ICP2432 User's Guide for Windows NT®

DC 900-1510C

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

SIMPACT

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

ICP2432 User's Guide for Windows NT

Contents

List of Figures

List of Tables

Preface

Purpose of Document

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

Intended Audience

This document is intended primarily for Windows NT system managers and applications programmers. Application programmers can use Simpact's data link interface (DLI) to interface to the ICP2432 device driver. The DLI provides `dlInit`, `dlOpen`, `dlClose`, `dlWrite`, `dlRead`, and related functions for accessing the ICP2432. Refer to Chapter 3 for details.

Organization of Document

Chapter 1 is an overview of the product.

Chapter 2 describes how to install the ICP2432 software in a Windows NT system. This chapter is of interest primarily to system managers.

Chapter 3 describes the Windows NT embedded interface to Simpact's data link interface (DLI). This chapter supplements the *Freeway Data Link Interface Reference Guide* and is of interest primarily to programmers who are either porting an existing application (currently operational in the Freeway server environment) to the embedded envi-

ronment (for example, the PCI ICP2432) or who are developing an initial application to the DLI in the embedded environment.

Chapter 4 describes the Win32 interface to the ICP2432 device driver.

Appendix A describes Simpact's ICPTool for Windows NT which supports the software installation procedure in Chapter 2 and provides a graphical user interface to the ICP command-line tools.

Appendix B describes debug support.

Appendix C describes the loopback test procedure for the ADCCP NRM protocol.

Appendix D describes the loopback test procedure for the Asynchronous Wire Service (AWS) protocol.

Appendix E describes the loopback test procedure for the BSC protocols.

Appendix F describes the loopback test procedure for the FMP toolkit.

Appendix G describes the loopback test procedure for the protocol toolkit.

Appendix H describes the loopback test procedure for the STD1200A protocol.

Appendix I describes the loopback test procedure for the X.25 protocol.

Document Conventions

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

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

Program code samples are written in the “C” programming language.

Document Revision History

The revision history of the *ICP2432 User’s Guide for Windows NT*, Simpact document DC 900-1510C, is recorded below:

Document Revision	Release Date	Description
DC 900-1510A	November 1997	Original release
DC 900-1510B	March 1998	General enhancements; see `freeway\relnotes`.
DC 900-1510C	July 1998	Added Chapter 4, a description of the Win32 interface

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 Windows NT 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 Windows NT system communicates with the ICP2432 by means of the Simpact-supplied device driver.

The ICPTool program, supplied with the product, downloads the ICP-resident software to the ICP2432. Simpact's ICPTool for Windows NT (described in Chapter 2 and Appendix A) supports the software installation process and provides a graphical user interface to download protocols and run diagnostic test programs.

The ICP controls the communications links for the user applications. The user application programs can use Simpact's data link interface (DLI) to read and write data to the ICP2432 for transmission to or receipt from the communications links, and can change the link configuration parameters. See Chapter 3.

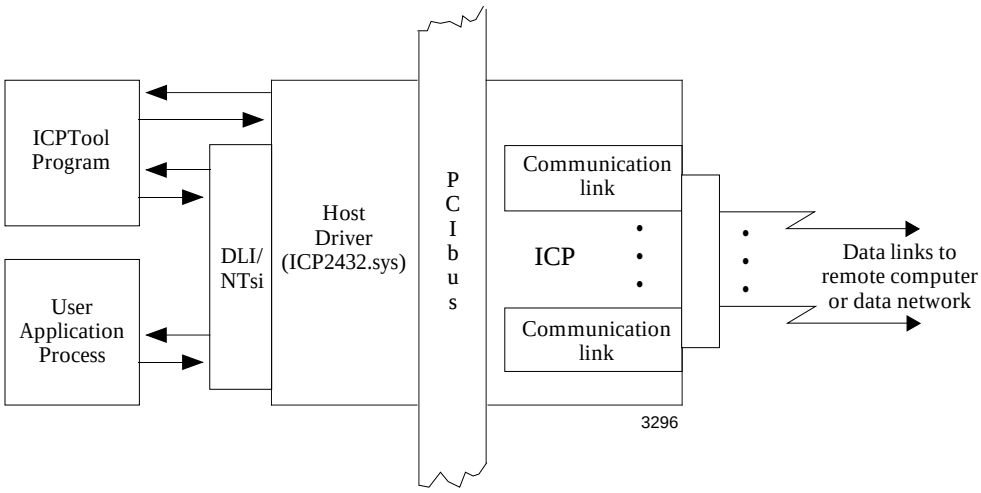

Figure 1–1: Typical Data Communications System Configuration

Chapter 2 Software Installation

This chapter describes Simpact's ICP2432 software installation procedure for Windows NT 4.0.

2.1 Memory Requirements

Simpact recommends that you have at least 32 megabytes of system memory for the ICP2432 product.

2.2 ICP2432 Software 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 first ICP2432 installation diskette or the CD-ROM into your Windows NT computer.

Step 3:
Start the installation by running the `setup.exe` program on the installation diskette or CD-ROM. Click `Next` when the startup information, shown in Figure 2–1, is displayed.

Note

If you install another ICP2432 board later, you do not have to run the `setup.exe` program again.

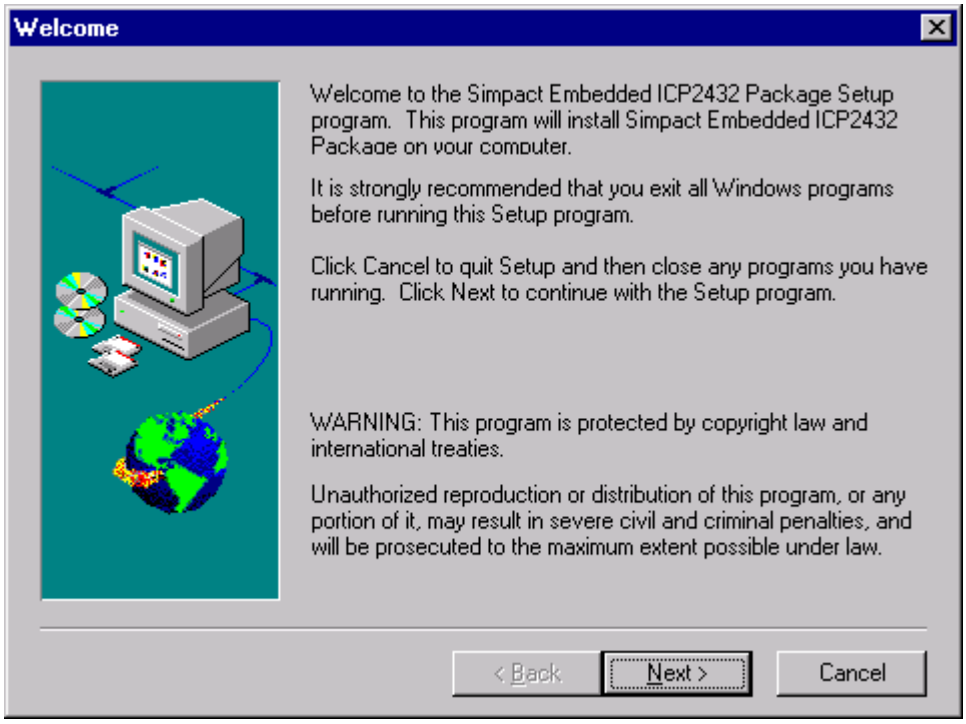

Figure 2–1: Startup Information for Embedded ICP2432

Step 4:

The installation script prompts for an installation directory in which to install the distribution software (Figure 2–2). The default directory is `C:\`. The software directory installed under `C:\` is `freeway`. All system files are installed in the Windows NT system home directory (for example, `C:\WinNT\system32`). If the default directory is acceptable, click `Next`.

To install the software in a different directory, click `Browse`. If you install the software in a directory other than `C:\`, the `<installed directory>\freeway\boot\icptoolload` file must be modified to point to the correct directory.

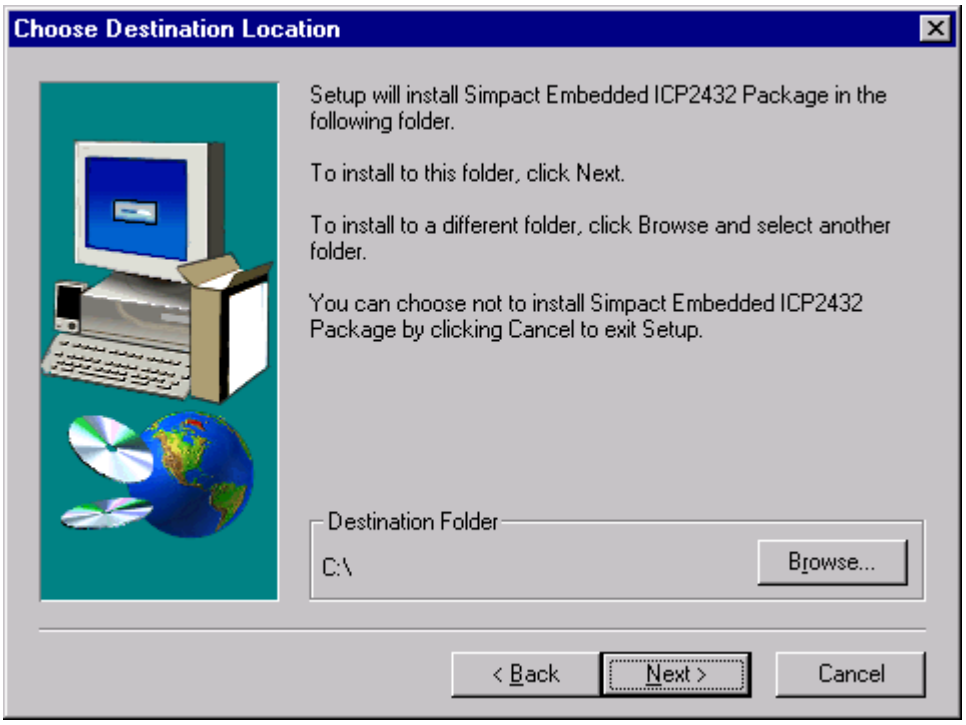

Figure 2–2: Installation Directory for Embedded ICP2432

Step 5:

Setup will then add program icons to the program folder (Figure 2–3). If the default folder name is acceptable, click `Next`. To choose a new folder name, click `Browse`.

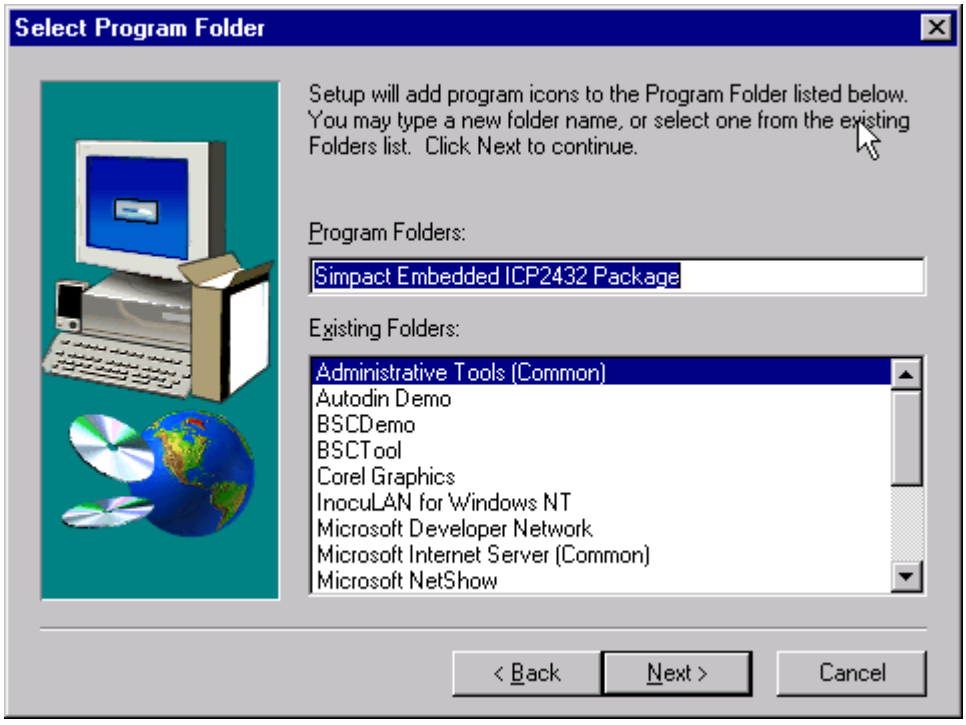

Figure 2–3: Program Folder

Step 6:

After completion of *Step 5*, the installation script updates and inserts keys into the system registry.

Step 7:

When the prompt asking "Do you want the ICP driver to be started automatically upon reboot?" appears, click Yes.

Step 8:

The Restart Windows menu (Figure 2–4) provides two options, to restart your computer now or later. Simpact recommends that you restart your computer now.

Note

Remove the installation diskette before restarting your computer.

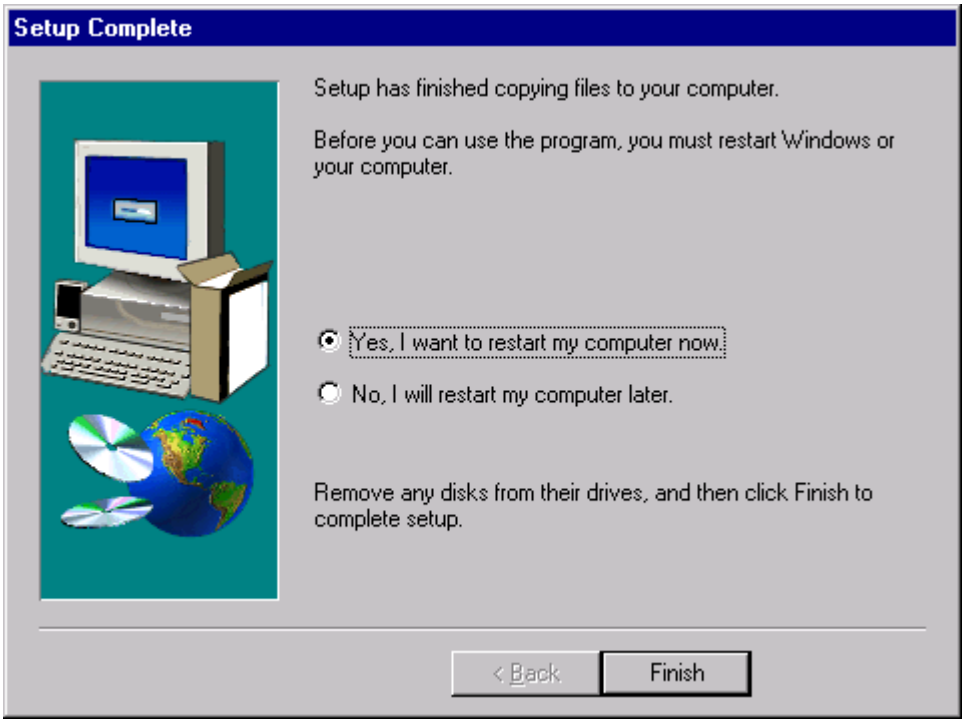

Figure 2–4: Restart Windows

2.3 Protocol or Toolkit Software Installation Procedure

The *ppp* variables mentioned throughout this section specify the particular protocol you are using. Refer to Table 2–1.

Table 2–1: Protocol Identifiers

Protocol or Toolkit	Protocol Identifier (*ppp*)
ADCCP NRM	`nrm`
AWS	`aws`
BSC3270	`bsc3270`[a]
BSC2780/3780	`bsc3780`[a]
FMP	`fmp`
Protocol Toolkit	`sps`
STD1200A	`s12`
X.25/HDLC	`x25`[b]

[a] Except for the load configuration file where *ppp* is `bsc`. For example, `bscload` is used for BSC3270 and BSC2780/3780.

[b] Except for 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

The load file, *ppp*`load`, is in the `freeway\boot` directory.

The executable object for protocol software other than protocol toolkit (*ppp*_fw_2432.mem) is in the freeway\icpcode\icp2432\protocols directory. The executable object for the protocol toolkit software (sps_fw_2432.mem) is in the freeway\icpcode\proto_kit\icp2432 directory.

The executable object for the system-services module for protocol software other than protocol toolkit (xio_2432.mem) is in the freeway\icpcode\icp2432\osimpact directory. The executable object for the system-services module for the protocol toolkit (xio_2432.mem) is in the freeway\icpcode\os_sds\icp2432 directory.

Source code for the loopback tests is in the freeway\client\test*ppp* directory.

Step 1:
Insert the protocol installation diskette or CD-ROM into your Windows NT computer.

Step 2:
Start the installation by running the setup.exe program on the installation diskette or CD-ROM. Click Next when the startup information, as shown in the FMP example in Figure 2–5, is displayed.

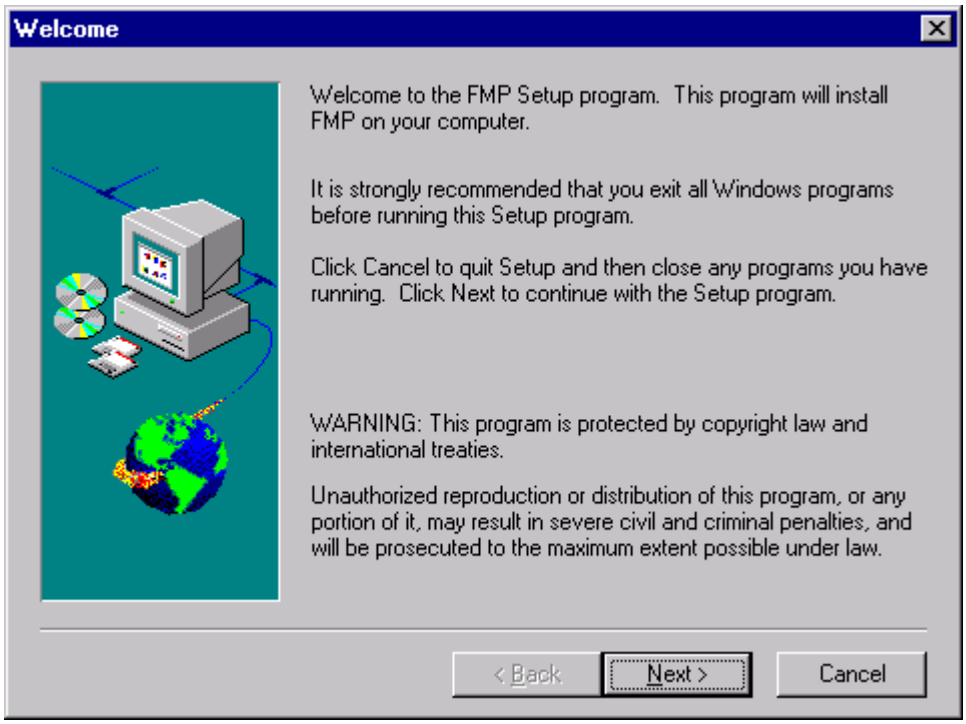

Figure 2–5: Startup Information for FMP

Step 3:

The installation script prompts for an installation directory in which to install the distribution software (Figure 2–6). The default directory is `C:\`. All system files are installed in the Windows NT system home directory (for example, `C:\WinNT\system32`). If the default directory is acceptable, click `Next`. To install the software in a different directory, click `Browse`.

