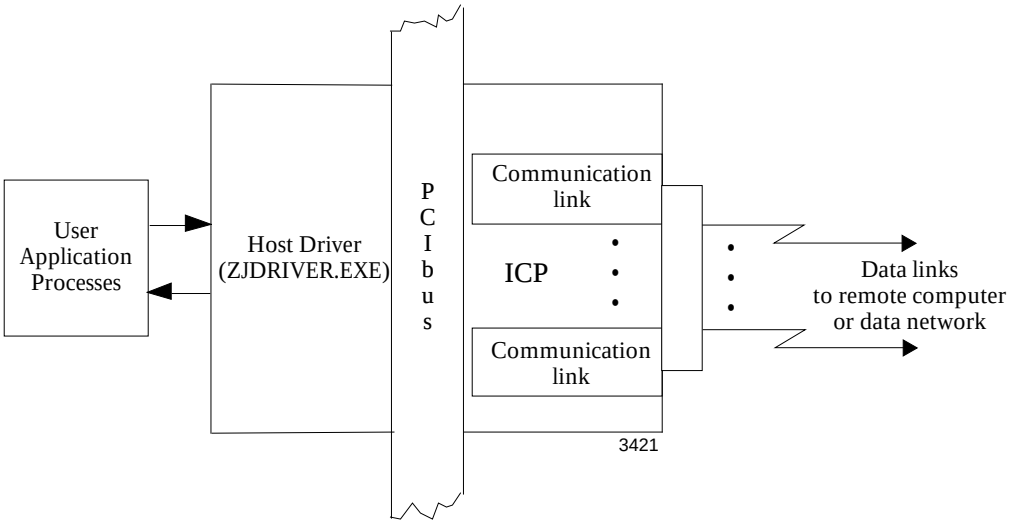

Figure 1–1: Typical Data Communications System Configuration

Chapter

2 Software Installation

A typical software installation contains two or more distribution media packages (tapes, disks, and so on). One package contains the ICP2432 VMS device driver and its related files. The other package contains a specific Simpact protocol and its related files. This chapter describes the installation procedure for both the device driver and the protocol software for VMS systems.

The software installation procedures in this chapter refer to directory names that are used by Simpact's "Freeway" line of server products. The ICP2432 driver and protocol software use the "Freeway" directory tree for building executable images.

2.1 Device Driver Installation Procedure

Step 1:
Verify that you have installed one or more ICP2432 boards in your computer, as described in the *ICP2432 Hardware Installation Guide*.

Step 2:
Insert the ICP2432 installation tape into your VMS computer. The software distribution media contains several VMS `BACKUP` savesets. To install the software from the distribution media onto your VMS computer, use the `VMSINSTAL` utility as described in the following procedure.

Caution

Remember that installing new software overwrites the previous software.

After the distribution media is mounted, the procedure is automated and only requires that you respond to menu prompts. Console displays are shown in `typewriter type` and your responses are shown in **bold type**. Follow each entry with a carriage return. The abbreviation **DDCU** signifies that a device name is required.

You might find it useful to perform the installation at a hardcopy terminal. This provides a printed record that you can use for troubleshooting if needed.

Step 3:
On the host computer, log in to an account that has system-manager privileges.

Step 4:
Insert the protocol distribution media into the appropriate drive. Run VMSINSTAL as follows to copy the files from each distribution media to your VMS computer (V*nnnn* is the current software version number).

```
$ @SYS$UPDATE:VMSINSTAL

OpenVMS AXP Software Product Installation Procedure Vnnnn

It is today's date at current time.

Enter a question mark (?) at any time for help.
```

The computer checks the following conditions:

- Are you logged in to the system manager's account? You should install the software from that account; however, any account with the necessary privileges is acceptable.

- Do you have adequate account quotas for installing software? VMSINSTAL checks for the various quota values.
- Are any users logged on the system? Problems might occur if someone tries to use the system while you are installing a new release of the software.

Step 5:
If there are potential problems with the account quotas, the computer displays:

```
The following account quotas may be too low.
```

The computer lists the account quotas that might be too low. Next, it lists any other active processes.

If any potentially conflicting conditions are noted, the computer gives you the opportunity to stop the installation by displaying the following message:

```
* Do you want to continue anyway [NO]?
```

If you answer **yes**, the computer asks:

```
Are you satisfied with the backup of your system disk [YES]?
```

If you answer **no**, the installation stops so you can save your data before restarting the installation.

Step 6:
If you proceed with the installation, the computer displays the following message. Remember that **DDCU** means a device name.

```
* Where will the distribution volumes be mounted: DDCU:
```

For **DDCU**, substitute a device name such as MUA0, MKA100, DUAl, or something similar.

Step 7:

The computer displays:

```
Enter the products to be processed from the first distribution
volume set.

* Products: *
```

Enter an asterisk (this causes all products to be installed).

Step 8:

The computer displays:

```
* Enter installation options you wish to use (none):
```

Refer to Digital's *VMS Installation Guide* for a list of the `VMSINSTAL` options and how to enter them. Press <return> to select the standard installation options.

Step 9:

The computer displays:

```
This installation procedure will place the files on device
SYS$SYSDEVICE.

* Is this acceptable [Y]? y
```

Press <return> to answer yes (*this is highly recommended*). If you answer **no**, you are prompted to enter the name of a target disk.

Step 10:
The computer displays:

```
This installation procedure will place the product files in
directory [FREEWAY...]
on device ddcu

* Is this acceptable [Y]? y
```

Remember that **DDCU** means a device name. Press <return> to answer yes (*this is highly recommended*). If you answer **no**, you are prompted to enter the name of a directory.

Step 11:
The computer displays:

```
There are no more questions. The installation will proceed.
```

The procedure completes automatically. Depending on the speed of your system, this will take several minutes, then it displays:

```
%VMSINSTAL-I-MOVEFILES, Files will now be moved to their target directories...
Installation of Product Vnnnn completed at current time.
```

Step 12:
The computer displays:

```
Enter the products to be processed from the next distribution volume set.
* Products:
```

If you will be installing another protocol, enter an asterisk (*) to continue. When there are no other distribution sets, enter **exit**. The computer displays:

```
VMSINSTAL procedure done at current time.
```

The ICP2432 software is now installed onto your computer's disk.

2.2 Protocol Software Installation Procedure

The software installation procedures described in this section refer to file names that include a "*ppp*" identifier to indicate a specific protocol. Table 2–1 shows the "*ppp*" identifiers for various protocols. For example, *ppp*_FW_2432.MEM translates to BSC3270_FW_2432.MEM for BSC3270 or X25_FW_2432.MEM for X.25.

