ICP2432 User's Guide for Windows NT® 4.0 and NT® 5.0 (Windows 2000®) (DLITE Interface)

Protogate, Inc.
12225 World Trade Drive, Suite R
San Diego, CA 92128
January 2003

PROTOGATE

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

ICP2432 User's Guide for Windows NT 4.0 and NT 5.0 (Windows 2000) (DLITE Interface)

Printed in the United States of America

Contents

List of Figures

List of Tables

Preface

Purpose of Document

This document describes how to use the ICP2432 intelligent communications processor in a peripheral component interconnect (PCI) bus computer running the Windows NT 4.0 or 5.0 (Windows 2000) operating system.

Intended Audience

This document is intended primarily for Windows NT system managers and applications programmers. Application programmers can use Protogate's data link interface (DLI) embedded module to interface to the ICP2432 device driver. This embedded DLI interface is called DLITE. The interface 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 DLITE interface. 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 environment (for example, the

PCIbus ICP2432) or who are developing an initial DLITE application in the embedded environment.

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

Appendix A describes Protogate'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 is the user's guide for the DLITE Windows NT Logger System Service. This appendix supplements the logging and tracing information in Chapter 3.

Appendix D describes the multithreaded sample programs.

Protogate References

The following documents provide useful supporting information, depending on the customer's particular hardware and software environments. Most documents are available on-line at Protogate's web site, www.protogate.com.

General Product Overviews

- *Freeway 1100 Technical Overview* 25-000-0419
- *Freeway 2000/4000/8800 Technical Overview* 25-000-0374
- *ICP2432 Technical Overview* 25-000-0420
- *ICP6000X Technical Overview* 25-000-0522

Hardware Support

- *Freeway 3400 Hardware Installation Guide* DC 900-2004
- *ICP2432 Hardware Description and Theory of Operation* DC 900-1501
- *ICP2432B Hardware Description and Theory of Operation* DC 900-2006
- *ICP2432 Hardware Installation Guide* DC 900-1502
- *ICP2432B Hardware Installation Guide* DC 900-2009

Freeway Software Installation Support

- *Freeway User's Guide* DC 900-1333
- *Loopback Test Procedures* DC 900-1533

Embedded ICP Installation and Programming Support

- *ICP2432 User's Guide for Digital UNIX* DC 900-1513
- *ICP2432 User's Guide for OpenVMS Alpha* DC 900-1511
- *ICP2432 User's Guide for OpenVMS Alpha (DLITE Interface)* DC 900-1516
- *ICP2432 User's Guide for Windows NT (DLITE Interface)* DC 900-1514

Application Program Interface (API) Programming Support

- *Freeway Data Link Interface Reference Guide* DC 900-1385
- *Freeway Transport Subsystem Interface Reference Guide* DC 900-1386

Socket Interface Programming Support

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

Toolkit Programming Support

- *Freeway Server-Resident Application and Server Toolkit Programmer's Guide* DC 900-1325
- *OS/ProtogateProgrammer's Guide* DC 900-2008
- *Protocol Software Toolkit Programmer's Guide* DC 900-2007

Protocol Support

• *ADCCP NRM Programmer's Guide*	DC 900-1317
• *Asynchronous Wire Service (AWS) Programmer's Guide*	DC 900-1324
• *Addendum: Embedded ICP2432 AWS Programmer's Guide*	DC 900-1557
• *AUTODIN Programmer's Guide*	DC 908-1558
• *BSC Programmer's Guide*	DC 900-1340
• *BSCDEMO User's Guide*	DC 900-1349
• *BSCTRAN Programmer's Guide*	DC 900-1406
• *Military/Government Protocols Programmer's Guide*	DC 900-1602
• *SIO STD-1200A (Rev. 1) Programmer's Guide*	DC 908-1359
• *SIO STD-1300 Programmer's Guide*	DC 908-1559
• *X.25 Call Service API Guide*	DC 900-1392
• *X.25/HDLC Configuration Guide*	DC 900-1345
• *X.25 Low-Level Interface*	DC 900-1307