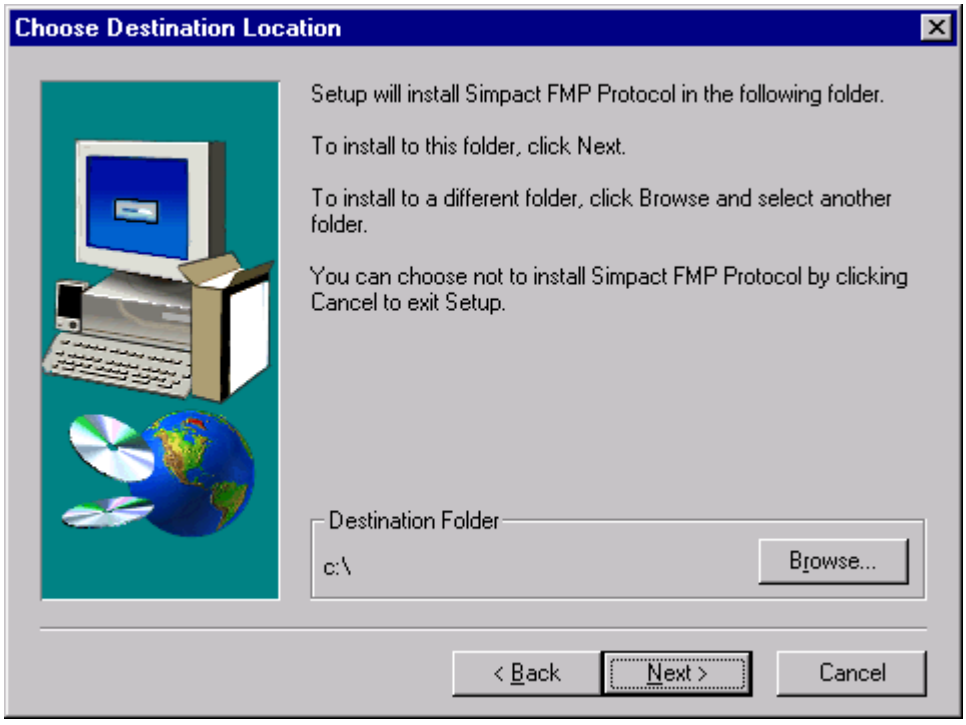

Figure 2–6: Installation Directory for FMP

Step 4:

Using any text editor, edit the load file (`freeway\boot\pppload`) for your protocol. Uncomment the lines associated with ICP2432. Modify the path names as needed. Do not change the memory locations (such as `40001200`) for the `LOAD` commands.

Note

If you are installing the X.25 protocol, you must build the CS API files. A make file is included that performs this operation.

From the `freeway\lib\cs_api` directory, enter the following command. The newly created file will be placed in the `freeway\client\int_nt_emb\bin` directory.

```
cd C:\freeway\lib\cs_api
nmake -f makefile.nta (for an Alpha NT system)
or
nmake -f makefile.nti (for an Intel NT system)
```

Dynamic link libraries must reside in the current working directory or in a directory specified in your "PATH" environment variable. Do one of the following:

Add `\freeway\client\int_nt_emb\lib` to your path.

or

Copy the `.dll` files from `\freeway\client\int_nt_emb\lib` to your `bin` directory or to another directory in your path.

Continue the installation at Step 5 below.

Step 5:

From the `freeway\client\test\`*ppp* directory, enter one of the following commands:

`nmake -f makefile.nta` (for an Alpha NT system)
`nmake -f makefile.nti` (for an Intel NT system)

The newly created files are placed in the `freeway\client\int_nt_emb\bin` directory.

Step 6:

Select "Start → Programs → Simpact ICP2432 → Simpact ICPTool" (or just double click on the Simpact ICPTool icon shown in Figure 2–7), then select `Download Protocol` from the *ICPTool Main Menu* (Figure 2–8) to display the *Protocol Download Menu* (Figure 2–9).

Figure 2–7: Simpact ICPTool Icon

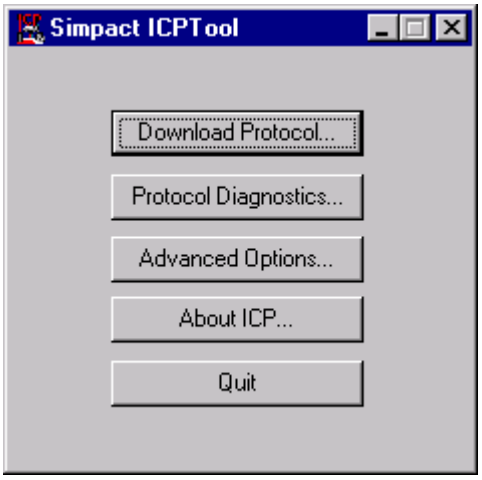

Figure 2–8: ICPTool Main Menu

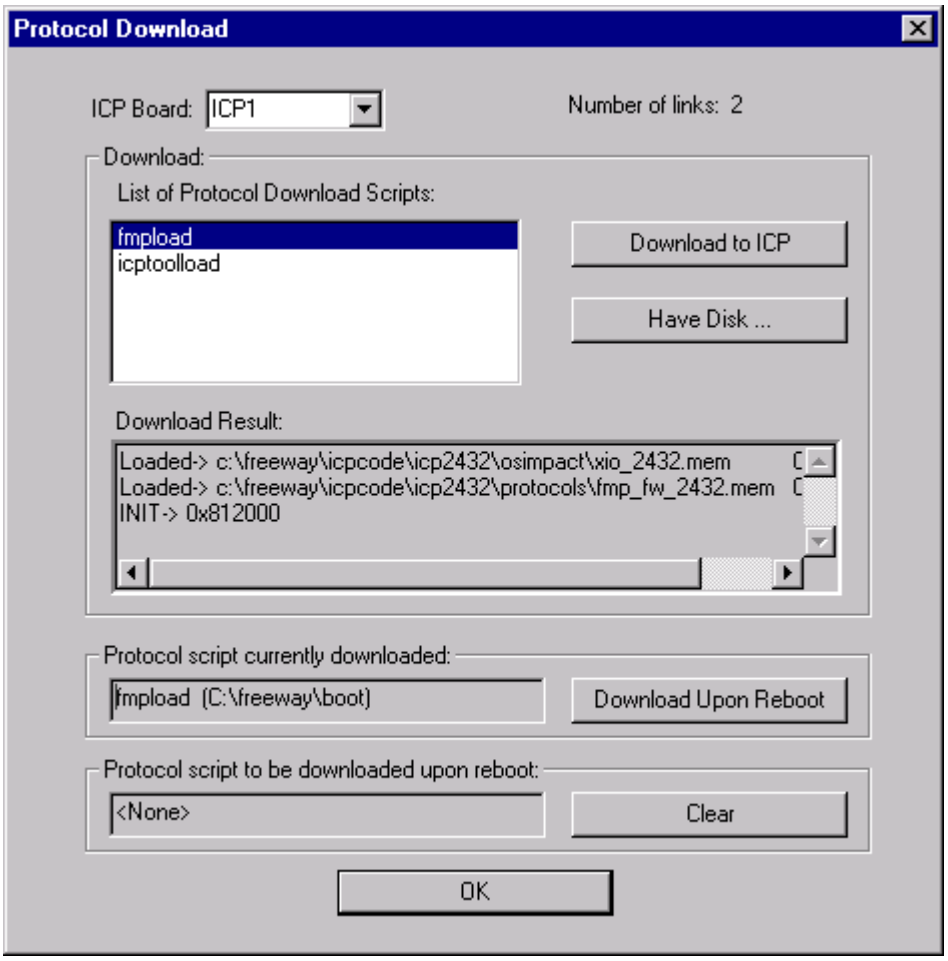

Figure 2–9: Protocol Download Menu

Step 7:

Select the protocol you wish to download in the `List of Protocol Download Scripts`, then select `Download to ICP`.

Step 8:

When the protocol is downloaded successfully, click `OK`, then `OK` again to exit Protocol Download, and `Quit` in the *ICPTool Main Menu*.

Step 9:

Go to the `freeway\client\int_nt_emb\bin` directory. Run the loopback test as described in the appropriate appendix for your protocol.

Chapter

3 Programming Using the Data Link Interface

This chapter describes the application program interface to Simpact's data link interface (DLI) module residing in an "embedded PCI" environment, as opposed to a Freeway server environment. An example of the embedded PCI environment is the PCI ICP2432 in a Windows NT system.

This chapter supplements the DLI interface described in the *Freeway Data Link Interface Reference Guide*, which supports the Freeway environment and is the primary source of information for application programmers. This chapter focuses on the differences between the DLI interface to Freeway and the interface required for the embedded PCI in an NT system. The *Freeway Data Link Interface Reference Guide* also describes DLI functionality and specific interface requirements.

Some references are made to the transport subsystem interface (TSI). However, a new embedded TSI interface was developed for the NT system, called the "NTsi." The NTsi resides between the DLI and the NT system, and is transparent to the user programming to the DLI interface.

This chapter is of interest primarily to programmers who are either porting an existing application (currently operational in the Freeway environment) to the embedded environment (such as the embedded PCI ICP2432), or who are developing an initial application to the DLI in the embedded environment. For those developing initial applications, first read (and always have available) the *Freeway Data Link Interface Reference Guide*.

3.1 Embedded Interface Description

3.1.1 Comparison of Freeway Server and Embedded Interfaces

An embedded interface is one where the application does not access a network in communicating with the ICP. Traditionally, the DLI and TSI interfaces supported client applications communicating with the Freeway server on a local-area network (LAN). This type of interface is shown in Figure 3–1.

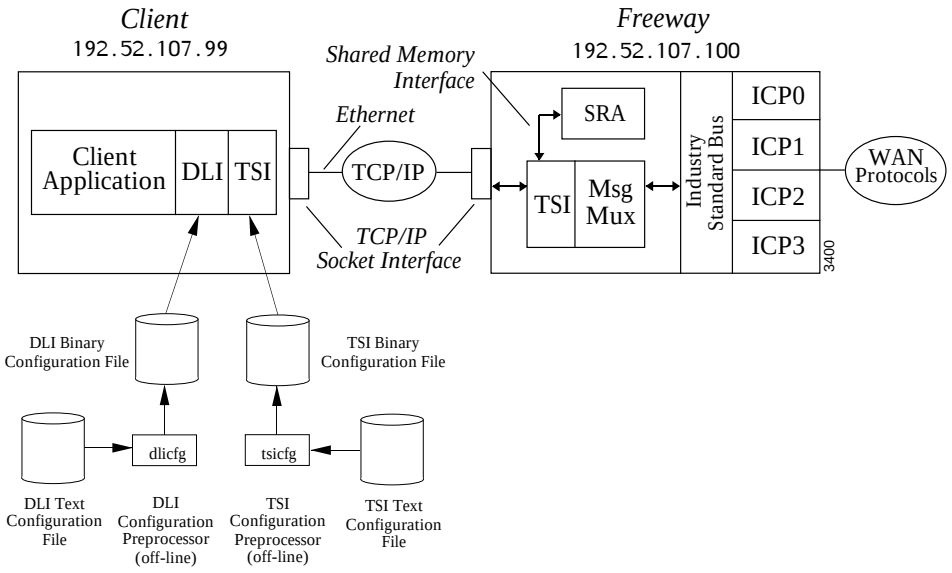

Figure 3–1: DLI/TSI Interface in the Freeway Server Environment

In an embedded interface, the DLI does not access the ICP over a network. Instead, the DLI references a modified TSI interface (NTsi) to access a locally attached ICP. This interface is shown in Figure 3–2. The NTsi interface supports the functionality that DLI expects by simulating a "Freeway-like" environment. In this environment no TSI messages are exchanged, and messages between the Freeway and MsgMux are simulated by NTsi.

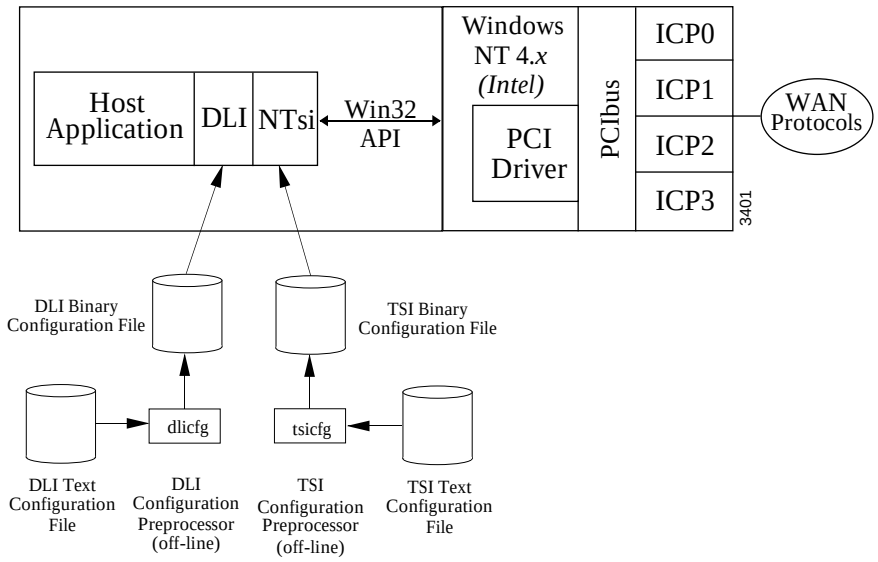

Figure 3–2: DLI/NTsi Interface in the Embedded ICP2432 Environment

3.1.2 Embedded Interface Objectives

The major design objective of the embedded transport subsystem interface (NTsi) was an implementation requiring no source code changes to the user's DLI interface when porting from a Freeway environment to an embedded PCI environment. The user program could be re-linked with the DLI/NTsi library and executed in the embedded environment. This objective has been met with the following exceptions:

- The `dlControl` function is not implemented
- The write expedite feature is not implemented

There are differences between these environments which the user must account for, as well as differences in system behavior; these are described in Section 3.2.

3.2 DLI Embedded Interface

The DLI embedded Interface is described in terms of the application's DLI/TSI configuration files (Section 3.2.1), and in the DLI itself (Section 3.2.2 on page 36). Within each context, required changes and any behavior differences are noted.

Caution

The DLI embedded interface is not thread-safe (which is also true with the Freeway interface). The multi-threaded user application must provide protection to the DLI interface.

3.2.1 Configuration Files

3.2.1.1 TSI Configuration File

A TSI configuration file that is operational in the Freeway environment can be used by the embedded PCI application without change. However, only a subset of those parameters required for Freeway operation are used in control of the embedded interface. Those parameters which are not used are simply ignored. Only the following parameters are used in the TSI configuration file:

TSI configuration file "Main" section:

- `MaxConns`
- `LogLev`
- `AsyncIO`
- `MaxBufSize`
- `MaxBuffers`
- `LogName` — Modify by adding an "`nt`" prefix and current process suffix
- `TraceName` — Modify by adding an "`nt`" prefix and current process suffix
- `TraceLev`

TSI configuration file connection-specific section:

- `MaxBufSize`
- `MaxInQ`
- `MaxOutQ`
- `MaxErrors`
- `LogLev`
- `TraceLev`
- `AsyncIO`
- `Timeout`
- `Transport` — Specify as either "`tcp-socket`" or "`shared-memory`". Even though not applicable in the embedded environment, this parameter must be specified to ensure compatibility with the Freeway environment.

Keep in mind the following points regarding the TSI configuration file:

1. Other than the `Transport` parameter, no parameters relating to TCP/IP or Shared Memory connections are used.

2. The use of tracing by the NTsi is strongly discouraged in an operational environment. Tracing is controlled by the `TraceLev` parameter in the TSI configuration file. NTsi tracing is performed "on-line" and adds significant overhead to the application. Its use should be reserved for testing the interface in the event of problems. When tracing is required, Simpact recommends that you first rely on the DLI tracing capabilities.

3. Simpact also recommends that you rely on the DLI logging capabilities. If tracing is specified in the TSI configuration file, a trace level of 3 or lower is strongly recommended. See Section 3.2.2.3 on page 39 and Section 3.2.2.4 on page 40 for more information on NTsi tracing and logging.

3.2.1.2 DLI Configuration File

The DLI configuration file is used in the same manner as in the Freeway environment. The relationship between the DLI and TSI configuration files is exactly as in the Freeway environment.

3.2.2 The Application Program's Interface to DLI

The embedded interface does not change the application's interface to the DLI. While this interface has remained intact, changes have been made in both the methods supporting the DLI and in the underlying functionality.

3.2.2.1 Embedded Interface — Changes in DLI/TSI Protocol

The lack of a network connection has eliminated the need for some of the current DLI and TSI functionality. While the changes described below are transparent to the user application program, they might be noted when examining NTsi trace files.

TSI commands — The TSI Bind, Unbind, Unbind Force, Ack, and Nak commands are not implemented. Essentially, the TSI transport layer has been replaced with a Win32 interface to the PCI driver, which does not require (or support) these commands.

DLI commands and responses — The Open Session command, Open Session response, Close Session command, and Close Session response are not required, but their transmission and reception are emulated. Even in *Raw* operation these commands are processed only by the DLI; their emulation should be transparent to the user application. While these commands are only emulated, they are recorded in the NTsi trace file (the command and response is treated exactly like a "real" transmission, except that they do not generate any I/O).

Transmission/Reception Buffer Format — The user data buffer is not affected by the embedded interface. As in the Freeway interface, the DLI builds the ICP and Protocol header using data from the DLI optional arguments (`OptArgs`), if they are supplied; otherwise it uses its same algorithm to generate these headers. The DLI continues to build a Freeway header and expects to receive this header in buffers from the ICP. NTsi strips the Freeway header from the transmission buffer before giving the buffer to the NT interface, and builds an appropriate Freeway header when receiving buffers, prior to giving the buffer to the DLI. The user sees the Freeway, ICP, and Protocol headers in the DLI trace file, but only sees the ICP and Protocol headers in the NTsi trace file.

3.2.2.2 Changes in the Application Program's Interface to DLI

No changes are required in the user application's interface to the DLI. Some DLI functions have changed in their implementation which might affect the user's expected behavior of the function. Changes in affected functions are described below.

dlBufAlloc

Implementation of buffer allocation has changed. Rather than allocating buffers from a pre-allocated buffer pool managed by TSI, buffer allocation requests are presented to NTsi which uses the NT system memory services to allocate buffers (using `malloc` calls). Do not assume any type of buffer initialization. Also, the size requested in the `dlBufAlloc` request is the size requested from the system. If the application requests one byte for the data buffer size, it should assume only one byte is returned. If buffers are not freed, rather than eventually receiving a `DLI_BUFA_ERR_NO_BUFS` error, indicating the TSI buffer pool has been exhausted, the user might notice a degradation of system performance (as would any application "misusing" system memory). A `DLI_BUFA_ERR_NO_BUFS` error should be interpreted as a failure in the underlying request for system memory.

Note

The user's buffer allocation only reflects the data size required by the application. That size is further modified to include "header" information required by the DLI. This header requirement is invisible to the user application.

dlBufFree

This function has also changed its implementation. In concert with the change in buffer allocation, a call to `dlBufFree` returns the requested buffer to the NT memory services (using `free`). Where previously the user could use the buffer pointer returned with the successful `dlBufFree` request (the buffer still existed in the TSI buffer pool), now that buffer is indeed freed. Any further reference to the buffer results in unpredictable results. Requests with a NULL buffer pointer continue to be returned with a `DLI_BUFF_ERR_INVALID_BUF` error message.