Table 2–1: Protocol Identifiers

Protocol or Toolkit	Protocol Identifier (*ppp*)
AUTODIN	autodin[a]
AWS	aws
BSC3270	bsc3270[b]
BSC2780/3780	bsc3780[a]
DDCMP	ddcmp
FMP	fmp
ADCCP NRM	nrm
Protocol Toolkit	sps
Server-resident Application	sra[c]
STD1200A	s12
TACMIL	mil[d]
X.25/HDLC	x25[e]

[a] Except for the readme and release notes, where *ppp* is adn.
[b] Except for the readme, release notes, release history, and load configuration files where *ppp* is bsc for both BSC3270 and BSC2780/3780.
[c] Except for the executable object for the protocol software where *ppp* is sps (sps_fw_2432.mem).
[d] Except for the load configuration files where *ppp* is mil*xxxyyy* (*xxxyyy* identifies the particular TACMIL product designation, distinguished by the unique subset of the full set of military protocols that it contains).
[e] Except for the DLI and TSI configuration files which are apidcfg and apitcfg and the test directory where *ppp* is x25mgr.

The following files are in the FREEWAY directory:

- README.*ppp* provides general information about the protocol software
- RELNOTES.*ppp* provides specific information about the current release of the protocol software
- RELHIST.*ppp* provides information about previous releases of the protocol software

For software releases prior to June 1, 1998, the executable object for the protocol software, *ppp*_FW_2432.MEM, was distributed in the [FREEWAY.ICPCODE.ICPXXXX.PROTOCOLS] directory. For releases after June 1, 1998, this file is in the [FREEWAY.BOOT] directory.

For software releases prior to June 1, 1998, the executable object for the system-services module, XIO_2432.MEM, was distributed in the [FREEWAY.ICPCODE.ICPXXXX.OSIMPACT] directory. For releases after June 1, 1998, this file is in the [FREEWAY.BOOT] directory. The load files provided with protocols with a release date prior to June 1, 1998 contain a fully qualified path for the protocol and XIO image files. Such files should be modified to remove the path to the XIO image. This allows your system to boot the local copy of the XIO image provided in the [FREEWAY.BOOT] directory.

Step 1:
Insert the protocol installation tape into your VMS computer.

Step 2:
To install the protocol software from the distribution media onto your VMS computer, use the VMSINSTAL utility as described in Section 2.1 on page 15.

Caution

Remember that installing new software overwrites the previous software.

2.3 Loading the ICP2432 Driver and Protocol Software

Step 1:

After you have installed the ICP2432 software, load the ICP2432 driver as follows:

```
$ SET DEF DDCU:[FREEWAY.CLIENT.VMS_EMB.BIN]
```

Step 2:

Execute the configuration program:

```
$ @ZJCONFIGURE
```

Step 3:

Set the SIMPACT_ prefix using the SYSMAN utility. First display the current prefix list:

```
$ MCR SYSMAN

SYSMAN> IO SHOW PREFIX

SYSMAN-I-OUTPUT, command execution on node GABIN
SYSMAN-I-IOPREFIX, the current prefix list is: SYS$,DECW$
```

The current prefix list is SYS$,DECW$. The empty string equates to the prefix SYS$.

Next set the SIMPACT_ prefix:

```
SYSMAN> IO SET PREFIX="SYS$,DECW$,SIMPACT_"
```

Step 4:

Configure the ICP cards in the system:

```
$ MCR SYSMAN

SYSMAN> IO AUTOCONFIGURE /SELECT=ZJ*
```

Step 5:

Edit the SYS$MANAGER:SYSCONFIG.COM file. If you want autoconfigure to execute as part of the system startup, add the following line as the last line of the SYS$MANAGER:SYSCONFIG.COM file.

```
@[FREEWAY.CLIENT.VMS_EMB.BIN]SIMPACT_ICBM_INSTALL.COM
```

Step 6:

The SIMPACT_STARTUP.COM file contains commands to download the protocol software. If you did not install the software in the default directory, edit the SIMPACT_STARTUP.COM file to modify the necessary pathnames. The following example is for X25 protocol software.

```
$! Download Protocol Software
$!
$! $ICPLOAD device-name [/reset] -
$!                      [/file=filename] -
$!                      [/address=addr] -
$!                      [/startup=addr]
$!
$!! set verify
$!! ICPLOAD ZJA0 /reset
$!! ICPLOAD ZJA0 -
$!!    /file=SYS$SYSDEVICE:[FREEWAY.BOOT]XIO_2432.MEM -
$!!    /address=%x801200
$!! ICPLOAD ZJA0 -
$!!    /file=SYS$SYSDEVICE:[FREEWAY.BOOT]X25_FW_2432.MEM -
$!!    /address=%x818000
$!! ICPLOAD ZJA0 /startup=%x818000
$!! set noverify
```

Step 7:

Execute SIMPACT_STARTUP.COM to download the protocol software into the ICP2432.

```
$ @SIMPACT_STARTUP

SIMPACT_STARTUP.COM Procedure starting ....
Resetting zja0. This will take about 5 seconds...
Loading Firmware....
$
```

Chapter

3 Application Interface

Application programs running on VMS systems communicate with Simpact protocol software by sending and receiving formatted packets to the ICP2432 device. This is done by issuing VMS queued I/O (QIO) requests to the device driver (`ZJDRIVER`) supplied by Simpact. This chapter describes the use of the VMS system services as they apply to the Simpact device driver.

3.1 Device Driver Interface

The Simpact VMS device driver provides the interface between one or more VMS application programs and the protocol software on the ICP2432. The VMS program builds formatted buffers in user space which consist of one or more headers and a data area. The headers contain information such as commands and response codes that both the program and the protocol software use to determine the type and purpose of each packet. The Simpact VMS device driver has no knowledge of which protocol the ICP2432 is using. It simply provides a logical path to move buffers between AXP and ICP physical memory. The VMS program must do all the interpretation of data within the buffer.

The flow of information between the AXP and ICP generally follows a command/response sequence. For each command sent by the VMS program to the ICP, the program receives a response from the protocol software. There are, however, exceptions to the command/response rule due to the asynchronous nature of communications. For instance, once a link is started, data packets from the remote end of the communication line can be received at any time. These packets are read by the VMS program through

the QIO read path and are not associated with any command sent by the program. Asynchronous line events such as sudden changes in modem control signals are reported in the same way. For this reason, the VMS program should always keep a no-wait read posted to each active link in order to handle any unexpected packets.

Simpact's standard VMS device driver (`ZJDRIVER`) provides an interface to the ICP2432 that is used by several Simpact protocols. Although this driver follows the general design of most other Digital device drivers, there are some functions that may be different from other drivers. The following is a list of important facts about the standard Simpact driver:

- The driver assigns one device name (for example, `ZJA0`) for each ICP2432 board. The user program accesses different ICP links through this one device name by using node numbers (described later in this section). Multiple programs can access the same device name.
- Except for download commands, all reads and writes are directed to a node number. Multiple programs can write to the same node number on the same ICP. However, each program accessing the same ICP should read from a different node number.
- Successful completion of a QIO write call simply means that the client buffer (header and data) has been copied from AXP memory to ICP memory. The VMS program must post a separate read to receive confirmation of the command or data.
- If the VMS program is not able to post a QIO read for an incoming message immediately, the message is not lost; if the ICP has available memory, it holds the packet until the read is posted.
- VMS error codes found in the IOSB of the QIO calls are different from protocol error codes found in the protocol header.
- The Simpact driver does not support timer functions such as timed reads.

Your VMS system must have available PCIbus slots in order to use the ICP2432 boards. After the device driver is installed in the VMS system, ICP boards appear as the device names ZJA0, ZJB0. ZJC0, and so on.

The device driver supports the following VMS system service calls for normal program applications:

- SYS$ASSIGN - Assign a channel
- SYS$QIO (IO$READxBLK, IO$WRITExBLK) - Read and write data
- SYS$DASSGN - Close a channel
- SYS$CANCEL - Cancel pending I/O

The device driver supports the following VMS system service calls for ICP download applications:

- SYS$QIO (IO$INITIALIZE) - Reset an ICP
- SYS$QIO (IO$LOADMCODE) - Download an ICP
- SYS$QIO (IO$STARTMPROC) - Start the ICP protocol software

These system services can be accessed from programs written in MACRO-32 assembly language or any high-level language supported by Digital such as C, FORTRAN, PASCAL, ADA, BASIC, and COBOL. The following sections describe the system services that are normally used by a VMS application programmer who is interfacing to an ICP. More detailed information on these system service calls are described in more detail in Section 3.2 on page 31.

3.1.1 Channel Assignment

The VMS application program must assign a channel to the device driver before any I/O can take place. To do this, the program uses the SYS$ASSIGN system service. The format of this system service is shown in Section 3.2.1 on page 31. Once a VMS program

assigns a channel to an ICP, it has access to all communication ports on that ICP. A program can access more than one ICP by assigning a separate channel for each board. Multiple VMS programs can access the same ICP board by assigning channels to the same device name. Read and write operations for each of the programs are kept separate through the use of node numbers (described later in this chapter).

3.1.2 $QIO Interface

On VMS systems, application programs communicate with the ICP protocol software through the use of the $QIO system service. The format of the `SYS$QIO` call as it relates to the ICP device is shown in Section 3.2.4 on page 34. More detailed information on the QIO call and parameters can be found in the *VMS System Services Reference Manual*. The following sections describe parameters that have specific use with ICP protocol applications.

3.1.2.1 I/O Function Code

The I/O function code determines whether the QIO operation is a read or a write. Use `IO$WRITEVBLK` (write virtual block) when writing a buffer to the ICP and `IO$READVBLK` (read virtual block) when reading a buffer from the ICP. No other modifiers are required. The function codes for logical block (`IO$WRITELBLK`, `IO$READLBLK`) and physical block (`IO$WRITEPBLK`, `IO$READPBLK`) are also supported, but are normally not used by ICP programmers.

3.1.2.2 I/O Status Block (IOSB)

The programmer should always check the status field (first word) of the IOSB after each QIO completion. This field returns a VMS completion code or error code that indicates the success of the call or reason for failure. The return codes used by the ICP device driver are described in Section 3.2.4.4 on page 39. Note that these error codes indicate VMS errors only and are different than the protocol error codes that are returned in the data portion of the QIO read. Protocol-specific errors are described in the Simpact programmer's guide for the specific protocol you are using.

The fourth word of the IOSB contains the actual number of bytes transferred for READ operations.

3.1.2.3 Buffer Address and Size (P1 and P2)

The P1 parameter contains the address of the buffer to be transferred to the ICP for WRITE operations or the address of a buffer to receive data from the ICP for READ operations. The address can be an array name or pointer to a data area. The buffer consists of the protocol header(s) followed by an optional data area. If a data area exists, it must immediately follow the protocol header.

For WRITE operations, the P2 parameter equals the total size (in bytes) of the protocol header(s) plus any data that follows the header. The size of the data area must not exceed the maximum buffer size specified by the protocol software or a VMS buffer overflow error occurs. For example, if the maximum ICP buffer size is set to 1024 bytes, the maximum value of the P2 parameter would be the size of the protocol header(s) plus 1024.

For READ operations, the P2 parameter equals the size of the program's read buffer. This buffer must be large enough to accept the protocol header(s) plus largest data area expected from the ICP. Using the above example, the read buffer size would always be header size plus 1024 bytes. When the read completes, the program can obtain the actual number of bytes transferred from the IOSB. Since all ICP data transfers include at least a protocol header, each buffer read from the ICP contains at least the size of that header.

3.1.2.4 Node Numbers (P4)

Once a channel is assigned to the ICP device name, data is directed to individual ports (links) on that ICP through the use of a node number in the P4 parameter of the QIO call. A node number represents a logical full-duplex path to the protocol software on the ICP. The legal values for node numbers in the ICP driver are 1 to 126. Note that this range of numbers starts over again for each ICP device name. For example, node 1 on

ZJA0 is different from node 1 on ZJB0. The P4 parameter is a 32-bit word that contains the read node number in the low byte and the write node number in the next byte, as shown in Figure 3–1. As a general rule, application programs should place the desired node number in both low bytes of the parameter (for example, 0101 hex, 0202 hex, and so on) for all QIO transfers, read or write. The device driver uses the appropriate byte and ignore the other.

Note

Read and write node numbers should not be confused with PCIbus node numbers.