The user must supply the pointer received from `dlBufAlloc` when releasing this same buffer resource using `dlBufFree`.

dlClose

There is a minor change in the implementation which should be transparent to the user. The `dlClose` function continues to "unbind" the protocol and "detach" the ICP, but NTsi simulates a Freeway close action for the DLI (no Freeway to close). This action, rather than releasing a socket resource, results in closing the file handle to Simpact's ICP NT driver. In *Raw* operation, this Freeway close can always be assumed to have been completed successfully when returned from the call. The Freeway close and response are in the Ntsi trace file even though the buffers are never transmitted.

dlOpen

There is a minor change in the implementation which should be transparent to the user. NTsi simulates a Freeway open action for the DLI (no Freeway to open) by returning a Freeway Open response. When using *Raw* operation, the user can safely assume a successful open when an open request returns with `EWOULDBLOCK`. If successful (when using non-blocking I/O with callbacks enabled), callbacks occur as in the Freeway environment.

Callbacks

The DLI function now always receives callbacks in pairs; a connection callback followed by a main callback. Also, the DLI's callback function is never entered while "in a callback." However, the application's interface to DLI's callback mechanism remains as in the Freeway environment. The only difference the application might notice is more "main" callbacks in the open/close sequences when using *Normal* operation. However, in both cases, only one session callback is returned after either sequence completes. User callbacks are invoked using a callback thread unique to the user's process. This may require examining data shared between the user's callback processing and other user threads. Users can invoke any DLI service from their callbacks.

3.2.2.3 NTsi Tracing

NTsi tracing is similar to the TSI tracing supported in the Freeway environment; differences are described below. One important difference for the user is that printing of formatted trace information now occurs as the trace data is captured. Trace data is not saved to a trace buffer and printed at program completion. Rather, it is formatted and printed as soon as the buffer completes an I/O operation (whether successful or failed). For this reason, we strongly recommend that NTsi tracing be enabled only in diagnostic situations. The user might notice a less responsive interface with NTsi tracing enabled. However, because trace data is printed immediately, trace data is almost always available (even with abnormal program termination).

Trace file differences:

1. NTsi time stamps each buffer with millisecond granularity.
2. NTsi presents the number of bytes requested as well as the number transferred (input or output).
3. NTsi presents the NT system error code returned from the operation.
4. Freeway headers are not in NTsi buffers.
5. TSI headers are not in NTsi buffers.
6. NTsi decodes the ICP and Protocol headers.
7. The trace option (the `TraceLev` parameter in the TSI configuration file) of 16 (trace user's data) is not supported.
8. NTsi does not require running `tsidecode.exe` to decode the trace file. The trace file is written in a decoded format.

Figure 3–3 shows an example NTsi trace buffer.

3.2.2.4 NTsi Logging

Like tracing, NTsi logging is similar to the TSI logging supported in the Freeway environment; differences are described below. NTsi maps NT system errors into `tserrno`. Section 3.2.2.5 describes how specific NT errors are mapped to `tserrno`.

Log file differences:

1. NTsi presents the NT system error code (if no NT error occurred, 0 is displayed).
2. NTsi does not accompany the error code with a text description.

Figure 3–4 shows an example of an NTsi log buffer.

```
------------------------------------------------------------
Conn(0) :Sun Jun 08 13:11:02.213 1997
IOQ information: NumberReq(142), NumberXfered(98), ErrorCode(0)
======> (WRITE 98 bytes)
 000000: 00 00 00 00 00 52 08 05 40 00 cd cd 00 00 cd cd .....R..@.......
 000016: 01 10 00 00 00 00 00 00 01 00 cd cd cd cd cd cd ................
 000032: 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 defghijklmnopqrs
 000048: 74 75 76 77 78 79 7a 20 41 42 43 44 45 46 47 48 tuvwxyz ABCDEFGH
 000064: 49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 IJKLMNOPQRSTUVWX
 000080: 59 5a 20 30 31 32 33 34 35 36 37 38 39 20 61 62 YZ 0123456789 ab
 000096: 63 00  c.
@@@@@ Decoding begins
 ICP header info:
 OldClientID = 0 OldServerID = 0
 DataLength = 82 Cmd(2053) = DLI_ICP_CMD_WRITE
 Status(16384) = DLI_ICP_ERR_NO_ERR
 Parms: [0] = 52685 [1] = 0 [2] = 52685
 Protocol header info:
 Cmd(4097) = DLI_PROT_SEND_TRANS_DATA_EOM
 Modifier = 0 Link = 0
 Cir = 0 Sess = 1 Seq = 52685
 Parms: [0] = 52685 [1] = 52685
------------------------------------------------------------
```

Figure 3–3: NTsi Trace Buffer Example

```
Conn - 1:     tserrno = -712,    nterrno = 121,
Conn - 0:     tserrno = -716,    nterrno = 995,
Conn - 0:     tserrno = -716,    nterrno = 995,
Conn - 0:     tserrno = -716,    nterrno = 22,
Conn - 0:     tserrno = -716,    nterrno = 22,
```

Figure 3–4: NTsi Log Buffer Example

3.2.2.5 Error Codes

All NT system errors are mapped into existing TSI error codes (`tserrno`) so DLI can recognize the error condition and react accordingly. All NT errors are returned from calls to `GetLastError()` when an NT service fails. NT errors are mapped to `tserrno` definitions as shown in Table 3–1.

Table 3–1: NT Errors Mapped to tserrno Definitions

NT Error Code	tserrno	Value
ERROR_BAD_COMMAND	TSI_READ_ERR_SOCK_CLOSED	- 615
	TSI_WRIT_ERR_SOCK_CLOSED	-1714
	TSI_POLL_ERR_SOCK_CLOSED	- 716
ERROR_MORE_DATA	TSI_READ_ERR_OVERFLOW	- 612
	TSI_POLL_ERR_OVERFLOW	- 711
ERROR_ACCESS_DENIED	TSI_READ_ERR_INVALID_STATE	- 601
	TSI_WRIT_ERR_INVALID_STATE	-1701
ERROR_BUSY(170)		
ERROR_SEM_TIMEOUT	TSI_READ_ERR_READ_TIMEOUT	- 613
	TSI_WRIT_ERR_WRITE_TIMEOUT	-1712
	TSI_POLL_ERR_READ_TIMEOUT	- 712
	TSI_POLL_ERR_WRITE_TIMEOUT	- 713
ERROR_INVALID_FUNCTION		
ERROR_INVALID_PARAMETER		
ERROR_IO_DEVICE		
ERROR_NOACCESS		
ERROR_NOT_ENOUGH_MEMORY		
ERROR_OPERATION_ABORTED	TSI_READ_ERR_INTERNAL	- 614
	TSI_WRIT_ERR_INTERNAL	-1713
	TSI_POLL_ERR_SOCKET_CLOSED	- 716
ERROR_INVALID_USER_BUFFER	TSI_READ_ERR_INVALID_BUF	- 604
	TSI_WRIT_ERR_INVALID_BUF	-1707
	TSI_POLL_ERR_SOCKET_CLOSED	- 716

Chapter

4 Programming Using the Win32 Interface

Simpact's API layers are designed to free developers from the often-difficult programming details of an operating system and the interface details of the protocol software on the ICP. Simpact's API layers take care of tasks such as queuing I/O requests, buffer allocation (with properly aligned I/O buffers), building protocol headers, endian translation, session management, and others. Using the DLI interface described in Chapter 3 allows developers to concentrate more on their specific applications rather than the difficult communication and programming details associated with transferring data from one system to the next via a wide-area network. Simpact strongly encourages users to implement their applications using the DLI interface; however, users who wish to bypass Simpact's API layers and use the Win32 system services directly may do so, although many services provided by the DLI will need to be "reinvented" in the user application. This chapter provides the information necessary to build Win32 applications.

4.1 Function Mappings

This section describes how a user application interfaces with the ICP2432 device driver using Win32 system calls. It is not intended to be a Win32 tutorial; users who bypass Simpact's API layers are assumed to already know how to write Win32 applications, the purpose of the individual Win32 functions, and the programming issues that arise. This section merely lists the Win32 functions used to communicate with the ICP (via the device driver) and the actions performed.

4.1.1 Opening the ICP

Before a user application can perform any I/O transaction with the ICP, a handle to the ICP must be obtained. This is done by opening the ICP using the `CreateFile` Win32 system service.

One of the parameters to the `CreateFile` function is a device name having the form `\\.\IcpX`, where 'X' represents the device number (1, 2, ...).[1] `CreateFile` returns a handle to the ICP2432. After the handle is obtained, it is used in other Win32 system service calls, such as `ReadFile` or `WriteFile`.

Note that normal Windows NT file access control is in effect when the device is opened. For example, if an application sets the `dwDesiredAccess` parameter for `CreateFile` to `GENERIC_READ` and then later attempts to perform a write request to the ICP, the write request will fail. Access control is especially important when considering the value to use for the `dwSharedMode` parameter, since users will most likely wish to have multiple sessions to the ICP open simultaneously.

A typical call to `CreateFile` would look like this:

```
HANDLE hFile;
...
hFile = CreateFile( "\\\\.\\Icp1",
                    GENERIC_READ    | GENERIC_WRITE,
                    FILE_SHARE_READ | FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED,
                    NULL );
```

When `CreateFile` returns, `hFile` contains the handle to the ICP. Note also that overlapped I/O is being requested in the above example.[2] For non-overlapped I/O, remove the `FILE_FLAG_OVERLAPPED` flag from the call.

1. Keep in mind that unlike other Simpact products, device numbers begin at one instead of zero under Windows NT.

4.1.2 Reading Data

The ReadFile Win32 function is called by a user application to receive data from the ICP2432. One of the parameters to this function is the file handle that was returned from CreateFile. The handle must have been opened with GENERIC_READ access. The user buffer address and buffer size are also passed to ReadFile.

A typical call to ReadFile would look like this:

```
char    Buffer[ 1024 ];
DWORD   BytesReceived;
HANDLE  hFile;
BOOLEAN Status;
...
Status = ReadFile( hFile,
                   Buffer,
                   1024,
                   &BytesReceived,
                   NULL );          // Assume non-overlapped operation.
```

The final parameter must point to an OVERLAPPED structure if the handle was originally opened using the FILE_FLAG_OVERLAPPED flag in CreateFile.

It should be noted that direct I/O (as opposed to buffered I/O) is used to exchange data with the ICP. This means that when an I/O request is made, the physical page frames for the user buffer are locked in memory and become temporarily non-pageable until the ICP satisfies the request (which could be at a much later time). Hence, if a user application uses large I/O buffers and/or has a high number of outstanding read requests, memory resources are being used up and some system degradation might occur due to an increased number of page faults. When the I/O request is satisfied, the pages become unlocked and can be paged by Windows NT in the normal manner.

---

2. *Overlapped I/O* is the Win32 term used to describe non-blocking I/O (also called asynchronous I/O). When an overlapped I/O request is issued, the executing thread does not block, but continues executing concurrently with the I/O. When overlapped I/O is used, it is up to the user application to synchronize with I/O completion before processing the data. This is usually done by associating an event object with the I/O request and using the Win32 function WaitForSingleObject or WaitForMultipleObjects to wait for the event(s) to enter the *signalled* state.

4.1.3 Writing Data

The `WriteFile` Win32 function is called by a user application to send data to the ICP. One of the parameters to this function is the file handle that was returned from `CreateFile`. The handle must have been opened with `GENERIC_WRITE` access. The user buffer address and requested transfer size are also passed to `WriteFile`.

A typical call to `WriteFile` would look like this:

```
char    Buffer[ 1024 ];
DWORD   BytesWritten;
HANDLE  hFile;
BOOLEAN Status;
...
Status = WriteFile( hFile,
                    Buffer,
                    1024,
                    &BytesWritten,
                    NULL );         // Assume non-overlapped operation.
```

The final parameter must point to an `OVERLAPPED` structure if the handle was originally opened using the `FILE_FLAG_OVERLAPPED` flag in `CreateFile`.

Caution

For proper communication with the ICP, as well as efficient data transfer over the 32-bit data path of the PCIbus, the ICP requires user I/O buffers to be aligned on a longword boundary. In addition, the Windows NT operating system itself may impose additional alignment requirements. User applications are responsible for meeting all alignment requirements; the Windows NT I/O Manager does not correct alignment discrepancies during a DMA transfer. The alignment requirement for a particular ICP may be determined by using the `IOCTL_ICP_GET_DRIVER_INFO` device control request (Section 4.1.5).

4.1.4 Cancelling I/O

I/O requests may be cancelled using the Win32 `CancelIo` function. [3] This function takes one parameter; a file handle obtained from `CreateFile`. Using `CancelIo` automatically implies the use of overlapped I/O. That is, a thread that issues a non-overlapped I/O request blocks on the `ReadFile` or `WriteFile` call until the I/O completes; and if the thread is blocked, it cannot call `CancelIo`. A typical call to `CancelIo` looks like this:

```
HANDLE  hFile;
BOOLEAN Status;
...
Status = CancelIo( hFile );
```

The `CancelIo` function cancels all I/O requests – both reads and writes – that were issued by the calling thread for the handle specified. If two or more threads have duplicate handles (for example, when one thread creates a second thread, and the second thread inherits the first thread's handles), only the I/O requests issued by the calling thread are cancelled for the given handle; any other I/O requests for the handle are still active. One implication of this is that a thread cannot use `CancelIo` to unblock a second thread that is waiting for a non-overlapped I/O request to complete.

4.1.5 Device Control

User applications might sometimes need to communicate directly to the device driver (rather than the ICP) to obtain information or perform other control functions. The `DeviceIoControl` Win32 function makes special requests directly to the driver. Again, the handle returned by `CreateFile` is necessary as a parameter to this function. In addition, a control code is passed in the `dwIoControlCode` parameter. This control code tells the driver which special function to perform. The control codes recognized by the ICP2432 driver are given in Table 4–1, and defined in the `Icp2432Nt.h` header file that is included on the product installation media.

3. `CancelIo` is a new Win32 function as of Windows NT release 4.0.

Table 4–1: ICP2432 Driver Control Codes

IOCTL Code	Description
IOCTL_ICP_CANCEL_READS	Cancel all pending read requests for a given file handle
IOCTL_ICP_CANCEL_WRITES	Cancel all pending write requests for a given file handle
IOCTL_ICP_GET_DRIVER_INFO	Get internal information from the driver
IOCTL_ICP_INIT_ICP	Reset the ICP
IOCTL_ICP_INIT_PROC	Inform the ICP to execute its INIT routine
IOCTL_ICP_SET_DNL_TARGET_ADDR	Set ICP target address of next download block
IOCTL_ICP_WRITE_EXPEDITE	Send a high-priority request to the ICP

4.1.5.1 Cancelling I/O Requests

The IOCTL_ICP_CANCEL_xxx (where xxx is either READS or WRITES) control codes are used to cancel I/O requests that were issued by the file handle indicated in the DeviceIoControl call. No input or output buffers need to be specified in the function call when one of these control codes is used. The following example shows how to cancel all read requests issued for a handle:

```
DWORD   Dummy;
HANDLE  hFile;
OVERLAPPED Overlap;
BOOLEAN Status;
...
Status = DeviceIoControl( hFile,
                          IOCTL_ICP_CANCEL_READS,
                          NULL,
                          0,
                          NULL,
                          0,
                          &Dummy,   // Not used, but required.
                          &Overlap );
```

The final parameter must point to a valid OVERLAPPED structure. Threads using non-overlapped I/O block until a request completes, and therefore cannot cancel I/O requests.

Note that the IOCTL_ICP_CANCEL_xxx functions have different semantics than the CancelIo Win32 function. The CancelIo function cancels I/O requests based on a particular thread/handle combination; the IOCTL functions supplied by Simpact cancel all I/O requests of a particular type (reads or writes) for a particular handle, regardless of who issued the requests.

The IOCTL_ICP_CANCEL_WRITES function cancels all pending write requests for a given file handle, including any expedited writes (see Section 4.1.5.3).

Caution

The IOCTL_ICP_CANCEL_xxx functions are supplied by Simpact for backward compatibility with device drivers prior to version 1.1-0. Simpact does not guarantee that these functions will be supported in future releases, and recommends that the CancelIo function be used to cancel I/O requests.

4.1.5.2 Obtaining Internal Driver Information

The IOCTL_ICP_GET_DRIVER_INFO control code is used to retrieve information from the driver. The application supplies an output buffer large enough to hold an ICP_Driver_Info structure, which is defined in the Icp2432Nt.h header file and has the format shown in Figure 4–1. Table 4–2 describes the ICP_Driver_Info structure fields. The possible ICP states are given in Figure 4–2 and also defined in the Icp2432Nt.h header file.

```
typedef struct _ICP_Driver_Info
  {
     /* Handle-specific items. */
     ULONG     Node;
     BOOLEAN   IcpWasReset;

    /* Items about the ICP to which the handle is opened. */
     ULONG     DeviceNumber;
     ULONG     NumberOfPorts;
     ICP_State IcpState;
     ULONG     BufferAlignment;
     ULONG     NumberOfOpenHandles;

     /* Driver-wide global information. */
     ULONG     NumberOfIcps;

     /* Driver-specific items. */
     UCHAR     Version[ MAX_VERSION_LENGTH ];
  } ICP_Driver_Info, *PICP_Driver_Info;
```

Figure 4–1: `ICP_Driver_Info` Structure

Table 4–2: `ICP_Driver_Info` Structure Fields

Field	Description
`Node`	Driver's internal node number corresponding to the file handle used in the `DeviceIoControl` request (Section 4.2.3 describes node numbers)
`IcpWasReset`	TRUE if the ICP has been reset since the handle was open
`DeviceNumber`	Device number of the ICP to which the handle is opened
`NumberOfPorts`	Number of ports (links) on the ICP (2, 4, or 8)
`IcpState`	Current state of the ICP (see Figure 4–2)
`BufferAlignment`	The device's alignment requirement for user I/O buffers. For example, a value of four is returned if buffers must be aligned on a longword boundary, eight is returned for quadword alignment, and so on
`NumberOfOpenHandles`	Number of distinct handles open to this particular ICP
`NumberOfIcps`	Total number of ICP2432s in the system recognized by the driver
`Version`	A NULL-terminated string specifying the driver version number

```
typedef enum
{
   ICP_State_Unknown,    // Unknown state.  ICP is unusable.
   ICP_State_POST,       // RESET# asserted.  POSTs active.
   ICP_State_Reset,      // POSTs complete.  ICP is reset.
   ICP_State_Download,   // ICP is in download mode.
   ICP_State_Init,       // ICP is executing INIT procedure.
   ICP_State_Ready       // Normal operation.
} ICP_State, *PICP_State;
```

Figure 4–2: IcpState Field Definitions

The following excerpt shows how to obtain the driver information:

```
DWORD           BytesReturned;
ICP_Driver_Info DriverInfo;
HANDLE          hFile;
BOOLEAN         Status;
...
Status = DeviceIoControl( hFile,
                          IOCTL_ICP_GET_DRIVER_INFO,
                          NULL,
                          0,
                          &DriverInfo,
                          sizeof( DriverInfo ),
                          &BytesReturned,
                          NULL );   // Assume non-overlapped operation.
```

When the function completes, DriverInfo contains the driver information.

4.1.5.3 Expedited Write Requests

The IOCTL_ICP_WRITE_EXPEDITE control code is used to send an "expedited" message to the ICP. The device driver sends expedited write requests to the ICP before any normal write requests (that is, requests that were posted with WriteFile). Multiple expedited write requests are sent to the ICP in the order in which they are received by the driver, but always before any normal writes that may be queued. The following segment shows how to make an expedited write request:

```
char       Bfr[ 1024 ];
DWORD      BytesWritten;
HANDLE     hFile;
OVERLAPPED Overlap;
BOOLEAN    Status;
...
Status = DeviceIoControl( hFile,
                          IOCTL_ICP_WRITE_EXPEDITE,
                          Bfr,
                          1024,
                          NULL,
                          0,
                          &BytesWritten,
                          &Overlap );
```

The above example uses overlapped I/O. An application using non-overlapped I/O probably has no need to make expedited write requests because only one write request will be active at any given time (that is, the thread blocks during the write). However, if multiple threads share a single file handle, there is nothing to stop one of the threads from making expedited write requests using non-overlapped I/O (for example, one of the threads might be a "control" thread whose messages have precedence over those of the other threads).