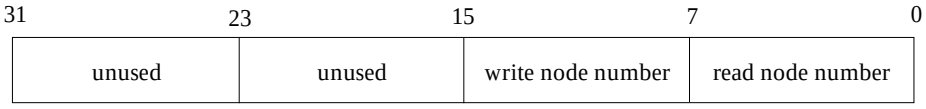

Figure 3–1: P4 Parameter Format

The protocol software on the ICP determines how the device driver node numbers are used. Most of Simpact's current protocol software uses node numbers to form "session" connections through the device driver. Using this method, all writes to the ICP use nodes 1 and 2. All reads from the ICP use nodes 3 to 126. Some Simpact protocols have the ability to revert to an earlier node number scheme used by Simpact's ICP3222 and Digital's Commserver products. This scheme connects a single node number to each ICP port. Whatever node number scheme or protocol you use, it is transparent to the VMS device driver. More information about protocol specifics can be found in Chapter 4.

3.2 Supported VMS System Services

The ICP2432 device driver supports the VMS system services described in the following sections.

3.2.1 SYS$ASSIGN

Before issuing VMS QIO calls, the application must first assign a channel to an ICP with the VMS SYS$ASSIGN call. This call sets up an association between this channel and the ICP on subsequent QIO calls. See the VMS system services manual for a detailed description of SYS$ASSIGN.

Synopsis

```
SYS$ASSIGN ( device_name, &channel [,acmode] [,mbxnam] )
```

Parameters

device_name	ICP device name (for example, ZJA0)
channel	The address of the communication channel that is assigned

Note

Access mode (acmode) and mailbox (mbxnam) are not supported by the ICP2432 device driver.

3.2.2 SYS$CANCEL

To cancel all active or pending read or write requests associated with an I/O channel, the application issues the VMS `SYS$CANCEL` call. See the VMS system services manual for a detailed description.

Synopsis

```
SYS$CANCEL ( channel )
```

Parameters

`channel`	Communication channel

3.2.3 SYS$DASSGN

To terminate its association with an ICP device, the application issues the VMS SYS$DASSGN call for the channel associated with the ICP. See the VMS system services manual for a detailed description of SYS$DASSGN.

Synopsis

```
SYS$DASSGN ( channel )
```

Parameters

channel	Communication channel

3.2.4 SYS$QIO(W)

To issue VMS read or write I/O calls, the client application issues the VMS `SYS$QIOW` or `SYS$QIO` calls (for I/O with, or without wait). See the VMS system services manual for a detailed description of `SYS$QIOW` and `SYS$QIO`.

Synopsis

```
SYS$QIO(W) ( [efn], channel, function [,&iosb] [,&astadr] [,astprm],
             [,p1] [,p2] [,p3] [,p4] [,p5] [,p6] )
```

Parameters

`efn`	Event flag to be set on completion of I/O
`channel`	Communication channel associated with a device
`function`	Supported functions are described in Section 3.2.4.1 through Section 3.2.4.5
`iosb`	Address of the I/O Status Block fields to receive the I/O completion status
`astadr`	Address of an Asynchronous System Trap (AST) routine to be executed on I/O completion
`astprm`	AST parameter passed to the AST routine on I/O completion
`P1–P6`	Optional device- and function-specific I/O request parameters

The ICP2432 device driver supports these function codes, described in the following sections:

1. `IO$_INITIALIZE[|IO$M_NOWAIT]`
2. `IO$_LOADMCODE`
3. `IO$_STARTMPROC`

4. IO$_READVBLK, IO$_READLBLK, IO$_READPBLK

5. IO$_WRITEVBLK, IO$_WRITELBLK, IO$_WRITEPBLK

All functions are handled as direct I/O using DMA transfer.

3.2.4.1 IO$_INITIALIZE[|IO$M_NOWAIT]

The IO$_INITIALIZE function initializes the ICP2432.

Condition Values Returned

SS$_NORMAL	Initialization completed successfully
SS$_CANCEL	Request canceled
SS$_CTRLERR	Request not completed; a fatal error occurred
SS$_TIMEOUT	Request timed out; no response from ICP

The transfer count and the device-specific information of IOSB are not used.

Parameters

None.

Description

If the SS$M_NOWAIT modifier is used, the driver resets the device and returns immediately; it does not wait for initialization to complete.

If the SS$M_NOWAIT modifier is not used, the driver resets the device and initializes the ICP2432 to prepare for downloading the software.

3.2.4.2 IO$_LOADMCODE

The IO$_LOADMCODE function loads a software block onto the ICP2432.

Condition Values Returned

SS$_NORMAL	Request completed successfully
SS$_BADPARAM	Parameter is incorrect
SS$_CANCEL	Request canceled
SS$_ILLBLKNUM	ICP load address is incorrect
SS$_INSFMAPREG	DMA error occurred
SS$_TIMEOUT	Request timed out; no response from ICP

The transfer count and the device-specific information of IOSB are not used.

Parameters

P1	Packet address (must be on a longword boundary)
P2	Packet size (less than 1 megabyte)
P3	0
P4	ICP load address
P5	0
P6	0

Description

The driver accesses user virtual address space (specified by the P1 parameter) to access the packet. The packet must be set on a longword boundary. For details of the ICP load address, see the *ICP2432 Hardware Description and Theory of Operation.*

3.2.4.3 IO$_STARTMPROC

The IO$_STARTMPROC function starts the ICP2432 software.

Condition Values Returned

SS$_NORMAL	Request completed successfully
SS$_BADPARAM	Parameter is incorrect
SS$_CANCEL	Request canceled
SS$_IVMODE	Software was not downloaded
SS$_TIMEOUT	Request timed out; no response from ICP

The transfer count and the device-specific information of IOSB are not used.

Parameters

P1	0
P2	0
P3	0
P4	ICP starting address
P5	0
P6	0

Description

For details of the ICP starting address, see the *ICP2432 Hardware Description and Theory of Operation*.

3.2.4.4 IO$_READVBLK, IO$_READLBLK, and IO$_READVPLK

The IO$_READ*x*BLK function reads a packet to the ICP2432 firmware.

Condition Values Returned

SS$_NORMAL	Request completed successfully
SS$_ACCVIO	Buffer does not allow write access
SS$_BADPARAM	Parameter is incorrect
SS$_BUFFEROVF	Received data is larger than specified buffer
SS$_CANCEL	Request canceled
SS$_IVMODE	Software was not downloaded
SS$_INSFMAPREG	DMA error occurred
SS$_NOSUCHNODE	Node number is incorrect
SS$_TIMEOUT	Request timed out; no response from ICP

The transfer count of IOSB is set, but the device-specific information of IOSB is not used.

Parameters

P1	Packet address (must be on a longword boundary)
P2	Packet size
P3	0
P4	Read and write node numbers
P5	0
P6	0

Description

The driver accesses user virtual address space (specified by the P1 parameter) to access the packet. The packet must be set on a longword boundary.

The read and write node numbers are used for communication between the driver and the ICP2432. The node numbers decide the source and destination of messages. Allowable values are 1 through 126. The read node number of P4 is bit 0 through bit 7. The write node number of P4 is bit 8 through bit 15. See Section 3.1.2.4 on page 29 for more information about node numbers.

3.2.4.5 IO$_WRITEVBLK, IO$_WRITELBLK, and IO$_WRITEPBLK

The IO$_WRITE*x*BLK function writes a packet to the ICP2432 firmware.

Condition Values Returned

SS$_NORMAL	Request completed successfully
SS$_ACCVIO	Buffer does not allow write access
SS$_BADPARAM	Parameter is incorrect
SS$_CANCEL	Request canceled
SS$_IVMODE	Software was not downloaded
SS$_INSFMAPREG	DMA error occurred
SS$_NOSUCHNODE	Node number is incorrect
SS$_TIMEOUT	Request timed out; no response from ICP

The transfer count of IOSB is set, but the device-specific information of IOSB is not used.

Parameters

P1	Packet address (must be on a longword boundary)
P2	Packet size
P3	0
P4	Read and write node numbers
P5	0
P6	0

Description

The driver accesses user virtual address space (specified by the P1 parameter) to access the packet. The packet must be set on a longword boundary.

The read and write node numbers are used for communication between the driver and the ICP2432. The node numbers decide the source and destination of messages. Allowable values are 1 through 126. The read node number of P4 is bit 0 through bit 7. The write node number of P4 is bit 8 through bit 15. See Section 3.1.2.4 on page 29 for more information about node numbers.

3.3 DLI Session Interface

Simpact protocols designed for use on ICP2432 boards use a session-based method of communicating between the client application program and the protocol software on the ICP. This method of communication allows greater flexibility in connecting TCP/IP sockets to individual ICP ports for the Freeway line of servers. Simpact's Data Link Interface (DLI) library uses this session-based interface on both the Freeway server and embedded ICP boards. If you have previously used Simpact protocols on older ICP boards, you will find that the session-based interface differs somewhat from the older interface. As a rule, Simpact protocol image files that begin with `fw` use the DLI session interface.

Inside a Freeway server, a program called `msgmux` manages protocol sessions for all applications. When you use Simpact's standard device driver with a session-based protocol image, your VMS program must take over these session management functions as described in the following subsections.

3.3.1 DLI Session Basics

A session is made up of a logical connection to an ICP protocol. A session simply defines a single connection or "service" to an ICP protocol. A session is started by "attaching" to an ICP port number using a specific access mode. Sessions have different access modes depending on the protocol. Consult your protocol programmer's guide for details.

3.3.2 Use Of Node Numbers (DLI)

When using DLI sessions, all writes to the ICP are performed on nodes 1 and 2. All reads are performed on nodes 3 to 126. When using multiple programs to access the same ICP, different read nodes are used to direct packets to the correct program. The following subsections describe the node numbers in more detail.

3.3.2.1 Node 1

Node 1 is the primary node number to which all data is written. The VMS driver allows multiple programs to write to the same node number. The P4 parameter in the QIO call should be 0x0101 for all writes.

3.3.2.2 Node 2

Node 2 is the alternate write node. Its use varies per protocol. In some documents, node 2 is referred to as the "express node" for sending priority packets to the ICP. However, for most protocols this node is treated the same as node 1. Unless your protocol programmer's guide says otherwise, you should not use node 2 to write data packets to the ICP. You should use node 2 to send the `TERMINATE` command described in Section 3.3.3.3 on page 46.

3.3.2.3 Nodes 3 through 126

Nodes 3 through 126 are used as "read only" nodes and are assigned for use by the `ATTACH` command. Although most Simpact protocols allow multiple sessions per node number, it is easier if your programs assign a separate node number per session. The Freeway server assigns the next lowest available read node number for each new connection it receives. Again, your program does not have to follow this scheme. It would be easier to assign a fixed node number for each ICP port (or service). For example, a session to port 0 would always use node 3, port 1 would use node 4, and so on.

3.3.3 DLI Session Commands

The following commands are used to establish and terminate sessions with Simpact protocols on the ICP. The command codes described here are placed in the command field of the ICP header (see Chapter 4).

3.3.3.1 ATTACH Command

The ATTACH command creates a session between your program and the protocol software on the ICP.

The following values must be placed in the ICP header of the ATTACH command:

Command field = DLI_ICP_CMD_ATTACH
Status field = hex 4000 (this tells the ICP to swap bytes for VMS systems)
Params[0] = read node number

Some protocols may require you to fill in fields in the protocol header portion of the ATTACH command with such things as access mode and protocol type. Consult your protocol programmer's guide for details.

Your program can read the ATTACH response by posting a QIO read to the node number you supplied in the ATTACH command. The two most important values to read from the ATTACH response are the status field of the ICP header and the session ID field of the protocol header. The status field contains 0 if the ATTACH command was successful and an error code if it was unsuccessful. If the ATTACH was successful, the session ID field contains a number associated with this session. This number must be placed in the session ID field of all subsequent writes to this session.

3.3.3.2 DETACH Command

The DETACH command closes an individual session between your program and the protocol software on the ICP. The following values must be placed in the ICP header of the ATTACH command:

ICP Header:
Command field = DLI_ICP_CMD_DETACH
Status field = hex 4000 (this tells the ICP to swap bytes for VMS systems)

Protocol Header:
Session ID field = session ID from ATTACH command

The DETACH command disassociates the protocol session ID from the session, freeing it for use by another program. Your program can read the DETACH response from the read node number specified in the ATTACH command for the session.

3.3.3.3 TERMINATE Command

The TERMINATE command closes all sessions that use a particular read node number. The following values must be placed in the ICP header of the ATTACH command:

Command field = DLI_ICP_CMD_TERM
Status field = hex 4000 (this tells the ICP to swap bytes for VMS systems)
Params[0] = read node number

If there are one or more sessions using a single read node number, the TERMINATE command forces the protocol software to do implied DETACH commands to each open session. The ICP sends a TERMINATE response to the supplied read node.