Care must be taken when using expedited writes because an expedited write is a global entity to the driver. That is, an expedited write is sent before all normal write requests that the driver has queued, not just before normal writes for the specified handle. For example, if five processes, each with a unique handle open to the ICP, simultaneously issue write requests to an ICP, and one of those requests is an expedited write, the expedited write preempts the requests of the other processes and is sent to the ICP first. [4] Additionally, there is a greater amount of overhead associated with expedited writes than with normal writes, and expedited writes are less efficient and require more system resources. Developers should use the expedited write capability judiciously.

4. Requests cannot be queued "exactly" at the same time, of course, so it is possible that the driver may have started sending a normal write request to the ICP before receiving the expedited write request from the user application. Once in progress, however, a normal write request cannot be preempted. The expedited write will be the next request sent to the ICP.

Not all Simpact protocols recognize expedited write requests and treat them the same as normal write requests. Some protocols that do recognize expedited writes also associate special characteristics with them in addition to the high-priority nature (for example, expedited writes may not be subject to flow control). Consult the programmer's guide for your particular protocol to determine whether expedited writes are supported and what attributes are given to them by the protocol software. Regardless of how the protocol software treats expedited writes, the ICP2432 device driver does not assign any special characteristics to them other than to send them to the ICP before any normal writes that are queued.

4.1.5.4 Support for ICP Initialization

The remaining control codes – `IOCTL_ICP_INIT_ICP`, `IOCTL_ICP_INIT_PROC`, and `IOCTL_ICP_SET_DNL_TARGET_ADDR` – are used to initialize the ICP and are beyond the scope of this document. The `IcpTool` utility provided by Simpact on the distribution media should be used to initialize an ICP. Customers who have a genuine need to dynamically reinitialize an ICP from within their application should contact Simpact Customer Support as described on page 13 for information on using the `IcpDnld.dll` dynamic link library provided on the distribution media.

4.1.6 Closing A Handle

A user application terminates a session with the ICP by closing the associated file handle. The `CloseHandle` function is used to close a handle to the ICP.

A typical call to `CloseHandle` would look like:

```
HANDLE  hFile;
BOOLEAN Status;
...
Status = CloseHandle( hFile );
```

4.2 Driver Features and Capabilities

The ICP2432 device driver provides the following capabilities:

- Support for downloading an application system to the ICP
- Communication with ICP-resident tasks
- Multiplexed I/O (multiple active requests per device)
- Error logging

4.2.1 Download Support

Before applications can use the ICP, it must be *downloaded*; that is, the ICP-resident application system must be copied to the ICP's memory, then executed. This procedure must occur whenever the ICP is reset. The ICP2432 device driver provides the services necessary to reset and download the ICPs.

Note

User applications normally do not have to worry about downloading the ICP. The `ICPTool` program supplied with the ICP2432 takes care of downloading the ICP with the appropriate software.

4.2.2 Communication With ICP-Resident Tasks

A Windows NT application controls the ICP by communicating with the protocol software that is executing on the ICP. It accomplishes this by opening a "session" with the ICP. In normal ICP operation (that is, after the download sequence has completed), user applications communicate with the ICP software by making read and write requests. Creating a file handle opens a data path to the ICP and its software, and the first command sent by the application to the ICP software is usually an "attach" command, which opens a session to a particular link on the ICP. The commands and

responses recognized by the ICP software are described in the *Programmer's Guide* for the particular protocol executing on the ICP.

4.2.3 Multiplexed I/O

Whenever a file handle is created ***(not duplicated, but created)***, a new data path is made with the ICP. File handles can be thought of as being associated with a *logical* channel to the ICP (what is known as a *node* internally to the driver). All nodes share one physical interface to the ICP. At any given moment, there is *at most* one command being sent to the ICP (because there is only one physical channel), but there can be any number of pending I/O requests active. Requests are queued on their associated node until such time when the ICP completes the request. User applications using non-overlapped I/O, have at most one I/O request pending on a given node; whereas any number of reads or writes can be pending on a node when overlapped I/O is used.

I/O requests on a given node always complete sequentially.[5] However, I/O requests complete randomly on a global device-wide basis; that is, if Process A issues a read request and Process B then issues a read request, there is no guarantee that Process A's request will complete before Process B's request (assuming the two processes are using distinct file handles to the ICP).

4.2.4 Error Logging

When the ICP2432 driver detects an error, it creates an entry in the Windows NT system event log. The system event log can be viewed by opening the Event Viewer (Start–>Programs–>Administrative Tools (Common)–>Event Viewer) and selecting Log–>System from the menu bar. Figure 4–3 shows a sample event log displayed in the Event Viewer.

5. At least within the type of request. That is, all read requests on a node complete sequentially in the order in which they were issued, and all write requests on a node complete sequentially, but the combined set of reads and writes does not necessarily complete in the order issued.

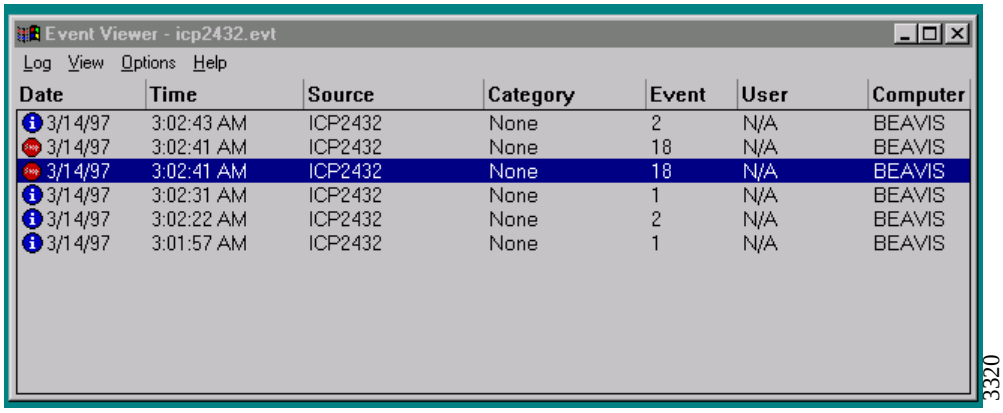

Figure 4–3: Sample Event Log Displayed in the Event Viewer

The "Source" column identifies the source of the log message. As shown in Figure 4–3, error messages from the ICP2432 driver are identified by the string "ICP2432." The icon at the beginning of each line indicates the severity of the event; an 'i' indicates an informational message, an exclamation point indicates a warning message, and a stop sign indicates an error message. Double-clicking on a line gives further details about the event, as shown in Figure 4–4.

The "Description" field in the Event Detail describes the event, and the severity is indicated in the "Type" field. Depending on the event, the ICP2432 driver might dump internal information along with the event notification. This information (which is for Simpact internal use only) is displayed in the "Data" field of the Event Detail (beginning at offset 0028).

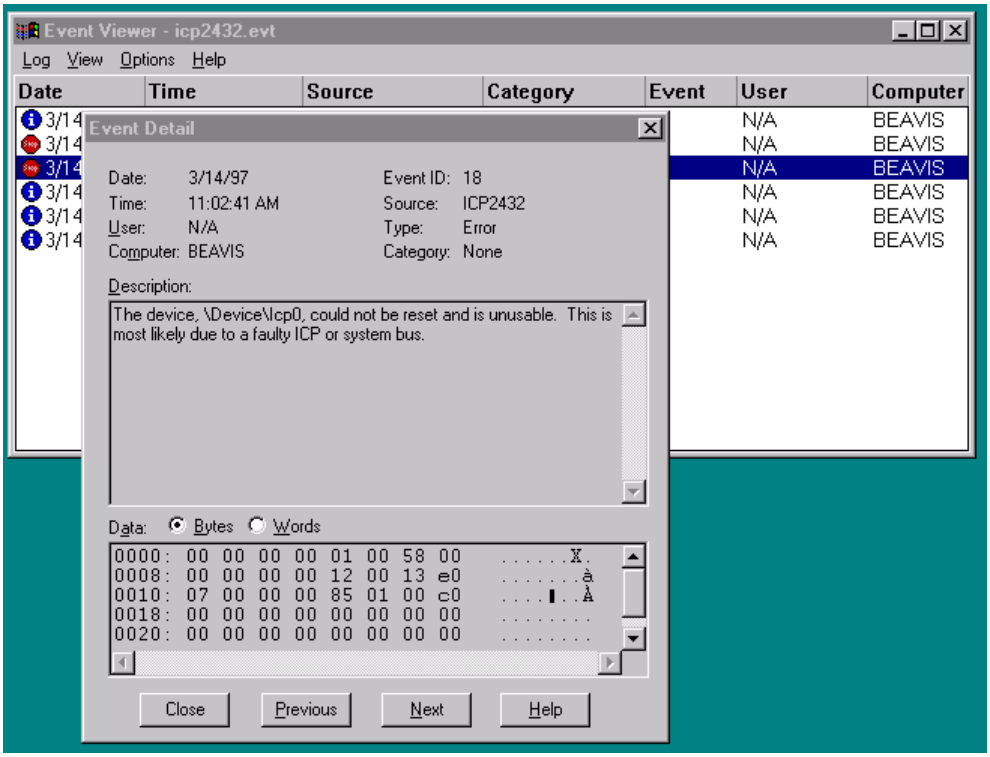

Figure 4–4: Log Message Event Detail

4.3 I/O Completion Status

The ICP2432 driver is responsible for setting the completion status of any I/O request that it processes.[6] If a Win32 I/O function returns an error, the `GetLastError` or `GetOverlappedResult` function can be used by the application to obtain the error code that indicates the reason for the failure. Because the meaning of a Win32 error code can sometimes be obscured when it is translated from the original status code returned to the I/O Manager by the driver, this section describes the error responses that user applications might encounter and their cause. Note that this is a subset of all possible error returns, because other Windows NT components can also fail an I/O request.

4.3.1 Successful Completion

The following success codes are returned by the driver.

`ERROR_IO_PENDING`

The request requires additional processing and is pending. Only applications using overlapped I/O see this completion code.

`NO_ERROR` or **`ERROR_SUCCESS`**

These are two names for the same completion code and indicate that a request completed successfully.

4.3.2 Error Completion

The following error codes are returned by the driver.

`ERROR_ACCESS_DENIED`

The requesting handle is stale (that is, the ICP has been reset since the handle was opened). The handle must be closed (with `CloseHandle`).

6. Not all I/O requests necessarily reach the ICP2432 driver; other Windows NT components such as the I/O Manager can fail an I/O request without passing it to the driver.

ERROR_BAD_COMMAND

A read request or an expedited write request was issued while the ICP was not in normal operating mode. Reads and expedited writes cannot be requested until the ICP has been initialized.

A write request was issued while the ICP was not in normal operating mode or download mode.

A cancel request was issued while the ICP was not in normal operating mode. Requests may not be cancelled until the ICP has been initialized.

An IOCTL_ICP_INIT_PROC request was issued while the ICP was not in download mode. User applications should never encounter this scenario because ICPs are initialized with Simpact-supplied utilities only.

An IOCTL_ICP_SET_DNL_TARGET_ADDR request was issued while the ICP was not in download mode. User applications should never encounter this scenario because ICPs are initialized with Simpact-supplied utilities only.

ERROR_BUSY

An attempt was made to open a handle to the ICP during board initialization while a handle was already open. The device driver forces exclusive access to the ICP during initialization to prevent collisions between two or more threads that might attempt to initialize the ICP concurrently.

A read request or an IOCTL_ICP_CANCEL_READS request was issued while a read cancel operation was in progress.

A write request, expedited write request, or IOCTL_ICP_CANCEL_WRITES request was issued while a write cancel operation was in progress.

An IOCTL_ICP_INIT_PROC request was issued while the ICP was writing a download block or there was already an initialization request in progress. User applica-

tions should never encounter these scenarios because ICPs are initialized with Simpact-supplied utilities only.

An `IOCTL_ICP_SET_DNL_TARGET_ADDR` request was issued while the target address was already set. User applications should never encounter this scenario because ICPs are initialized with Simpact-supplied utilities only.

`ERROR_FILE_NOT_FOUND`

The device driver did not find any ICP2432s in the system. User applications will never see this error because it can only occur when the driver is initially loaded into the system.

`ERROR_INVALID_FUNCTION`

An `DeviceIoControl` function call was made with an unrecognized control code.

A request to write a download block was issued before the target address was set or while a download write was already in progress. User applications should never encounter these scenarios because ICPs are initialized with Simpact-supplied utilities only.

`ERROR_INVALID_PARAMETER`

A filename was specified with the device name in `CreateFile` (for example, `\\.\Icp1\Filename`). ICPs are not storage devices, and therefore a filename cannot be specified when opening a handle to the device.

A NULL buffer pointer was used in an I/O request.

An `IOCTL_ICP_GET_DRIVER_INFO` request was made with a NULL output buffer pointer.

An `IOCTL_ICP_INIT_PROC` or `IOCTL_ICP_SET_DNL_TARGET_ADDR` request was made with a NULL input buffer pointer, or a value of zero was supplied. User applica-

tions should never encounter these scenarios because ICPs are initialized with Simpact-supplied utilities only.

ERROR_INVALID_USER_BUFFER

An invalid buffer size was used in an I/O request. Buffers must be at least large enough to contain the headers recognized by the protocol software. The one exception to this is the download block, which may be a minimum of one byte in length. The maximum buffer size allowed by the driver is 65K, which is the maximum amount of data that the ICP can transfer in a single DMA operation. The Windows NT kernel can also impose additional restrictions on the maximum buffer size. Kernel-imposed restrictions are defined by the maximum number of mapping registers that it allocates for a single DMA transaction. Because there is a one-to-one correspondence between mapping registers and virtual memory pages, the system's page size also influences the maximum buffer size allowed by the kernel.

An `IOCTL_ICP_GET_DRIVER_INFO` request was made with an output buffer that was too small to hold the information.

An `IOCTL_ICP_INIT_PROC` or `IOCTL_ICP_SET_DNL_TARGET_ADDR` request was made with an input buffer that was too small to hold the information required by the driver. User applications should never encounter these scenarios because ICPs are initialized with Simpact-supplied utilities only.

ERROR_IO_DEVICE

The file object pointer passed from the I/O Manager to the device driver does not correspond to any active node. This is an internal driver error.

No work queue entry was found for an I/O Request Packet (IRP) that the I/O Manager was attempting to cancel. This is an internal driver error.

The ICP negatively acknowledged a driver command. This is an internal driver error, or possibly an indication of a hardware error in the system.

The ICP did not finish its power-on tests within the allotted time from reset, or a failure was detected during the tests. Both of these are indications of a bad ICP. User applications should never encounter these scenarios because ICPs are initialized with Simpact-supplied utilities only.

The ICP sent an unrecognized command after the protocol software was initialized. This indicates a bad ICP or possible system hardware problems. User applications should never encounter this scenario because ICPs are initialized with Simpact-supplied utilities only.

ERROR_MORE_DATA

A user buffer for a read request was too small to hold the amount of data that the ICP wanted to supply. The user buffer contains partial data (filled to capacity), but the remaining data is lost.

ERROR_NOACCESS

An I/O buffer was misaligned.

ERROR_NOT_ENOUGH_MEMORY

The driver could not allocate non-pageable system memory.

An attempt was made to open a handle to the ICP, but all nodes in the driver were already allocated.

An adapter object could not be allocated for a device. User applications will never see this error because it can only occur when the driver is initially loaded.

ERROR_OPERATION_ABORTED

The I/O request was cancelled. A request can be cancelled for various reasons. For example, an application may have explicitly issued a cancel request via the `CancelIo` function or the `DeviceIoControl` function (with a control code of `IOCTL_ICP_CANCEL_xxx`). Another reason could be that the board was reset, either explicitly when the user reinitialized the ICP or implicitly when the driver

detected an unrecoverable error (such as the board not responding to a command). Additionally, the I/O Manager may attempt to cancel I/O requests in response to a thread being terminated abnormally. However, this last scenario can only occur in applications that share file handles (and I/O requests) among multiple threads.

ERROR_RESOURCE_DATA_NOT_FOUND

The driver could not find resource information (such as the interrupt vector, base address, and so on) for an ICP2432. User applications will never see this error because it can only occur when the driver is initially loaded.

ERROR_SEM_TIMEOUT

The ICP did not respond to a driver command within the allotted time.

Appendix

A ICPTool for Windows NT

This appendix describes the features of the Simpact ICPTool program for Windows NT. ICPTool provides a graphical user interface to download protocol software to the ICP2432 and run the diagnostic test programs. ICPTool is installed with the ICP2432 product software.

A.1 ICPTool Main Menu

To start the ICPTool program, select "Start → Programs → Simpact ICP2432 → Simpact ICPTool" (or just double click on the Simpact ICPTool icon shown in Figure A–1).

Figure A–1: Simpact ICPTool Icon

The *ICPTool Main Menu* (Figure A–2) allows you to:

- download a protocol onto the ICP (Section A.1.1)
- run any protocol diagnostic test (Section A.1.2)
- do advanced configuration (Section A.1.3).

Select `About ICP` to display ICP information similar to Figure A–3.

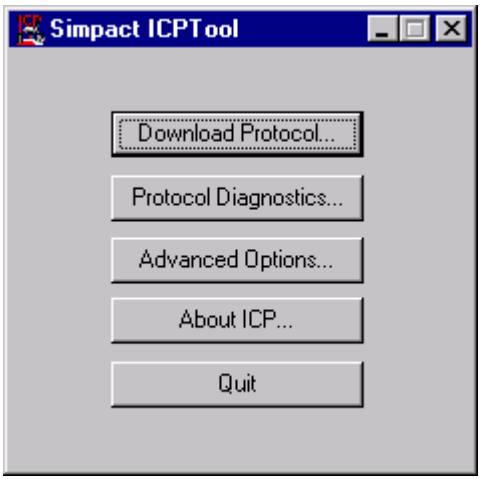

Figure A–2: ICPTool Main Menu

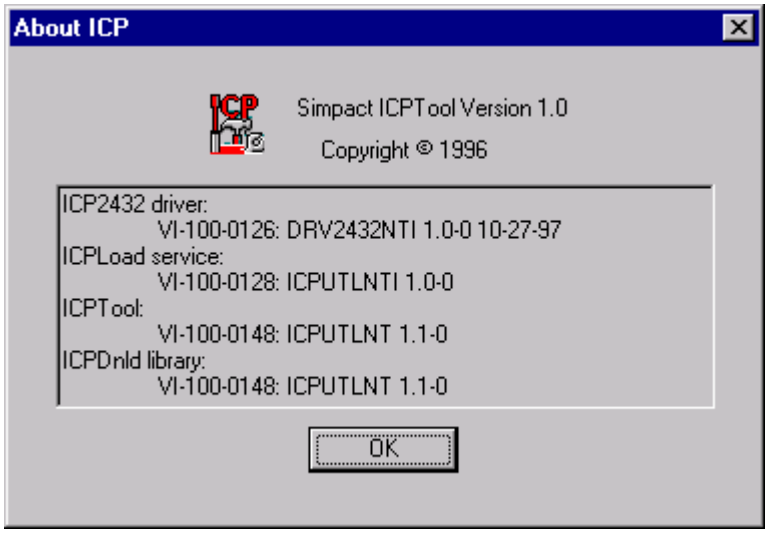

Figure A–3: ICP Information

A.1.1 Download Protocol

Select Download Protocol from the *ICPTool Main Menu* to display the *Protocol Download Menu* (Figure A–4). If your system contains more than one ICP2432 board, select the ICP to be downloaded. Select a download script from the List of Protocol Download Scripts (which are stored in <installation directory>\freeway\boot). Table A–1 summarizes the download selections.

Within the Protocol script currently downloaded box, if no protocol is currently loaded on the ICP, the message <None> is displayed. If there is no information from the driver, the message Not available is displayed for the Number of links.

Table A–1: Download a Protocol to the ICP

Button Selected	Action
Download to ICP	After you make a selection from the List of Protocol Download Scripts, the protocol software is downloaded to the ICP
Have Disk	Allows you to specify the location of a user-defined protocol download script to be loaded. A browser window appears to locate the download script file.
Download upon reboot	If you want the protocol to be automatically downloaded to the ICP upon future reboot, select this button. The script specified in "Protocol scripts currently downloaded" will be set in "Protocol script to be loaded upon Reboot."
Clear	If you do not want to load the download script specified in "Protocol script to be loaded upon Reboot" upon reboot, select this button.

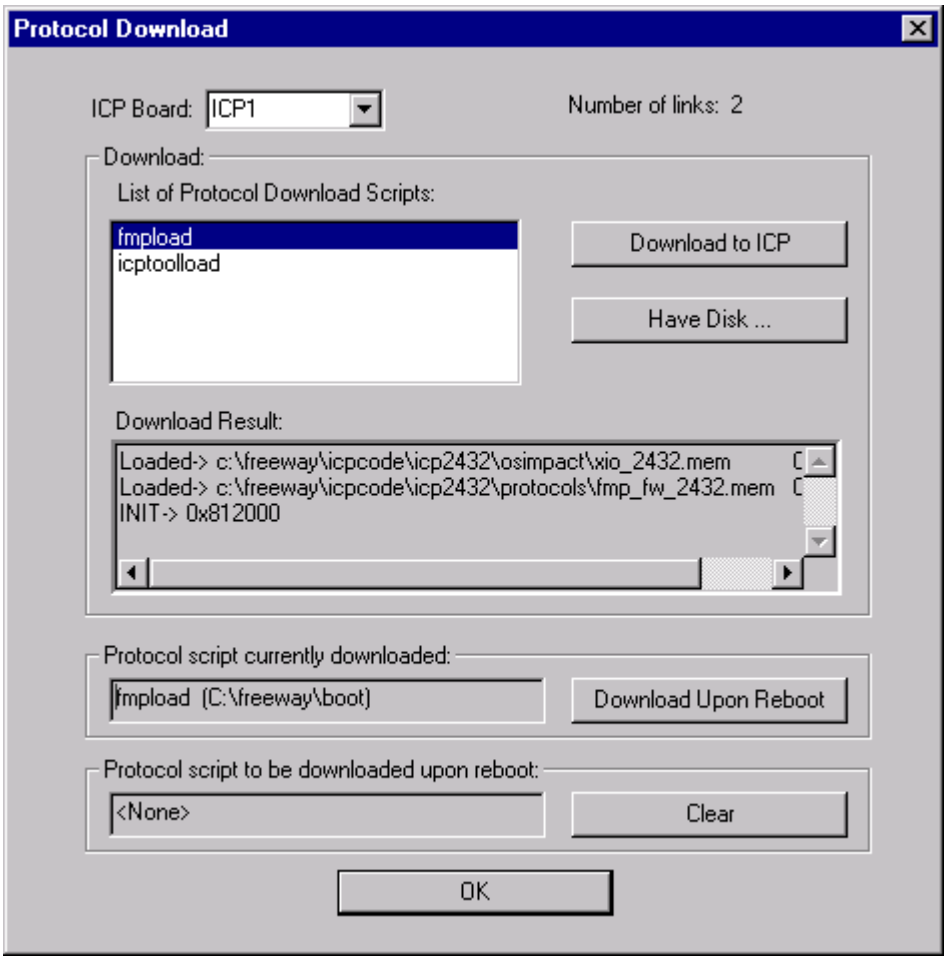

Figure A–4: Protocol Download Menu

A.1.1.1 Download Protocol Confirmation

A successful *Download to ICP* request is confirmed by the *Protocol Download Confirmation*. An example is shown in Figure A–5. Click `OK`.

Figure A–5: Protocol Download Confirmation

A.1.1.2 Specifying a Protocol Download Script

If you select `Have Disk` from the *Protocol Download Menu*, a browser window appears to locate the user-defined download script file, which is then used to download a protocol to the selected ICP.

Note

Specify the name of the `.mem` file in the download script file. All `.mem` files are in the `boot` directory.

After download completion, the *Protocol Download Confirmation* (Figure A–5) is displayed. Click `OK`.

A.1.2 Protocol Diagnostics

Select `Protocol Diagnostics` from the *ICPTool Main Menu* to display the *Protocol Diagnostics Menu* (Figure A–6). A `List of Protocol Diagnostics` is provided.

A.1.2.1 Run Protocol Diagnostics

To run the diagnostic tests, highlight the desired entries in the list and select `Run Diagnostics`. Table A–2 summarizes the menu selections.

The `List of Protocol Diagnostics` varies depending on your system configuration. The `Generic Diagnostics` test is always included with the ICPTool product. If you select the `Generic Diagnostics` test, Section A.1.2.2 on page 72 explains the menu sequence.

Table A–2: Protocol Diagnostics Menu Selections

Button Selected	Action
Run Diagnostics	The tests highlighted in the List of Protocol Diagnostics are run. The results are displayed in a report window.
Select All	All tests in the list are highlighted.

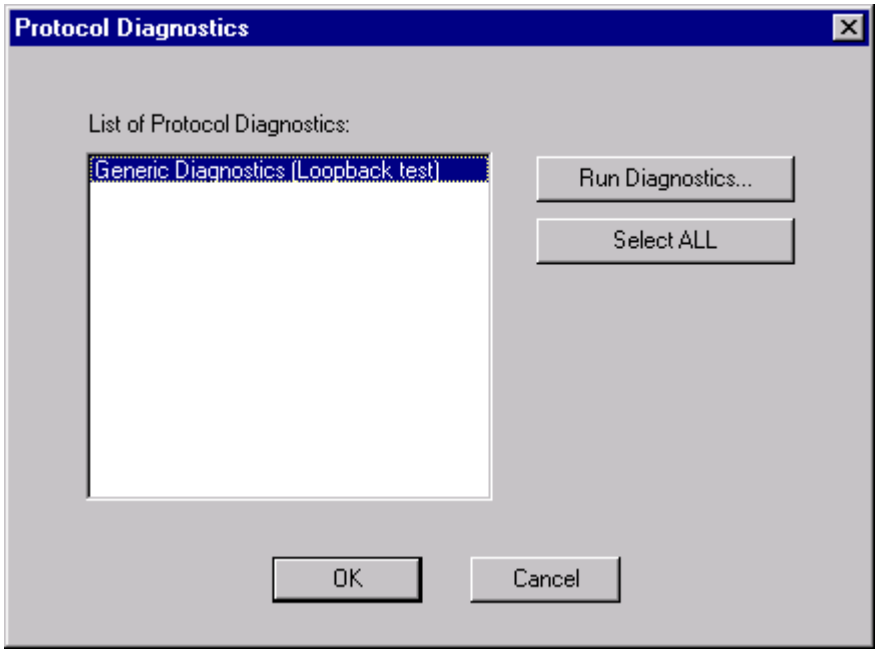

Figure A–6: Protocol Diagnostics Menu

A.1.2.2 Generic Diagnostic (Loopback) Test

Caution

This is a loopback test, so make sure you have the loopback cable connected on the ICP. This diagnostic only works with the `ICPToolLoad` protocol script.

When you select `Generic Diagnostics (Loopback test)` from the *Protocol Diagnostics Menu* (Figure A–6 on page 71), a warning message appears (Figure A–7) asking you to make sure the `ICPToolLoad` protocol script has been downloaded to the ICP. If you click "Yes" when asked if you want to continue, the *Generic Diagnostic Main Menu* appears as shown in Figure A–8. You can run the test with the default configuration (Section A.1.2.3) or control the entire test process interactively using the button selections from the *Generic Diagnostic Main Menu* (Section A.1.2.4 through Section A.1.2.9).

Note

For most sites, you can select `Run Default` to verify the ICP hardware and software installation.

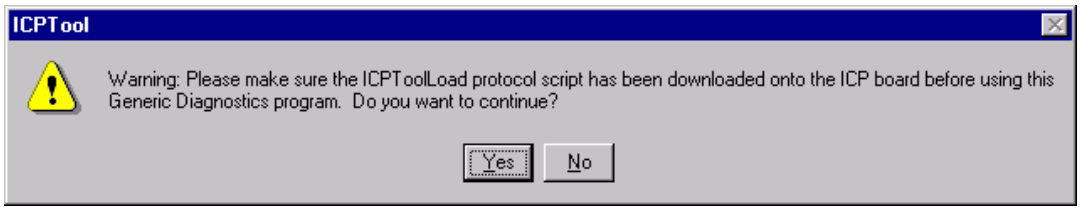

Figure A–7: Generic Diagnostic Warning

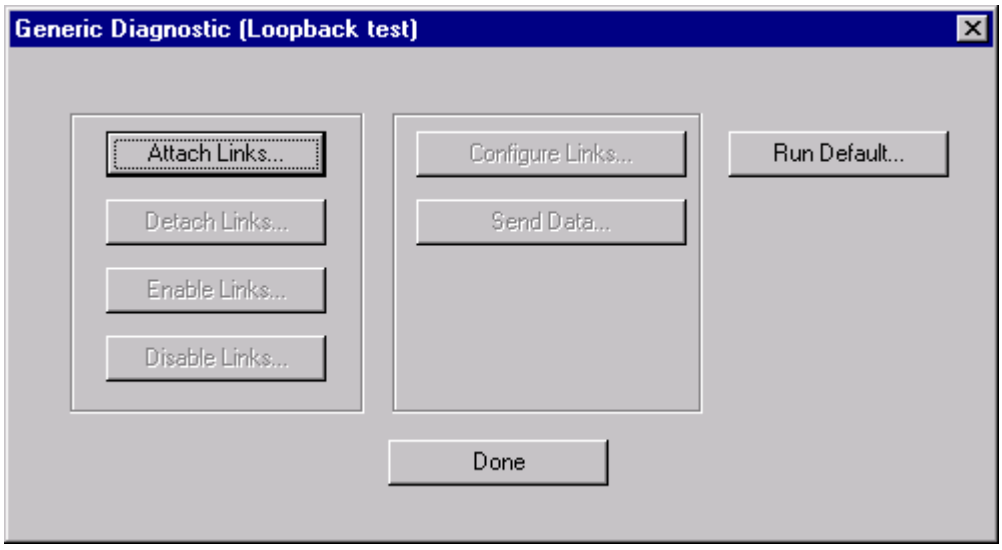

Figure A–8: Generic Diagnostic Main Menu

A.1.2.3 Default Configuration Menu

When you select `Run Default` from the *Generic Diagnostic Main Menu*, the *Default Configuration Menu* appears as shown in Figure A–9. You can run the generic test with the displayed defaults or you can reconfigure parameters prior to running the default test (pulldown menus are provided for some parameters). Select `Run Test` when you are ready to run the test.

Note

For most sites, the default configuration is adequate for running the generic protocol diagnostic to verify the ICP hardware and software installation.

Note

The menus in Section A.1.2.4 through Section A.1.2.9 allow you to control the entire generic test interactively using the button selections from the *Generic Diagnostic Main Menu* (page 73).