3.3.4 ICP Discarded Packets

When the protocol software on the ICP receives a packet that has an invalid protocol session ID, it writes the packet back through node 1. For this reason, you may want to have a separate program which reads packets from node 1 and displays the contents.

3.4 Compatibility with older ICP Protocols

Simpact's BSC and FMP protocol software for the ICP2432 now has the capability of emulating the interface used by older ICP products such as the ICP1622, ICP3222, and DEC Commserver. If you already have a VMS program using BSC or FMP on one of these devices, your interface does not have to change. When you send the Set Buffer Size command to node 1 (port 0), the BSC or FMP software automatically detects

(from the size of the packet) that you are using the older style of interface. The protocol software then posts reads on all nodes associated with port numbers in addition to the data ack, control, and trace nodes.

When using this method of interface, each read or write must contain the 8-byte protocol header and commands as described in your original BSC or FMP programmer's guide.

3.5 Protocol Toolkit

If you have purchased Simpact's Protocol Toolkit for the ICP2432, the Sample Protocol Software (SPS) example uses the DLI session interface. The toolkit allows you to change this to whatever style of interface you want to use, however, Simpact recommends that you use the DLI session interface so that you can also use the protocol image in a Freeway environment.

Chapter

4 ICP Packet Formats

This chapter describes the packet formats used by Simpact protocols. The packet formats that are written to the ICP2432 are the same whether the ICP is attached to a Freeway server or a PCI bus in your VMS system. Because most Simpact protocol programmer's guides mention commands and responses as they apply to the Freeway server, this chapter covers both Freeway and device driver use of packets.

4.1 DLI Packet Format

The OpenVMS ICP driver QIO interface provides a block-transfer interface between a client application and the protocol software resident on the embedded ICP. From the application's perspective, these packets consist of message blocks composed of an ICP header structure followed by a protocol header structure followed by an optional data array. Figure 4–1 shows the "C" definition of this ICP packet structure.

When preparing a packet to write to the ICP, the application must initialize the `usICPCount` field with the size in bytes of the `PROT_HDR` structure (16) plus the size of the data array that follows it. After reading a packet from the driver, the application may compute the size of the data array that follows the `PROT_HDR` structure by subtracting 16 from the value of the usICPCount field in the `ICP_HDR` structure.

Note that the `ICP_HDR` structure must be in network byte-order (Big Endian). This means that the VMS program must swap bytes in the ICP header before writing packets to the ICP. The VMS program must also swap bytes in the ICP header after reading each packet from the ICP. The `PROT_HDR` structure remains in the client computer's natural byte order, which is Little Endian for AXP systems.

```
typedef struct _ICP_PACKET
{
    ICP_HDR        icp_hdr;          /* Network-ordered header */
    PROT_HDR       prot_hdr;         /* Host-ordered header */
    char           *data;            /* Variable length data array */
} ICP_PACKET;

typedef struct   _ICP_HDR
{
    unsigned short usICPClientID;    /* Old su_id */
    unsigned short usICPServerID;    /* Old sp_id */
    unsigned short usICPCount;       /* Size of PROT_HDR plus data */
    unsigned short usICPCommand;     /* ICP's command */
             short iICPStatus;       /* ICP's command status */
    unsigned short usICPParms[3];    /* ICP's extra parameters */
} ICP_HDR;

typedef struct _PROT_HDR
{
    unsigned short usProtCommand;    /* Protocol command */
             short iProtModifier;    /* Protocol command's modifier */
    unsigned short usProtLinkID;     /* Protocol link ID */
    unsigned short usProtCircuitID;  /* Protocol circuit ID */
    unsigned short usProtSessionID;  /* Protocol session ID */
    unsigned short usProtSequence;   /* Protocol sequence */
    unsigned short usProtXParms[2];  /* Protocol extra parameters */
} PROT_HDR;
```

Figure 4–1: "C" Definition of ICP Packet Structure

4.2 DLI Optional Arguments

A program using the full DLI library interface to an ICP on a Freeway server is not allowed to write information directly to the ICP and Protocol headers. Instead, Freeway users place the desired values in a DLI_OPT_ARGS structure and the DLI write call moves these values into the proper places in the ICP and Protocol headers. The same applies to DLI read calls. Information from received packets is taken from the ICP and protocol headers and placed in the DLI_OPT_ARGS structure where the program can read it.

Although the DLI library is not used by programs accessing Simpact's standard device driver, it is mentioned here to allow embedded ICP users to see the similarity in packet format when reading Freeway documents. Figure 4–2 shows the DLI_OPT_ARGS structure as it is used in a Freeway environment. Note that the ICP_PACKET structure differs only slightly from the DLI_OPT_ARGS structure. The ICP_PACKET structure omits the three Freeway server header fields (usFWPacketType, usFWCommand, and usFWStatus) and adds one new field (usICPCount). See Table 4–1 for a comparison between the header fields in the DLI_OPT_ARGS and ICP_PACKET structures.

```
typedef struct _DLI_OPT_ARGS
{
    unsigned short usFWPacketType;   /* Server's packet type */
    unsigned short usFWCommand;      /* Server's command sent or received */
    unsigned short usFWStatus;       /* Server's status of I/O operations */
    unsigned short usICPClientID;    /* Old su_id */
    unsigned short usICPServerID;    /* Old sp_id */
    unsigned short usICPCommand;     /* ICP's command */
             short iICPStatus;       /* ICP's command status */
    unsigned short usICPParms[3];    /* ICP's extra parameters */
    unsigned short usProtCommand;    /* Protocol command */
             short iProtModifier;    /* Protocol command's modifier */
    unsigned short usProtLinkID;     /* Protocol link ID */
    unsigned short usProtCircuitID;  /* Protocol circuit ID */
    unsigned short usProtSessionID;  /* Protocol session ID */
    unsigned short usProtSequence;   /* Protocol sequence */
    unsigned short usProtXParms[2];  /* Protocol extra parameters */
} DLI_OPT_ARGS;
```