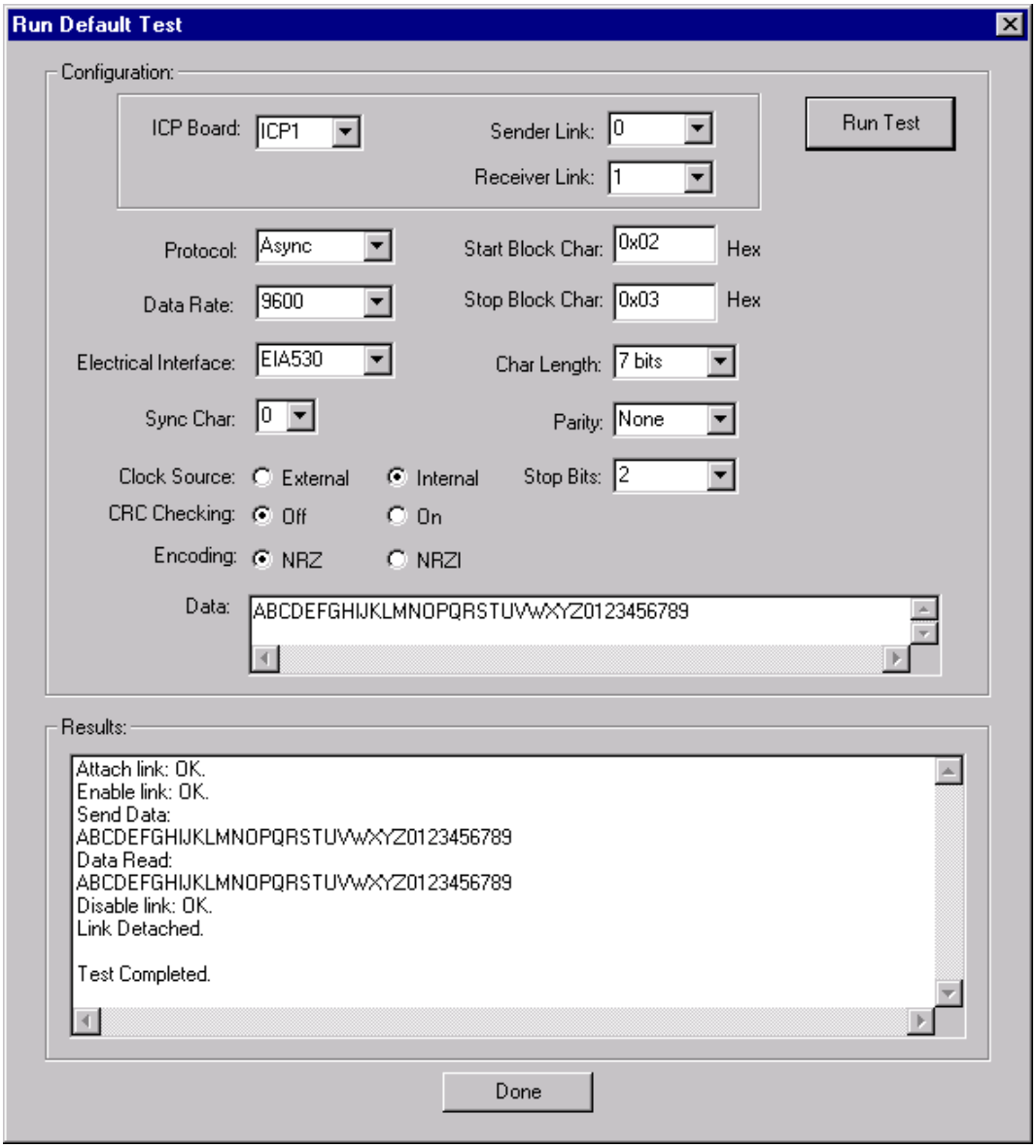

Figure A–9: Default Configuration Menu

A.1.2.4 Attach Link Menu

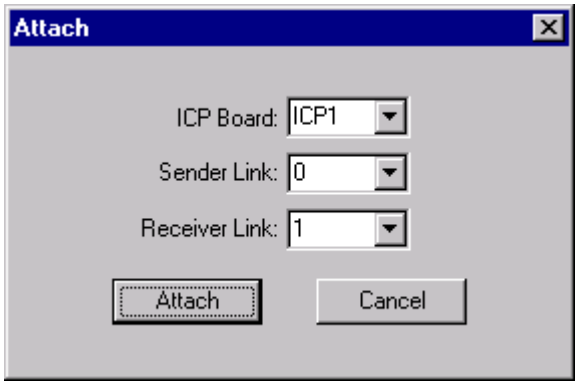

Figure A–10: Attach Link Menu

A.1.2.5 Configure Link Menu

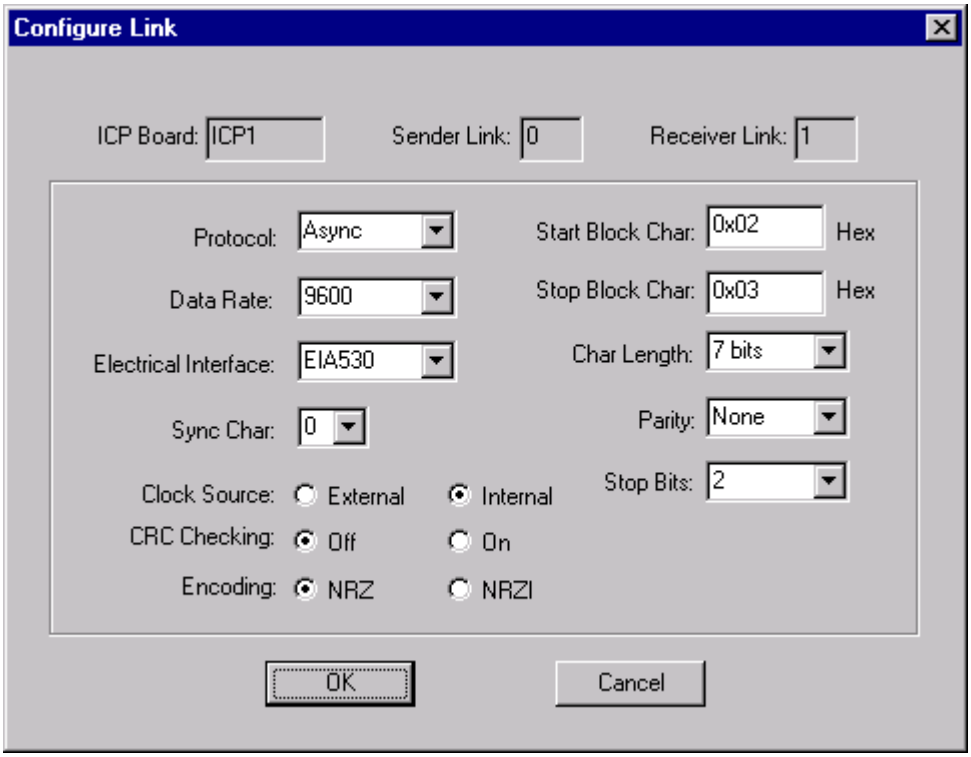

Figure A–11: Configure Link Menu

A.1.2.6 Enable Link Menu

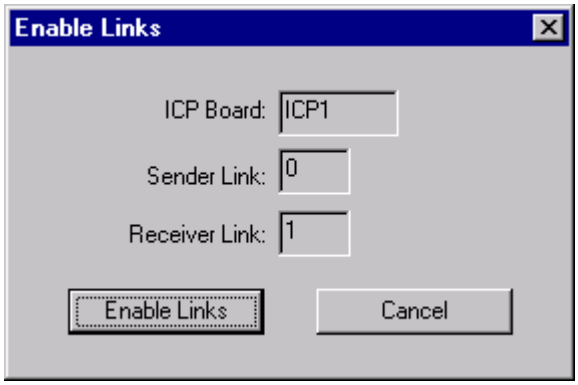

Figure A–12: Enable Link Menu

A.1.2.7 Send Data Menu

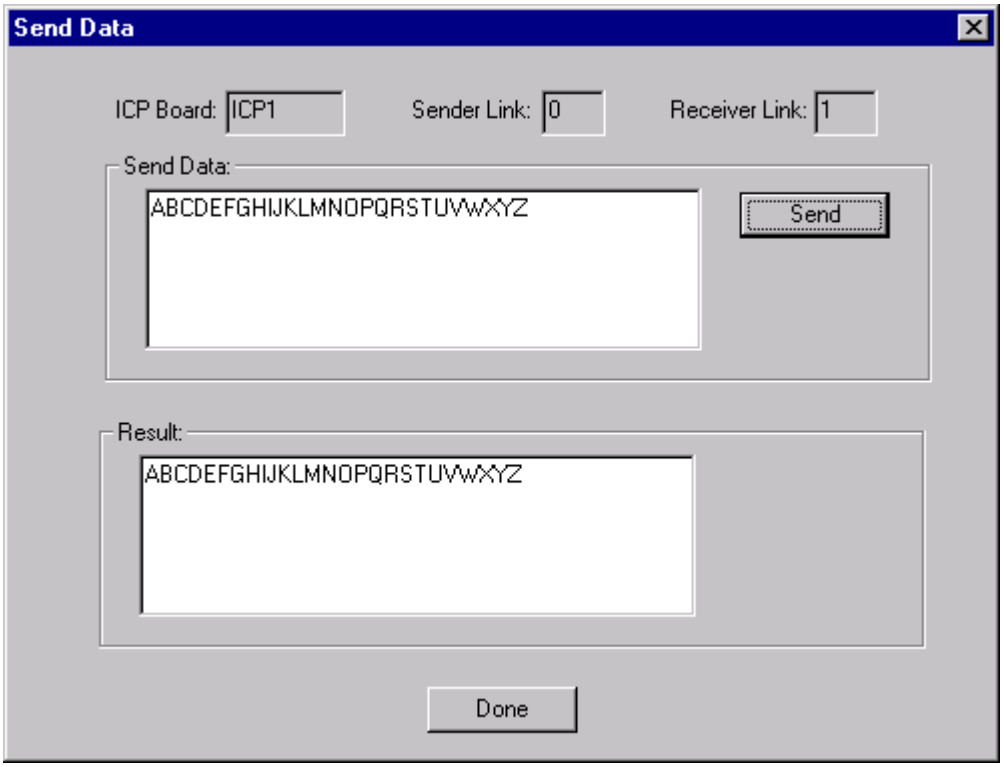

Figure A–13: Send Data Menu

A.1.2.8 Disable Link Menu

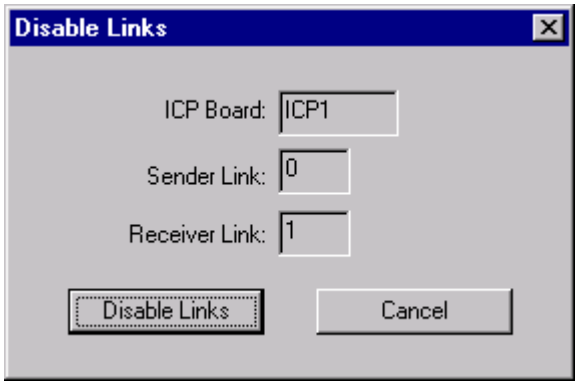

Figure A–14: Disable Link Menu

A.1.2.9 Detach Link Menu

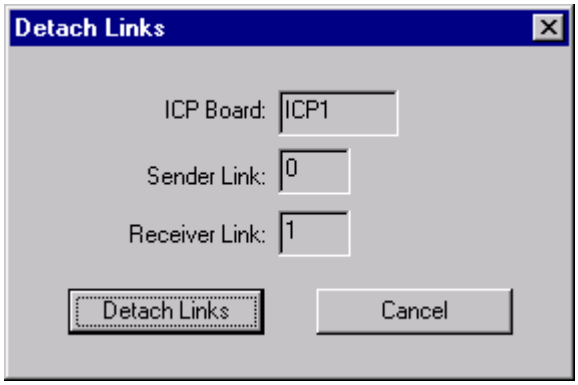

Figure A–15: Detach Link Menu

A.1.3 Advanced Options

Select `Advanced Options` from the *ICPTool Main Menu* to display the *Advanced Options Menu* (Figure A–16). Click `Yes` to automatically start the ICP2432 driver upon reboot. Click `Uninstall` to uninstall the ICP2432 software.

Figure A–16: Advanced Options Menu

A.1.3.1 Event Viewer

Run the *Event Viewer* to verify which protocols are downloaded during reboot.

Step 1:

Start the *Event Viewer*, typically from the "Start → Programs → Administrative Tools → Event Viewer" menu.

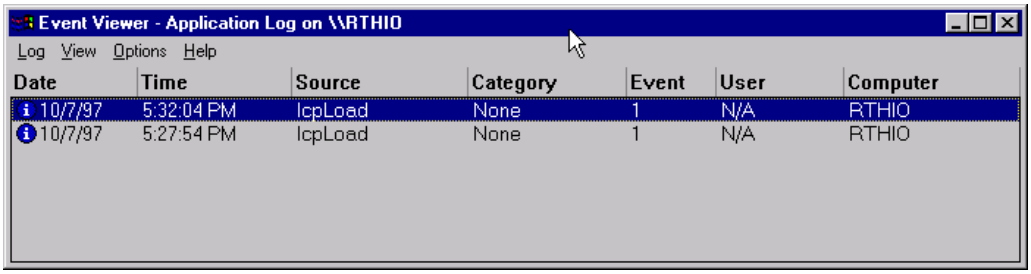

Figure A–17: Event Viewer

Step 2:

Select "Log → Application" to view the results of the automatic download.

Step 3:

To view the event log of a remote computer, select "Log → Select Computer," enter the computer name at the `Computer` entry box, and click `OK`.

Step 4:

To view the detail of an event in the *Event Viewer*, double click on the entry to display the *Event Detail Output* as shown in Figure A–18.

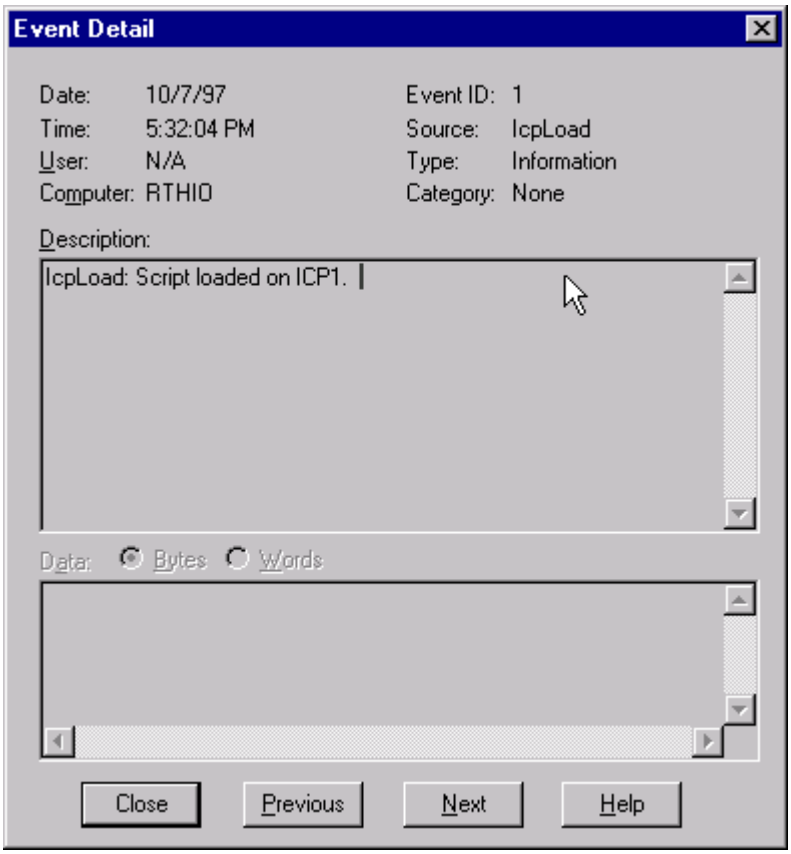

Figure A–18: Event Detail Output

Appendix

B Debug Support for ICP-resident Software

Simpact's Protocol Toolkit product allows users to develop ICP-resident protocol software. During software development, application programmers will probably need to set breakpoints to halt program execution while examining data structures and program flow. However, the Windows NT device driver for the ICP2432 uses a watchdog timer when sending commands to the ICP, so hitting a breakpoint in the debugger can cause the host driver to time out, resulting in the ICP being reset (and all pending I/O requests on the host to be completed with an error code of `Error_Operation_Aborted`).

To allow developers to set breakpoints without having the ICP reset by the host driver, Simpact ships two versions of the driver. During product installation, a copy of each version is placed in the `C:\freeway\client\int_nt_emb` directory (for Intel) or the `C:\freeway\client\axp_nt_emb` directory (for Alpha). `Icp2432.sys` is the "production" version and is also installed in the system drivers directory during installation. `Icp2432_Dbg.sys` is the "debug" version and must be installed manually. The difference between the two versions is that watchdog timers are disabled in the debug version.

To substitute the debug version for the production version, the following steps must be performed on the host machine:

1. Close the Event Viewer if it is currently open.

2. Delete `Icp2432.sys` from the `%SystemRoot%\System32\Drivers` directory (`%SystemRoot%` usually translates to `C:\WINNT`).

3. Copy `Icp2432_Dbg.sys` from the `C:\freeway\client\int_nt_emb` or `axp_nt_emb` directory into `%SystemRoot\System32\Drivers`.

4. Rename the new copy (in the system drivers directory) from `Icp2432_Dbg.sys` to `Icp2432.sys`.

5. Stop the currently installed driver. Go to Start, then Settings, then click Control Panel. From the Control Panel, click on the Devices icon, then select `Icp2432`. Click the Stop button if it is enabled.

6. Start the driver. There are two ways to do this. The first way is to go to Start, then Settings, then click Control Panel. From the Control Panel, click on the Devices icon, then select `Icp2432`. Click the Start button. The second way is to execute `IcpTool` which automatically starts the driver.

7. Verify that the debug version of the driver is running by starting `IcpTool` and clicking the "About IcpTool..." button. The driver version string should contain an extra "-NWT" in it, indicating 'N'o 'W'atchdog 'T'imers.

ICP-resident software may now be debugged without worry. Two things need to be noted, however. First, `IcpTool` will appear as if it is hung when downloading a protocol to the ICP because it is waiting for the host driver to complete the last request, and the driver in turn is waiting for a response from the ICP (which will have hit an initial breakpoint in the debug module linked with the operating system). When the breakpoint is exited and the ICP-resident software resumes execution, `IcpTool` will complete normally.

The second item to note is that watchdog timers are disabled! Therefore, if the ICP software crashes, hangs, or does anything abnormal so that it cannot respond to the host driver, then the host driver is hung; it cannot be stopped, nor can it be used any further. The host machine must be restarted when this occurs (select Restart from the Start → Shutdown icon and click the OK button).

After development of the ICP-resident software has completed, the procedure given above may be followed to reinstall the production version of the driver, with the following adjustments:

1. Omit Step 2.
2. In Step 3, change `Icp2432_Dbg.sys` to `Icp2432.sys`.
3. Omit Step 4. (Step 3 overwrites the debug version of the driver, which is why Step 2 and Step 4 may be omitted)
4. In Step 7, the driver version string should not contain "-NWT".

Appendix

C ADCCP NRM Loopback Test Procedure

This appendix describes the ADCCP NRM loopback test procedure, including the following:

- an overview of the test
- a description of how to install the hardware needed for the test
- instructions on how to run the test
- sample screen displays from the test

C.1 Overview of the Test Program

The ADCCP NRM loopback test program is placed in the `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin` directory during installation.

Note

Earlier Simpact terminology used the term “synchronous” for blocking I/O and “asynchronous” for non-blocking I/O. Some parameter names reflect the previous terminology.

One high-level test program written in C is supplied with the ADCCP NRM protocol, `nrmalp`, which uses non-blocking I/O. The program is interactive; it prompts you for all the information needed to run the test. The test communicates with the ICP through the client data link interface (DLI) commands.

The loopback test performs the following functions:

- Configures the link-level control parameters such as baud rates, clocking, and protocol
- Enables and disables links
- Initiates the transmission and reception of data on the serial lines
- Obtains link statistics from the ICP

You can use the loopback test to verify that the installed devices and cables are functioning correctly. You can also use it as a template for designing client applications that interface with the DLI layer.

C.2 Hardware Setup for the Test Program

The test program runs in loopback mode. Before running the test, perform the following procedure to install the loopback cabling:

Step 1:
Provide a synchronous modem. Configure the modem to supply continuous clocking at a data rate of 2400 bits per second. The ICPs are default configured for external clocking, and the modem supplies the clock signal for loopback testing.

Step 2:
Select a pair of adjacent ports to loopback. Ports are looped back in the following pairs: (0,1), (2,3), (4,5), and so on. Install the special three-headed loopback cable between the ports you selected and the synchronous modem.

Note

The loopback cable is only used during testing, not during normal operation.

C.3 Running the Test Program

Step 1:

Change to the directory that contains the test program: `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin`. Enter the following command at the system prompt:

```
nrmalp
```

Step 2:

The following prompts are displayed. Defaults are shown in brackets:

`Need help [n]?`

Enter **n** to proceed without help. Enter **y** to view a brief description of the test procedure.

`Minutes to run (1-1440) [1]`

Enter the number of minutes you want the test to run.

`ICP board on which to run test (0-3) [0]`

Enter the number of the ICP to be tested. This is the ICP that you cabled for testing in Section C.2, Step 2.

`Even port number (0, 2, ..., 14) [0]`

Enter the even-numbered port you cabled for testing in Section C.2, Step 2. For example, if you enter 0, the loopback test will be performed on ports 0 and 1.

Step 3:

After you answer the last prompt, the test starts. It displays a series of periods, greater than (>) symbols, or less than (<) symbols to indicate that it is running. When it completes, it displays the test results in the form of a brief Statistics Report that shows activity on the two ports being tested. If no errors are shown, your installation is verified.

Step 4:

Remove the loopback cable and configure the cables for normal operation.

C.4 Sample Output from Test Program

Figure C–1 shows the screen display from a sample ADCCP NRM non-blocking loopback test program (`nrmalp`). Output displayed by the program is shown in `typewriter type` and your responses are shown in **bold type**. Each entry is followed by a carriage return.

```
% nrmalp
Need help [n]? y

      This program transfers data between a pair of adjacent ports on an ICP
      board. These ports must be connected with the supplied Simpact THREE-
      headed loopback cable. The third head of the cable must be connected
      to your powered up modem. Your modem supplies clocking to move the
      data. The data does not reach the modem, but the program does not
      work without an external clock source. The configuration file,
      nrmaldcfg, specifies an external clock source, i.e. modem-supplied
      clocking. The ICP and the distribution panel jumpers are configured
      at the factory for external clocks. The first ICP is zero; the first
      port is zero. The program defaults to ICP board zero, ports zero and
      one.

      When prompted for values, the range of legal values appears within
      parentheses immediately following the prompt. The default value then
      appears within square brackets. To select the default value, simply
      press the RETURN key. To select a value other than the default, enter
      the desired value followed by the RETURN key.

Minutes to run (1-1440) [1]? 1
ICP board on which to run test (0-3) [0]? 1
Even port number (0, 2, ..., 14) [0]? 0

ADCCP NRM Asynchronous Port-to-Port LOOP BACK program.
   Test duration of 1 minute
   ICP board number 1
   Ports 0 & 1
INIT COMPLETED
OPEN SESSION server0icp1port0
OPEN SESSION server0icp1port1
WAIT FOR SESSION server0icp1port0 TO BECOME ACTIVE
WAIT FOR SESSION server0icp1port1 TO BECOME ACTIVE
COMPLETED dlOpen
ADCCP NRM version:
@(#) Simpact ADCCP NRM for FREEWAY 2000 - V03.3 13-Dec-95, OS/Impact - V1.6
ADCCP NRM (ANSI X3.66-1979)         13-DEC-1995 - 3.1.4

..<.>><.>.><<<<>>..>.>>>>>><..>.>.<<<<>>>>>.>>>>>><>>>>>><.>>>>>><>>>>..>>.<
<<<.>>>>>.<<<<>..>>.>>>>>><..>.<<<<>>>..>.>>>>>><..>.<<<>>>..>.>>>>>><..>.<<<<>>
>.<<>>.>.>>>>>><.>>>.<<<<>>><.>>
DONE READS / WRITES
```

Figure C–1: Sample Output: NRM Non-blocking Loopback Program (nrmalp)

```
ADCCP NRM Statistics Report ICP1
Links                0      1
               --------------
inv addresses        0      0
inv ctlfields        0      0
rcv FCS errs         0      0
lfrm too long        0      0
rcv overruns         0      0
txmt underruns       0      0
txmt wtchdg          0      0
stn resets           0      0
               --------------
data writes        533    534
data reads         535    533

CLOSING SESSIONS
Closing Session 0
Closing Session 1
Waiting for all sessions closed
Run time: 60 seconds.
nrmalp completed OK.
```

Figure C–1: Sample Output: NRM Non-blocking Loopback Program (nrmalp) *(Cont'd)*

Appendix

D AWS Loopback Test Procedure

This appendix describes the Asynchronous Wire Service (AWS) loopback test procedure, including the following:

- an overview of the test
- a description of how to install the hardware needed for the test
- instructions on how to run the test
- sample screen displays from the test

D.1 Overview of the Test Program

The AWS loopback test program is placed in the `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin` directory during installation.

Note

Earlier Simpact terminology used the term “synchronous” for blocking I/O and “asynchronous” for non-blocking I/O. Some parameter names reflect the previous terminology.

One high-level test program written in C is supplied with the AWS protocol: `awsalp`, which uses non-blocking I/O. The program is interactive; it prompts you for all the information needed to run the test. The test communicates with the ICP through the client data link interface (DLI) commands.

The loopback test performs the following functions:

- Configures the link-level control parameters such as baud rates, clocking, and protocol
- Enables and disables links
- Initiates the transmission and reception of data on the serial lines
- Obtains link statistics from the ICP

You can use the loopback test to verify that the installed devices and cables are functioning correctly. You can also use it as a template for designing client applications that interface with the DLI layer.

D.2 Hardware Setup for the Test Program

The test program runs in loopback mode. Before running the test, perform the following procedure to install the loopback cabling:

Step 1:
Provide a synchronous modem. Configure the modem to supply continuous clocking at a data rate of 9600 bits per second. The ICPs are default configured for external clocking, and the modem supplies the clock signal for loopback testing.

Step 2:
Select a pair of adjacent ports to loopback. Ports are looped back in the following pairs: (0,1), (2,3), (4,5), and so on. Install the special three-headed loopback cable between the ports you selected and the synchronous modem.

Note

The loopback cable is only used during testing, not during normal operation.

D.3 Running the Test Program

Step 1:

Change to the directory that contains the test program: `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin`. Enter the following command at the system prompt:

```
awsalp
```

Step 2:

The following prompts are displayed. Defaults are shown in brackets:

`Need help [n]?`

Enter **n** to proceed without help. Enter **y** to view a brief description of the test procedure.

`Minutes to run (1-1440) [1]`

Enter the number of minutes you want the test to run.

`ICP board on which to run test (0-3) [0]`

Enter the number of the ICP to be tested. This is the ICP that you cabled for testing in Step 2 on page 96.

`Even port number (0, 2, ..., 14) [0]`

Enter the even-numbered port you cabled for testing in Step 2 on page 96. For example, if you enter 0, the loopback test will be performed on ports 0 and 1.