Figure 4–2: "C" Definition of DLI Optional Arguments Structure

Table 4–1: Comparison of DLI_OPT_ARGS and ICP_PACKET Structures

DLI_OPT_ARGS field name	ICP_PACKET field name	Field Description
usFWPacketType	omitted	Server's packet type
usFWCommand	omitted	Server's command sent or received
usFWStatus	omitted	Server's status of I/O operations
usICPClientID	icp_hdr.usICPClientID	Old su_id
usICPServerID	icp_hdr.usICPServerID	Old sp_id
omitted	icp_hdr.usICPCount	Size of PROT_HDR plus data
usICPCommand	icp_hdr.usICPCommand	ICP's command
iICPStatus	icp_hdr.iICPStatus	ICP's command status[a]
usICPParms[0]	icp_hdr.usICPParms[0]	ICP's extra parameter
usICPParms[1]	icp_hdr.usICPParms[1]	ICP's extra parameter
usICPParms[2]	icp_hdr.usICPParms[2]	ICP's extra parameter
usProtCommand	prot_hdr.usProtCommand	Protocol command
iProtModifier	prot_hdr.iProtModifier	Protocol command's modifier
usProtLinkID	prot_hdr.usProtLinkID	Protocol link ID
usProtCircuitID	prot_hdr.usProtCircuitID	Protocol circuit ID
usProtSessionID	prot_hdr.usProtSessionID	Protocol session ID
usProtSequence	prot_hdr.usProtSequence	Protocol sequence
usProtXParms[0]	prot_hdr.usProtXParms[0]	Protocol extra parameter
usProtXParms[1]	prot_hdr.usProtXParms[1]	Protocol extra parameter
omitted[b]	data	Data array

[a] For writes to the driver, iICPStatus should be 0x4000 because an AXP system is a Little Endian processor.

[b] An application using Simpact's DLI specifies data separately from the DLI_OPT_ARGS structure.

Chapter

5 ICP Utility

This chapter describes how to use the ICPLOAD program to download the ICP-resident application to the ICP and get or set the driver's timeout value for the SingleStep debugger.

ICPLOAD may be used in several different ways:

- As an ordinary VMS executable image, invoked via the DCL RUN command; in this mode, ICPLOAD prompts the user for commands
- As a DCL foreign command; in this mode, qualifiers on the foreign command line dictate the operations to be performed
- As routines called from a user-written program; this allows user-written applications to perform the reset and download operations on an ICP without having to code the special $QIO calls that are required

5.1 ICPLOAD Components

The following files comprise the ICPLOAD program:

ICPLOAD.EXE The program in executable form

ICPLOAD.HLB A help library that may be accessed via the ICPLOAD command HELP

ICPLOAD.OLB An object library which includes the object modules (see Section 5.5 on page 64)

5.2 OS/Impact and Downloaded Files

Software on the ICP2432 executes under control of Simpact's OS/Impact operating system. The OS/Impact system generation procedure typically creates several different files, each of which must be downloaded to the ICP. A *load address* is specified for each file; this defines the address at which each file is to be placed within the ICP's memory. In addition, an *execution address* is specified for the ICP. After all the component files have been downloaded to the ICP, the ICP is informed of the execution address and it transfers control to that location.

If you are using an integrated hardware/software product, the necessary files, load addresses for each, and execution address are described in the product-specific documentation.

A download operation will only succeed if the ICP device is in the proper state to receive it. This is normally ensured by first issuing a reset command to the ICP. If a debugging console is connected to the ICP, there will be a brief delay before the ICP will accept the download.

5.3 Get or Set the Timeout Value

The `ICPLOAD` program can be used to get the driver's current timeout value for the SingleStep debugger or to set a new value. When the timeout value is 0, there is no timeout. We highly recommend that you do not change the default timeout value unless you are using for SingleStep debugger.

5.4 Using ICPLOAD.EXE

5.4.1 Invoking ICPLOAD via the RUN Command

ICPLOAD.EXE may be invoked via a RUN command from VMS's DCL prompt. It will then prompt for its first command, as follows:

```
$ RUN ICPLOAD
ICPLOAD>
```

ICPLOAD may be executed from a command procedure, in which case it reads commands from the lines in the command procedure immediately following the DCL RUN command.

5.4.2 Invoking ICPLOAD as a Foreign Command

ICPLOAD may be invoked as a foreign command as follows:

1. Define a DCL symbol that equates to the complete file specification of ICPLOAD.EXE, with a leading currency symbol ("$"), as follows:

   ```
   $ LDICP == "$ddcu:[dire]ICPLOAD"
   ```

   where ddcu:[dire] represents sufficiently qualified device and directory specifications to find the directory in which ICPLOAD.EXE resides.

2. Invoke ICPLOAD as follows:

   ```
   $ LDICP icp_device [/RESET] [/TIMEOUT=[time_value]] -
         [/FILE=filename] [/ADDRESS=addr] -
         [/STARTUP=addr]
   ```

Each qualifier on the foreign command line corresponds to a command verb accepted by ICPLOAD when it is in command mode. If multiple qualifiers are present, they will be interpreted in the order shown above. Refer to the descriptions of the ICPLOAD commands in Section 5.4.3 for the meanings of the various qualifiers.

In the preceding examples, the symbol `LDICP` was chosen arbitrarily; you can replace this with any symbol you like.

If `ICPLOAD` is invoked as a foreign command without specifying any parameters or qualifiers, the `ICPLOAD>` prompt will be given and the utility will operate as if it had been invoked via a `RUN` command.

5.4.3 ICPLOAD Commands

The general syntax of `ICPLOAD` commands is similar to that of DCL commands. Each command begins with a verb, followed by a device name parameter (except for the `HELP` and `EXIT` commands).

Most commands allow one or more optional qualifiers. All qualifiers are global (that is, their position within the command line is not significant). All command verbs and qualifier names may be abbreviated to the shortest string that is unique in context; four characters are sufficient in all cases.

Table 5–1 briefly lists the commands that are available at the `ICPLOAD>` prompt.

Table 5–1: ICPLOAD Command Summary

Command	Action
`HELP`	Request help on `ICPLOAD` commands
`RESET` *device*	Reset the ICP
`LOAD` *device*	Download a file to the ICP
`START` *device*	Start execution of downloaded code
`GET` *device*	Get the driver's current timeout value (in seconds)
`SET` *device*	Set a new timeout value (in seconds)
`EXIT`	End `ICPLOAD` execution, return to DCL prompt

The usual sequence of commands for downloading an ICP is:

- RESET the device
- LOAD the files to the ICP; the ICP-resident software is usually provided in several different files, and a separate LOAD command is required for each file
- START execution of the ICP-resident software

The following sections describe the RESET, LOAD, START, and HELP commands in detail.

The Format paragraph shows the command with all of its parameters and required qualifiers. All command arguments (values which must be supplied by the operator) are represented by descriptive words in *italics*. These same words are used in the subsequent descriptions of the individual parameters and qualifiers.