Step 3:

After you answer the last prompt, the test starts. It displays a series of periods, greater than (>) symbols, or less than (<) symbols to indicate that it is running. If no errors are shown, your installation is verified.

Step 4:

Remove the loopback cable and configure the cables for normal operation.

D.4 Sample Output from Test Program

Figure D–1 shows the screen display from a sample AWS non-blocking loopback test program (`awsalp`). Output displayed by the program is shown in `typewriter type` and your responses are shown in **bold type**. Each entry is followed by a carriage return

```
% awsalp
Need help [n]? y

      This program transfers data between a pair of adjacent ports on an ICP
      board. These ports must be connected with the supplied Simpact THREE-
      headed loopback cable. The third head of the cable must be connected
      to your powered up modem. Your modem supplies clocking to move the
      data. The data does not reach the modem, but the program does not
      work without an external clock source. The configuration file,
      awsaldcfg, specifies an external clock source, i.e., modem-supplied
      clocking. The ICP and the distribution panel jumpers are configured
      at the factory for external clocks. The first ICP is zero; the first
      port is zero. The program defaults to ICP board zero, ports zero and
      one.

      When prompted for values, the range of legal values appears within
      parentheses immediately following the prompt. The default value then
      appears within square brackets. To select the default value, simply
      press the RETURN key. To select a value other than the default, enter
      the desired value followed by the RETURN key.

Minutes to run (1-1440) [1]? 1
ICP board on which to run test (0-3) [0]? 0
Even port number (0, 2, ..., 14) [0]? 0

AWS Asynchronous Port-to-Port LOOP BACK program.
   Test duration of 1 minute
   ICP board number 0
   Ports 0 & 1
INIT COMPLETED
OPEN SESSION server0icp0port0
OPEN SESSION server0icp0port1
COMPLETED dl0pen
.>>.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<
<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.
>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<.>>.<<

CLOSING SESSIONS
Closing Session 0
Closing Session 1
Waiting for all sessions closed
Run time: 66 seconds.
awsalp completed OK.
```

Figure D–1: Sample Output: AWS Non-Blocking Loopback Program (awsalp)

Appendix E

BSC Loopback Test Procedure

This appendix describes the BSC loopback test procedure, including the following:

- an overview of the test
- a description of how to install the hardware needed for the test
- instructions on how to run the test
- a sample screen display from the test

E.1 Overview of the Test Program

The BSC loopback test program is placed in the `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin` directory during the installation procedures. Each BSC protocol has its own loopback test program as shown in Table 5–1.

Note

Earlier Simpact terminology used the term “synchronous” for blocking I/O and “asynchronous” for non-blocking I/O. Some parameter names reflect the previous terminology.

Table 5–1: BSC Protocol Loopback Test Programs

Protocol	Type of I/O[a]	Test Program	TSI Configuration File
BSC3270	Non-blocking	`bsc3270alp`	`bsc3270altcfg`
BSC3780	Non-blocking	`bsc3780alp`	`bsc3780altcfg`

[a] The type of I/O is set in the `Async10` parameter of the TSI configuration file in the `freeway\client\test\`*`filename`* directory.

One high-level test program written in C is supplied with each BSC protocol, `bsc3270alp` or `bsc3780alp`, which use non-blocking I/O. The program is interactive; it prompts you for all the information needed to run the test. The test communicates with the ICP through the client data link interface (DLI) commands.

The loopback test performs the following functions:

- Configures the link-level control parameters such as baud rates, clocking, and protocol
- Enables and disables links
- Initiates the transmission and reception of data on the serial lines
- Obtains link statistics from the ICP

You can use the loopback test to verify that the installed devices and cables are functioning correctly. You can also use it as a template for designing client applications that interface with the DLI layer.

E.2 Hardware Setup for the Test Program

The test program runs in loopback mode. Before running the test, perform the following procedure to install the loopback cabling:

Step 1:
Provide a synchronous modem. Configure the modem to supply continuous clocking at a data rate between 300 and 19,200 bits per second. The ICPs are default configured for external clocking, and the modem supplies the clock signal for loopback testing.

Step 2:
Select a pair of adjacent ports to loopback. Ports are looped back in the following pairs: (0,1), (2,3), (4,5), and so on. Install the special three-headed loopback cable between the ports you selected and the synchronous modem.

Note

The loopback cable is only used during testing, not during normal operation.

E.3 Running the Test Program

Step 1:
Change to the directory that contains the test program: `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin`. Enter one of the following commands at the system prompt:

```
bsc3270alp
```

or

```
bsc3780alp
```

Step 2:

The following prompts are displayed. Defaults are shown in brackets:

`Need help (Y/N) [N]?`

Enter **n** to proceed without help. Enter **y** to view a brief description of the test procedure.

`Minutes to run (1-1440) [1]?`

Enter the number of minutes you want the test to run.

`ICP board on which to run test (0-5) [0]?`

Enter the number of the ICP to be tested. This is the ICP that you cabled for testing in Step 2 on page 103.

`Even port number (0, 2, ..., 14) [0]?`

Enter the even-numbered port you cabled for testing in Step 2 on page 103. For example, if you enter 0, the loopback test will be performed on ports 0 and 1.

Step 3:

After you answer the last prompt, the test starts. It displays a series of periods, greater than (>) symbols, or less than (<) symbols to indicate that it is running. When it completes, it displays the test results in the form of a brief Statistics Report that shows activity on the two ports being tested. If no errors are shown, your installation is verified.

Step 4:

Remove the loopback cable and configure the cables for normal operation.

E.4 Sample Output from Test Program

Figure E–1 shows the screen display from a sample BSC3780 non-blocking loopback test program (`bsc3780alp`). Output displayed by the program is shown in `typewriter type` and your responses are shown in **bold type**. Each entry is followed by a carriage return.

```
% bsc3780alp
Need help (Y/N) [N]? Y

This program transfers data between a pair of adjacent ports on an ICP
board.  These ports must be connected with the supplied Simpact THREE-
headed loopback cable.  The third head of the cable must be connected
to your powered up modem.  Your modem supplies clocking to move the
data.  The data does not reach the modem, but the program does not
work without an external clock source.  The configuration file,
bsc3780aldcfg, specifies an external clock source, i.e. modem-
supplied clocking.  The ICP and the distribution panel jumpers are
configured at the factory for external clocks.  The first ICP is zero;
the first port is zero.  The program defaults to ICP board zero, ports
zero and one.

When prompted for values, the range of legal values appears within
parentheses immediately following the prompt.  The default value then
appears within square brackets.  To select the default value, simply
press the RETURN key.  To select a value other than the default, enter
the desired value followed by the RETURN key.

Minutes to run (1-1440) [1]? 1
ICP board on which to run test (0-5) [0]? 0
Even port number (0, 2, ..., 14) [0]? 0

BSC 2780/3780 Asynchronous Port-To-Port Loopback Program.
   Test duration in minutes: 1 minute
   ICP board number: 0
   Ports: 0 & 1

BSC 2780/3780 Software Version:
   <@@> VI-100-0186: BSC 2.0-0  BSC 2780/3780 16-June-98
 OS/Impact Version  V331

BSC 2780/3780 ICP Buffer Report:
      1024 ICP message buffer size
       278 Number of free ICP message buffers
       297 Total number of ICP message buffers
       512 Transmission buffer size
         1 Number of free transmission buffers
         1 Total number of transmission buffers
        16 Total number of links
><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>< - 1
><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>< - 2
><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>> - 3
```

Figure E–1: Sample Output from BSC3780 Non-Blocking Loopback Program

```
BSC 2780/3780 Statistics Report:
                              server0icp0port0      server0icp0port1
                              ----------------      ----------------
   Block check errors                    0                     0
   Parity errors                         0                     0
   Receive overrun errors                0                     0
   Buffer errors                         0                     0
   Messages sent                       338                   338
   Messages received                   338                   338
   NAKs sent                             0                     0
   NAKs received                         0                     0
   Buffer errors on send                 0                     0
   Transmission blocks sent            338                   338
   Transmission blocks received        338                   338

Loopback test complete
```

Figure E–1: Sample Output from BSC3780 Non-Blocking Loopback Program *(Cont'd)*

Appendix

F FMP Loopback Test Procedure

This appendix describes the FMP loopback test procedure, including the following:

- an overview of the test
- a description of how to install the hardware needed for the test
- instructions on how to run the test
- a sample screen display from the test

F.1 Overview of the Test Program

The FMP loopback test program is placed in the `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin` directory during the installation procedures.

Note

Earlier Simpact terminology used the term “synchronous” for blocking I/O and “asynchronous” for non-blocking I/O. Some parameter names reflect the previous terminology.

One high-level test program written in C is supplied with each FMP protocol, `fmpalp`, which use non-blocking I/O. The program is interactive; it prompts you for all the information needed to run the test. The test communicates with the ICP through the client data link interface (DLI) commands.

The loopback test performs the following functions:

- Configures the link-level control parameters such as baud rates, clocking, and protocol
- Enables and disables links
- Initiates the transmission and reception of data on the serial lines
- Obtains link statistics from the ICP

You can use the loopback test to verify that the installed devices and cables are functioning correctly. You can also use it as a template for designing client applications that interface with the DLI layer.

F.2 Hardware Setup for the Test Program

The test program runs in loopback mode. Before running the test, perform the following procedure to install the loopback cabling:

Step 1:
Provide a synchronous modem. Configure the modem to supply continuous clocking at a data rate between 300 and 19,200 bits per second. The ICPs are default configured for external clocking, and the modem supplies the clock signal for loopback testing.

Step 2:
Select a pair of adjacent ports to loopback. Ports are looped back in the following pairs: (0,1), (2,3), (4,5), and so on. Install the special three-headed loopback cable between the ports you selected and the synchronous modem.

Note

The loopback cable is only used during testing, not during normal operation.

F.3 Running the Test Program

Step 1:

Change to the directory that contains the test program: `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin`. Enter the following command at the system prompt:

```
fmpalp
```

Step 2:

The following prompts are displayed. Defaults are shown in brackets:

`Need help (Y/N) [N]?`

Enter **n** to proceed without help. Enter **y** to view a brief description of the test procedure.

`Minutes to run (1-1440) [1]?`

Enter the number of minutes you want the test to run.

`Number of initial writes (1-4) [1]?`

Enter the number of writes to be allowed before a response.

`ICP board on which to run test (0-5) [0]?`

Enter the number of the ICP to be tested. This is the ICP that you cabled for testing in Step 2 on page 108.

`Even port number (0, 2, ..., 14) [0]?`

Enter the even-numbered port you cabled for testing in Step 2 on page 108. For example, if you enter 0, the loopback test will be performed on ports 0 and 1.

Step 3:

After you answer the last prompt, the test starts. It displays a series of periods, greater than (>) symbols, or less than (<) symbols to indicate that it is running. When it completes, it displays the test results in the form of a brief Statistics Report that shows activity on the two ports being tested. If no errors are shown, your installation is verified.

Step 4:

Remove the loopback cable and configure the cables for normal operation.

F.4 Sample Output from Test Program

Figure F–1 shows the screen display from a sample FMP non-blocking loopback test program (`fmpalp`). Output displayed by the program is shown in `typewriter type` and your responses are shown in **bold type**. Each entry is followed by a carriage return. .

```
% fmpalp
Need help (Y/N) [N]? y

    This program transfers data between a pair of adjacent ports on an ICP
    board.  These ports must be connected with the supplied Simpact THREE-
    headed loopback cable.  The third head of the cable must be connected
    to your powered up modem.  Your modem supplies clocking to move the
    data.  The data does not reach the modem, but the program does not
    work without an external clock source.  The configuration file,
    fmpaldcfg, specifies an external clock source, i.e. modem-
    supplied clocking.  The ICP and the distribution panel jumpers are
    configured at the factory for external clocks.  The first ICP is zero;
    the first port is zero.  The program defaults to ICP board zero, ports
    zero and one.

    When prompted for values, the range of legal values appears within
    parentheses immediately following the prompt.  The default value then
    appears within square brackets.  To select the default value, simply
    press the RETURN key.  To select a value other than the default, enter
    the desired value followed by the RETURN key.

Minutes to run (1-1440) [1]? 1
Number of initial writes (1-4) [1]? 2
ICP board on which to run test (0-3) [0]? 0
Even port number (0, 2, ..., 14) [0]? 0

FMP Asynchronous Port-To-Port Loopback Program.
   Test duration in minutes: 1 minute
   ICP board number: 0
   Ports: 0 & 1

FMP Software Version:
@(#) Simpact FMP (Financial Market Protocols) - V1.5  22-Jan-96
     for the Freeway 2000/4000/8800 server  (ICP6000)
(OS/Impact Version  V1.6 )
```

Figure F–1: Sample Output from FMP Non-Blocking Loopback Program

```
FMP ICP Buffer Report:
      1024 ICP message buffer size
       523 Number of free ICP message buffers
       592 Total number of ICP message buffers
      1019 Transmission buffer size
        30 Number of free transmission buffers
        30 Total number of transmission buffers
         8 Total number of links
 ><><<><<>><><<><<>><<><><<>><<><<>><><<>><<><<><><<>><<><<>><><<>><<><<>><<><<>><<><
 ><>><<><<>><<><<>><<><<>><><<>><<><<>><<><>><<><<><<>><<><<>><<><<>><><<>><<><<><<>><<
 <>><<><>><><<><<>><<><<>><><<><<>><<><<>><><<>><<><<>><<><<><<>><<><<>><<><<>><><<
 ><><<><<>><<><><<><<><<>><<><><<>><<><<><<>><<><<>><<><<>><><<><<><<>><<><<>><<><>>
 <<><<><<><<>><<><<>><<><<><<
FMP Statistics Report:
                                 server0icp0port0     server0icp0port1
                                 -----------------    -----------------
   Block check errors                    0                    0
   Parity errors                         0                    0
   Receive overrun errors                0                    0
   Q limit errors                        0                    0
   Messages sent                       703                  580
   Messages received                   580                  703
   Buffers not available                 0                    0
   Buffer overruns                       0                    0
 Loopback test complete
```

Figure F–1: Sample Output from FMP Non-Blocking Loopback Program *(Cont'd)*

Appendix

G Protocol Toolkit Loopback Test Procedure

This appendix describes the protocol toolkit test procedure, including the following:

- an overview of the test
- a description of how to install the hardware needed for the test
- instructions on how to run the test
- a sample screen display from the test

G.1 Overview of the Test Program

The protocol toolkit loopback test program is placed in the `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin` directory during installation.

Note

Earlier Simpact terminology used the term “synchronous” for blocking I/O and “asynchronous” for non-blocking I/O. Some parameter names reflect the previous terminology.

One high-level test program written in C is supplied with the protocol toolkit: `spsalp`, which uses non-blocking I/O. The program is interactive; it prompts you for all the information needed to run the test. The test communicates with the ICP through the client data link interface (DLI) commands.

The loopback test performs the following functions:

- Configures the link-level control parameters such as baud rates, clocking, and protocol
- Enables and disables links
- Communicates with the ICP and initiates the transmission and reception of data on the serial lines
- Obtains link statistics from the ICP

You can use the loopback test to verify that the installed devices and cables are functioning correctly. You can also use it as a template for designing client applications that interface with the DLI layer.

The test program can configure any of the ICP links to perform one of three methods of communication: bit synchronous (HDLC/SDLC), byte synchronous (BSC), and asynchronous (ASYNC). The synchronous methods use an external modem as the clocking device. No external modem is required for the asynchronous method.

G.2 Hardware Setup for the Test Program

The test program runs in loopback mode. Before running the test, perform the following procedure to install the loopback cabling:

Step 1:
If you are using a synchronous protocol (HDLC/SDLC or BSC), locate a synchronous modem that you can use during the test.

Step 2:

Select a pair of adjacent ports to loopback. Ports are looped back in the following pairs: (0,1), (2,3), (4,5), and so on.

If you are using a synchronous protocol, install the special three-headed loopback cable between the ports you selected and the synchronous modem. It supplies the needed clock signal. Configure the modem to supply continuous clocking at a data rate between 300 and 19,200 bits per second.

If you are using an asynchronous protocol, the male connector of the loopback cable does not have to be attached to anything; however, if you have just run the synchronous test, you do not have to detach the modem before running the asynchronous test.

Note

The loopback cable is only used during testing, not during normal operation.

G.3 Running the Test Program

Step 1:

Change to the directory that contains the test program: `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin`. Enter the following command at the system prompt:

```
spsalp
```

Step 2:

You are asked if you need help. Enter **n** to proceed without help. Enter **y** to view a brief description of the test procedure.

Step 3:

You are prompted for the number of the ICP to be used for the test. This is the ICP that you cabled for testing in Step 2 on page 115.

Step 4:

The protocol toolkit main menu is displayed as shown in Figure G–1. You can select an operation by entering a digit from 0 to 8 followed by a carriage return. Depending on your choice, you are then prompted for additional information. When an operation completes, the main menu is displayed again. You can then choose another operation or exit the program by selecting menu option 8. For samples of the screen displays that you will see during the test, refer to Section G.4 on page 118.

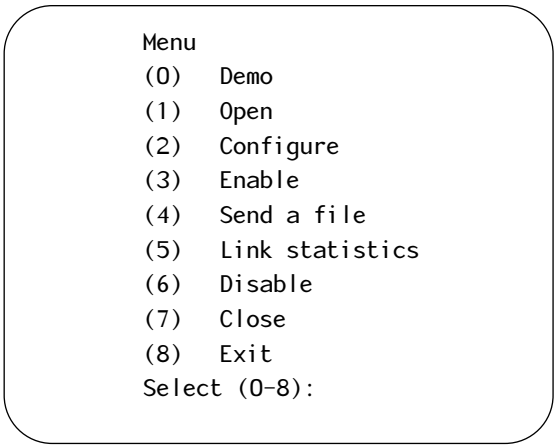

Figure G–1: Main Menu of Protocol Toolkit Test

If you want to use option 0 (Demo), you must have connected links 0 and 1 of ICP 0 in Section G.2 of this procedure. The Demo option automatically opens and enables links 0 and 1, using default link configurations. It then loops back several blocks of data before disabling and closing the links.

Step 5:

To further verify the software installation, use menu options 1, 2, and 3 to open, configure, and enable each of the two links connected to the loopback cable.

For example, suppose the loopback cable is connected to links 2 and 3. You would then:

- Select menu option 1 to open link 2.
- Select menu option 2 to configure link 2. You are prompted for the link configuration parameters. Be sure that the external clock is selected if you are using a synchronous protocol (BSC or HDLC/SDLC).
- Select menu option 3 to enable link 2.
- Select menu options 1, 2, and 3 again, this time for link 3. Be sure to enter the same link configuration parameters for link 3 as you did for link 2.

Step 6:

Select menu option 4 (Send a file) to loop back data between the two links you configured in Step 5. You are prompted for the block size, file name, and transmit and receive link numbers. Normally, you will specify 128 as the block size and `..\..\test\sps\spstest.h` as the file name.

After obtaining the configuration parameters, the test program opens the specified file, reads a block of data, and sends it to the ICP. The ICP-resident protocol task in the server transmits the block from the transmit link. The block is looped back into the receive link. The server then returns the block to the test program which compares it with the block it originally sent. If the blocks do not match, an error is displayed. If they match, the program reads the next block of data from the file and sends it to the ICP, and so on. This process repeats until the entire file has been looped back.

While the loopback process is running, the test program displays a greater than (>) symbol each time it successfully sends a block of data to the ICP. The program displays a less than (<) symbol each time the block that has been looped back matches the block

that was originally sent. If they do not match, an error message is displayed. When the process completes, the main menu is redisplayed.