The Parameters paragraph gives a detailed description of each parameter. Parameters must be typed in the order shown in the Format paragraph.

The Qualifiers paragraph gives a detailed description of each qualifier that may be specified on the command. You must include all the qualifiers shown in the Format paragraph; the other qualifiers are optional. Qualifiers may be typed in any order.

The Description paragraph provides, where necessary, a more elaborate explanation of the function of the command.

The Example paragraph gives one or more examples of the command's use. Each example is followed by a description of the exact function performed by the illustrated command.

The Foreign Command paragraph shows how to request the same operation when ICPLOAD is invoked as a foreign command. These examples assume that the symbol used to invoke ICPLOAD is LDICP.

5.4.3.1 HELP

This command provides help at the ICPLOAD command prompt.

Format HELP

Parameters None

Qualifiers None

Description The HELP command provides access to the help library ICPLOAD.HLB at the ICPLOAD command prompt. Operation is similar to that for DCL's HELP.

The logical name SIMPACT_HELPFILE must exist and must provide a full file specification (including device and directory name) for ICPLOAD.HLB. SIMPACT_HELPFILE is defined in simpact_startup.com.

Foreign Command None

5.4.3.2 RESET

This command performs a hardware reset of the ICP.

Format RESET *device_name*

Parameters *device_name*
This parameter specifies the ICP device to be reset.

Qualifiers None

Description The RESET command enables the ICP to be downloaded via a subsequent LOAD command.

Your process must have the OPER privilege to use this command.

Example
```
ICPLOAD> RESET ZJB0
```
This command resets the second ICP2432 in the system.

Foreign Command
```
$ LDICP device_name /RESET
```

5.4.3.3 LOAD

This command transfers the ICP-resident software from a file on the client system to the ICP.

Format LOAD *device_name*

Parameters *device_name*
This parameter specifies the ICP device to be downloaded.

Qualifiers /FILE=*file_name*
This qualifier specifies the name of an OS/Impact file.

/ADDRESS=*address*
This qualifier specifies the ICP address at which the file is to be loaded. (If desired, you can use the DCL %X prefix to denote a hexadecimal value.)

Description The LOAD command causes the file(s) named in the qualifiers to be transferred to the ICP.

Your process must have the OPER privilege to use this command.

The ICP-resident software is supplied in several files. Each must be transferred to the ICP in turn, with the appropriate /ADDRESS qualifier.

Example
```
ICPLOAD> LOAD ZJAO/FILE=X25.MEM/ADDR=%X40000
```
This command downloads the software from the X25.MEM file to the ICP known as ZJAO.

Foreign Command
```
$ LDICP device_name /FILE=filename/ADDRESS=address
```

5.4.3.4 START

This command causes the ICP to begin execution of the downloaded software.

Format START *device_name* /STARTUP=*address*

Parameters *device_name*
This parameter specifies the ICP device to be started.

Qualifiers /STARTUP=*address*
This qualifier specifies the starting execution address.
(If desired, you can use the DCL %X prefix to denote a hexadecimal value.)

Description The START command causes the ICP to begin execution of the ICP-resident software at the specified address.

The ICP2432 can receive multiple download images, so an explicit start request is required.

Example
```
ICPLOAD> START ZJA0/STARTUP=%X802000
```
This command causes the ICP known as ZJA0 to begin execution at address 802000 (hex).

Foreign Command
```
$ LDICP device_name /START=address
```

5.4.3.5 GET

This command gets the driver's timeout value (in seconds) for the SingleStep debugger.

Format `GET` *device_name* `/TIMEOUT`

Parameters *device_name*
This parameter specifies the ICP device to get.

Qualifiers `/TIMEOUT`
This qualifier specifies the timeout value in seconds.

Description The `GET` command shows the driver's timeout value for the SingleStep debugger.

Example
```
ICPLOAD> GET ZJAO /TIMEOUT
         3
```
The example command above shows the timeout value to be 3 seconds.

Foreign Command `$ LDICP` ***device_name*** `/TIMEOUT`

5.4.3.6 SET

This command sets the driver's timeout value (in seconds) for the SingleStep debugger.

Format SET *device_name* /TIMEOUT=*timeout_value*

Parameters *device_name*
This parameter specifies the ICP device to set.

Qualifiers /TIMEOUT
This qualifier specifies the timeout value in seconds.

Description The SET command sets the driver's timeout value for the SingleStep debugger.

Example
```
ICPLOAD> SET ZJAO /TIMEOUT=5
```
The example command above sets the timeout value at 5 seconds.

Foreign Command
```
$ LDICP device_name /TIMEOUT=5
```

5.5 ICPLOAD Callable Routines

The `ICPLOAD.OLB` file includes several routines that may be called by a user-written VMS application to affect downloading of an ICP. This section describes how to use these routines.

5.5.1 Conventions

The `ICPLOAD` callable routines are written in C. They may be called from any VMS language that conforms to the Alpha Procedure Calling Standard.

Each of these routines returns either a VMS or RMS system service status code. Integer arguments are passed by value. Character string arguments are passed by reference and must have a terminating null byte (ASCIZ). Unused optional arguments should be zero (passed by value) or zero-length strings (passed by reference).

When linking against `ICPLOAD.OLB`, the DEC C Run Time Library must be included in the link.

5.5.1.1 icpreset

This routine causes an ICP to be reset and prepared for a download operation.

Format `long icpreset (char *device);`

Returns The completion status of the operation (normally `SS$_NORMAL`).

Arguments `device`: the VMS device name of the ICP.

5.5.1.2 icpload

This routine causes a file to be downloaded to the ICP. This routine is typically used more than once per download sequence (once for each OS/Impact component file).

Format

```
long icpload (char *device,
              char *file,
              long address);
```

Returns

The completion status of the operation; normally `SS$_NORMAL`. Refer to Chapter 3 for descriptions of status codes returned by the ICP device driver. If `icpload()` returns an RMS status code, a problem was encountered when opening one of the files specified in the argument list.

Arguments

`device`: the VMS device name of the ICP.

`file`: name of the OS/Impact file.

`address`: starting address within ICP memory to which the file is to be loaded.

5.5.1.3 icpstart

This routine is used to cause an ICP to begin execution of the downloaded code.

Format

```
long icpstart (char *device,
               long address);
```

Returns

The completion status of the operation; normally `SS$_NORMAL`. Refer to Chapter 3 for descriptions of status codes returned by the ICP device driver.

Arguments

`device`: the VMS device name of the ICP.

`address`: starting address for execution.

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