Step 7:
The installation is verified if the test successfully finishes without errors. Select option 8 (Exit) to exit the program.

Step 8:
Remove the loopback cable and configure the cables for normal operation.

G.4 Sample Output from Test Program

Figure G–2 shows a sample screen display when the Demo option is selected (option 0). Figure G–3 shows a sample screen display for a typical byte synchronous test. Output displayed by the program is shown in `typewriter type` and your responses are shown in **bold type**. Each entry is followed by a carriage return.

```
% spsalp
Need help [n]? n
ICP number to run (0-15) [0]: 0
SPS Asynchronous Port-to-Port LOOP BACK program.

    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 0
Link 0 open.
Link 1 open.
Link 0 configured.
Link 1 configured.
Link 0 enabled.
Link 1 enabled.
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
File ../../test/sps/spstest.data sent on link 1, received on link 0.
Link 0 disabled.
Link 1 disabled.
Link 0 closed.
Link 1 closed.
Demo succeeded.

    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 8
Waiting for quiescence
spsalp complete.
```

Figure G–2: Sample Output from Protocol Toolkit Test Showing Demo Option

```
% spsalp
Need help [n]? y

      This program writes data to an even port and reads it from the adjacent
      higher numbered odd port, e.g. write to port 0 and read from port 1.
      These ports must be connected with the supplied Simpact THREE-headed
      loopback cable.  The third head of the cable must be connected to your
      powered up modem.  Your modem supplies clocking to move the data.  The
      data does not reach the modem, but the program does not work without
      an external clock source.  The configuration file, spssldcfg,
      specifies an external clock source, i.e. modem-supplied clocking.  The
      ICP and the distribution panel jumpers are configured at the factory
      for external clocks.  The first ICP is zero; the first port is zero.
      The program defaults to ICP board zero, ports zero and one.

      When prompted for values, the range of legal values appears within
      parentheses immediately following the prompt.  The default value then
      appears within square brackets.  To select the default value, simply
      press the RETURN key.  To select a value other than the default, enter
      the desired value followed by the RETURN key.

ICP number to run (0-15) [0]: 0
SPS Synchronous Port-to-Port LOOP BACK program.

    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 1
Open link: 2
Link 2 open.
```

Figure G–3: Sample Output from Protocol Toolkit Test Showing a BSC Test

```
    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 2
Configure link: 2

0 = BSC
1 = Asynch
2 = HDLC/SDLC
Communication mode: 0

0 = EIA232
1 = EIA562
2 = EIA530
3 = V.35
Electrical interface: 0

 0 = external
 1 = internal
Clock source: 0

    0 = 300
    1 = 600
    2 = 1200
    3 = 2400
    4 = 4800
    5 = 9600
    6 = 19200
    7 = 38400
    8 = 57600
    9 = 64000
   10 = 263300
   11 = 307200
   12 = 460800
Baud rate: 5
```

Figure G–3: Sample Output from Protocol Toolkit Test Showing a BSC Test *(Cont'd)*

```
   0 = off
   1 = on
CRC generation/checking: 1

Number of sync characters (1 - 8): 3

Hex value of character to start block: 1
Link 2 configured.

    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 3
Enable link: 2
Link 2 enabled.

    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 1
Open link: 3
Link 3 open.
```

Figure G–3: Sample Output from Protocol Toolkit Test Showing a BSC Test *(Cont'd)*

```
    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 2
Configure link: 3

0 = BSC
1 = Asynch
2 = HDLC/SDLC
Communication mode: 0

0 = EIA232
1 = EIA562
2 = EIA530
3 = V.35
Electrical interface: 0

 0 = external
 1 = internal
Clock source: 0

    0 = 300
    1 = 600
    2 = 1200
    3 = 2400
    4 = 4800
    5 = 9600
    6 = 19200
    7 = 38400
    8 = 57600
    9 = 64000
   10 = 263300
   11 = 307200
   12 = 460800
Baud rate: 5
```

Figure G–3: Sample Output from Protocol Toolkit Test Showing a BSC Test *(Cont'd)*

```
   0 = off
   1 = on
CRC generation/checking: 1

Number of sync characters (1 - 8): 3

Hex value of character to start block: 1
Link 3 configured.

    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8):  3
Enable link: 3
Link 3 enabled.

    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 4
Block size: 128
Filename: ../../test/sps/spstest.h
Transmit link: 3
Receive link: 2
```

Figure G–3: Sample Output from Protocol Toolkit Test Showing a BSC Test *(Cont'd)*

```
><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
File ../../test/sps/spstest.h sent on link 3, received on link 2.

    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 5
Read statistics of link: 2

Report of link errors:

Data blocks sent:                0
Data blocks received:           26
Received messages too long:      0
Times DCD lost:                  0
Received messages aborted:       0
Receive overruns:                0
Transmit underruns:              0
Receive parity errors:           0
Receive framing errors:          0
Receive CRC errors:              0

    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 5
```

Figure G–3: Sample Output from Protocol Toolkit Test Showing a BSC Test *(Cont'd)*

```
Read statistics of link: 3

Report of link errors:

Data blocks sent:                26
Data blocks received:             0
Received messages too long:       0
Times DCD lost:                   0
Received messages aborted:        0
Receive overruns:                 0
Transmit underruns:               0
Receive parity errors:            0
Receive framing errors:           0
Receive CRC errors:               0

    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 6
Disable link: 2
Link 2 disabled.

    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 6
Disable link: 3
Link 3 disabled.
```

Figure G–3: Sample Output from Protocol Toolkit Test Showing a BSC Test *(Cont'd)*

```
    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 7
Close link: 2
Link 2 closed.

    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 7
Close link: 3
Link 3 closed.

    Menu
     (0)  Demo
     (1)  Open
     (2)  Configure
     (3)  Enable
     (4)  Send a file
     (5)  Link statistics
     (6)  Disable
     (7)  Close
     (8)  Exit

Select (0-8): 8
Waiting for quiescence
spsalp complete.
```

Figure G–3: Sample Output from Protocol Toolkit Test Showing a BSC Test *(Cont'd)*

Appendix

H STD1200A Loopback Test Procedure

This appendix describes the STD1200A loopback test procedure, including the following:

- an overview of the test
- a description of how to install the hardware needed for the test
- instructions on how to run the test
- a sample screen display from the test

H.1 Overview of the Test Program

The STD1200A loopback test program is placed in the `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin` directory during installation.

Note

Earlier Simpact terminology used the term “synchronous” for blocking I/O and “asynchronous” for non-blocking I/O. Some parameter names reflect the previous terminology.

One high-level test program written in C is supplied with the STD1200A protocol: `s12alp`, which uses non-blocking I/O. The program is interactive; it prompts you for all the information needed to run the test. The test communicates with the ICP through the client data link interface (DLI) commands.

The loopback test performs the following functions:

- Configures the link-level control parameters such as baud rates, clocking, and protocol
- Enables and disables links
- Initiates the transmission and reception of data on the serial lines
- Obtains link statistics from the ICP

You can use the loopback test to verify that the installed devices and cables are functioning correctly. You can also use it as a template for designing client applications that interface with the DLI layer.

H.2 Hardware Setup for the Test Program

The test program runs in loopback mode. Before running the test, perform the following procedure to install the loopback cabling:

Step 1:
Provide a synchronous modem. Configure the modem to supply continuous clocking at a data rate of 2400 bits per second. The ICPs are default configured for external clocking, and the modem supplies the clock signal for loopback testing.

Step 2:
Select a pair of adjacent ports to loopback. Ports are looped back in the following pairs: (0,1), (2,3), (4,5), and so on. Install the special three-headed loopback cable between the ports you selected and the synchronous modem.

Note

The loopback cable is only used during testing, not during normal operation.

H.3 Running the Test Program

Step 1:

Change to the directory that contains the test program: `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin`. Enter the following command at the system prompt:

```
s12alp
```

Step 2:

The following prompts are displayed. Defaults are shown in brackets:

`Need help (H) or internal clocking (I) [N]`

Enter **N** if you are using external clocking and you do not want to view the help information. Enter **H** to view a brief description of the test procedure. Enter I if you are using internal clocking.

`Minutes to run (1-1440) [1]`

Enter the number of minutes you want the test to run.

`ICP board on which to run test (0-5) [0]`

Enter the number of the ICP to be tested. This is the ICP that you cabled for testing in Step 2 on page 130. The number must correspond to the `BoardNo` parameter in the `freeway\client\test\s12\stdaldcfg` file.

`Even port number (0, 2, ..., 14) [0]`

Enter the even-numbered port you cabled for testing in Step 2 on page 130. For example, if you enter 0, the loopback test will be performed on ports 0 and 1.

Step 3:

After you answer the last prompt, the test starts. It displays a series of greater than (>) or less than (<) symbols to indicate that it is running. When it completes, it displays the test results in the form of a brief Statistics Report that shows activity on the two ports being tested. If no errors are shown, your installation is verified.

Step 4:

Remove the loopback cable and configure the cables for normal operation.

H.4 Sample Output from Test Program

Figure H–1 shows the screen display from a sample STD1200A non-blocking loopback test program (`s12alp`). Output displayed by the program is shown in `typewriter type` and your responses are shown in **bold type**. Each entry is followed by a carriage return.

```
% s12alp

Need help (H) or internal clocking (I) [N] ? H

    This program transfers data between a pair of adjacent ports on an ICP
    board.  The first ICP is zero; the first port on an ICP is zero.  The
    program defaults to ICP zero, ports zero and one.

    The ICPs and distribution panels are configured at the factory for
    external clocking.  An adjacent port pair is normally connected with a
    Simpact-supplied THREE-headed loopback cable, with the third head of
    the cable connected to your powered up modem.  Your modem supplies
    clocking to move the data, but the data does not reach the modem.  The
    program does not work with an internal clock source unless "I" is
    entered in response to the above prompt.

    When prompted for values, the range of legal values appears within
    parentheses immediately following the prompt.  The default value then
    appears within square brackets.  To select the default value, simply
    press the RETURN key.  To select a value other than the default, enter
    the desired value followed by the RETURN key.


    Once the program is running, entering one Control C causes the program
    to turn the timer off and go through its normal exit process as if the
    time had expired.  Entering two consecutive Control C's causes the
    program to exit immediately.

Minutes to run (1-1440) [1]? 1
ICP board on which to run test (0-5) [0]? 0
Even port number (0, 2, ..., 14) [0]? 0

STD1200A Asynchronous Port-To-Port Loopback Program.
   Test duration in minutes: 1 minute
   ICP board number: 0
   Ports: 0 & 1

STD1200A Software Version:
FREEWAY 2000/4000 ICP6000 COMMUNICATIONS FRONT END PROCESSOR
SIO-STD-1200A Rev 1 (V5.1-3, VI-100-0160) 22-DEC-97

>>>>>>><<>>>>>>>><<<<>>>><<>>>><<>>><<<<<<<<<<<<>>>>>>>>>>>>>>>>>><<<<>>>><<>>><<<<
<<<<<<><<>>>><<>>>>><<>>>>>>>>>><<<<>>>><<<<<<<<<<<<><<>>>><<>>>><<>>>><<>>>><<<<<<
<<<<<<>>>>>>>>>><<<<<>>>>>>>>>><<<<>>><<<<<<<<<<<<><<>>>>>>>>>>>>>>>>>>>>>>>><<<<<<
```

Figure H–1: Sample Output from STD1200A Non-Blocking Loopback Program

```
STD1200A Statistics Report:
                              server0icp0port0      server0icp0port1
                              ----------------      ----------------
   inv addresses                         0                     0
   inv ctlfields                         0                     0
   rcv CRC errs                          0                     0
   lfrm too long                         0                     0
   rcv overruns                          0                     0
   txmt underruns                        0                     0
   txmt wtchdg                           0                     0
   ITS achieved                          1                     1
   data writes                         591                   584
   data reads                          584                   591
Loopback test complete
```

Figure H–1: Sample Output from STD1200A Non-Blocking Loopback Program *(Cont'd)*

Appendix

I X.25/HDLC Loopback Test Procedure

This appendix describes the X.25/HDLC loopback test procedure, including the following:

- an overview of the tests
- a description of how to install the hardware needed for the tests
- instructions on how to run the tests
- sample screen displays from the tests

I.1 Overview of the Test Programs

The X.25/HDLC loopback test programs are placed in the `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin` directory during installation. These test programs are written in C and communicate with the ICP through the CS API function calls to perform the following functions:

- Establish virtual circuit or data link connections.
- Initiate the transmission and reception of data on the serial lines.
- Terminate virtual circuit or data link connections.

You can use the loopback tests to verify that the installed devices and cables are functioning correctly. You can also use them as templates for designing client applications that interface with the CS API layer.

I.2 Hardware Setup for the Test Programs

The test programs run in loopback mode. Before running any test program, perform the following procedure to install the loopback cabling:

Step 1:
Provide a synchronous modem. Configure the modem to supply continuous clocking at a data rate between 300 and 64,000 bits per second. The ICPs are default configured for external clocking, and the modem supplies the clock signal for loopback testing.

Step 2:
Install the special three-headed loopback cable between ports 0 and 1 on ICP 0 and the synchronous modem. To test other ports or ICPs, you must edit the `.setup` file in the `freeway\client\test\x25mgr` directory, then run the make file.

Note

The loopback cable is only used during testing, not during normal operation.

I.3 Running the Test Programs

Step 1:
Change to the directory that contains the test programs: `freeway\client\[axp_nt_emb` or `int_nt_emb]\bin`.

Before running the X.25 test program, `x25_svc`, or HDLC test program, `hdlc_user`, you must run the `x25_manager` utility to configure the X.25/HDLC software. This utility runs interactively or uses an input setup file to configure the links to test either X.25 or HDLC. Table I–1 shows the appropriate setup file for each test program.

Table I–1: X.25/HDLC Test Files[a]

Program	Description	Setup File for Input to x25_manager
x25_manager	This is the configuration utility program described fully in the *Freeway X.25/HDLC Configuration Guide*. It runs interactively or accepts a setup file as input to the x25_manager file command.	–
hdlc_user	This is the sample test program used to verify the installation and configuration of the HDLC protocol service on the ICP.	fw1000_hdlc.setup
x25_svc	This is the sample test program used to verify the installation and configuration of the X.25 protocol service on the ICP.	fw1000_svc.setup

[a] These files are located in the freeway\client\test\x25mgr directory. The executable files are located in the freeway\client\[axp_nt_emb or int_nt_emb]\bin directory.

Step 2:

Enter the following command at the system prompt. If you omit the optional CS API configuration file name, x25_manager uses the default cs_config file.

```
x25_manager [CS API configuration file name]
```

Step 3:

At the x25_manager prompt, enter the file command with the appropriate setup file:

```
file(fw1000_hdlc.setup)
file(fw1000_svc.setup)
```

The fw1000_hdlc.setup input file instructs the x25_manager program to configure the ICP links for running HDLC. The fw1000_svc.setup input file instructs the x25_manager program to configure the ICP links for running X.25.

Step 4:

To start the test program, enter one of the following command at the system prompt. If you omit the optional CS API configuration file name, `x25_manager` uses the default `cs_config` file.

```
hdlc_user [CS API configuration file name]
x25_svc [CS API configuration file name]
```

Step 5:

The following prompts are displayed:

- Test length in minutes
- Packet data field size; this must not exceed the larger of the two buffer sizes configured in the setup file
- Packet transmit window size. The setup file configures the ICP to support a window size of 1–7. To use a window size greater than 7, you must change the setup file to support packet level modulo 128 operation. See the *Freeway X.25/HDLC Configuration Guide*.
- Link numbers of the links that were looped back in Step 2 on page 136
- User data field value (X.25 only). This may be any value in the given range. However, if you run multiple copies of the `x25_svc` test program, you must specify a different user data field value for each.

After you answer the last prompt, the test starts. The installation is verified if the test completes successfully without errors.

Step 6:

Remove the loopback cable and configure the cables for normal operation.

I.4 Sample Output from Test Programs

Figure I–1 shows the screen display from a sample `hdlc_user` test program. Figure I–2 shows the screen display from a sample `x25_svc` test program. Output displayed by the program is shown in `typewriter type` and your responses are shown in **bold type**. Each entry is followed by a carriage return.

```
% x25_manager

SIMPACT X.25 MANAGER
----------------------------
: file(fw1000_hdlc.setup)
SAPX25{: @(#) Protocol: CCITT/ISO 1984/1988 X.25:SERVICE_XIO:ICP2432:OS-V331
SAPX25{: @(#) Version: VI-100-0135: X25FW2432 3.2-5 Oct 17 1997 16:24:33
SAPX25{BUFFERS[: Configuring buffers.
SAPX25{BUFFERS[: Configured 1080 SMALL buffers 1024 bytes each.
SAPX25{SLP[: Configuring SLP 0.
SAPSLP{SLP[: Configuring SLP 1.


% hdlc_user

SIMPACT HDLC OPTIONS
--------------------

Test Length in Minutes (1 to 1440): 1

HDLC data field size (32 to 1024): 512

HDLC transmit window (1 to 127): 7

Lowest link ID in test (0 to 7): 0

Highest link ID in test (0 to 7): 1

Connecting clients
Transferring data
No further screen interruptions for 1 minute(s)

2 links in test
Packet data size 512 bytes.
Packets/second: XMIT        4   RECV        4   TOTAL        8
Bits/second:    XMIT    16384   RECV    16384   TOTAL    32768
Link ID number         0       1
LCN reset errors       0       0
Transport errors       0       0
RCV data packets     136     136
XMT data packets     143     142

Allowing ICP to settle
Disconnecting
HDLC TEST test terminated
```

Figure I–1: Sample Output: HDLC Loopback Program (hdlc_user)

```
% x25_manager

SIMPACT X.25 MANAGER
----------------------------
: file(fw1000_svc.setup)
SAPX25{: @(#) Protocol: CCITT/ISO 1984/1988 X.25:SERVICE_XIO:ICP2432:OS-V331
SAPX25{: @(#) Version: VI-100-0135: X25FW2432 3.2-5 Oct 17 1997 16:24:33
SAPX25{BUFFERS[: Configuring buffers.
SAPX25{BUFFERS[: Configured 12 BIG buffers 1024 bytes each.
SAPX25{BUFFERS[: Configured 3262 SMALL buffers 128 bytes each.
SAPX25{BUFFERS[: Configured 256 virtual circuit maximum.
SAPX25{SLP[: Configuring SLP 0.
SAPX25{CALLSERVICE[: Configuring CALLSERVICE on SLP 0.
SAPX25{SLP[: Configuring SLP 1.
SAPX25{CALLSERVICE[: Configuring CALLSERVICE on SLP 1.
SAPX25{REQUEST[: Enabling link 0.
SAPX25{REQUEST[: Enabling link 1.

% x25_svc

SIMPACT X.25 SVC OPTIONS
------------------------

Test Length in Minutes (1 to 1440): 1

Packet data field size (32 to 1024): 512

Packet transmit window (1 to 127): 7

Lowest link ID in test (0 to 15): 0

Highest link ID in test (0 to 15): 1

User data field value (0 to 32767): 2

Connecting clients
Transferring data
No further screen interruptions for 1 minute(s)

2 links in test
Packet data size 512 bytes.
Packets/second: XMIT        4   RECV        4   TOTAL        8
Bits/second:    XMIT    16384   RECV    16384   TOTAL    32768
Link ID number         0       1
LCN reset errors       0       0
Transport errors       0       0
RCV data packets     134     134
XMT data packets     139     138

Allowing ICP to settle
Disconnecting clients
X25 SVC TEST test terminated
```

Figure I–2: Sample Output: X.25 Loopback Program (x25_svc)

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