Document Conventions

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 4.0 and NT 5.0 (Windows 2000) (DLITE Interface)*, Protogate document DC 900-1514E, is recorded below:

Revision	Release Date	Description
DC 900-1514A	October 1998	Original release
DC 900-1514B	November 1998	Minor modifications and clarifications to Chapter 3 Added Appendix D, "Multithreaded Sample Programs"
DC 900-1514C	December 1998	Add `dlControl` alternative (Section 3.3.2.6 on page 41) Minor changes throughout
DC 900-1514D	February 1999	Modify Chapter 2 and Appendix D for Military/Government protocols Add new DLITE errors (Table 3–1 on page 52) Minor changes to Chapter 4
DC 900-1514E	January 2003	Modify the document to reflect the taking over of the document by Protogate, Inc. Modify as required to indicate NT 5.0 (Windows 2000) support as well as the use of InstallShield Express. Modify as required to indicate support of the new ICP2432B as well as the old ICP2432.

Customer Support

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

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

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

Chapter

1 Product Overview

The Protogate 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 Protogate-supplied device driver.

The ICPTool program, supplied with the product, downloads the ICP-resident software to the ICP2432. Protogate'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 Protogate'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.

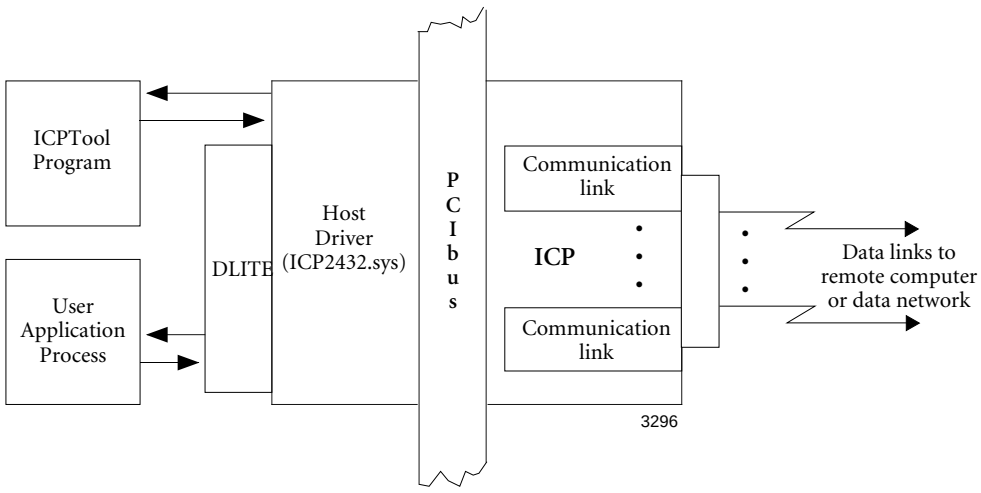

Figure 1–1: Typical Data Communications System Configuration

Chapter

2 Software Installation

This chapter describes Protogate's ICP2432 software installation procedure for Windows NT 4.0 and NT 5.0 (Windows 2000).

2.1 Memory Requirements

Protogate recommends that you have at least 32 megabytes of system memory for the ICP2432 product for NT 4.0 and 64 megabytes for NT 5.0.

2.2 ICP2432 Software Installation Procedure

Step 1: If you are using NT 4.0, you can install one or more ICP2432 boards in your computer, as described in the *ICP2432B Hardware Installation Guide* before loading the software. If you are using NT 5.0 (Windows 2000), it is easier to install the cards afterwards so that the Plug And Play manager will have the required `ICP2432.inf` file.

Step 2: Insert the proper *ICP2432 for Windows NT...* CD-ROM into your Windows NT computer. There are different CDs for NT 4.0 and NT 5.0.

Step 3: If you don't have "auto start" enabled, start the installation by opening the `index.html` program on the installation CD-ROM. Click the line `"Embedded ICP Software ..."` on the "home page" of the CD. On the new page that has been linked to, click the line `"InstallShield for Windows NT - Intel"`. Then the startup information, shown in Figure 2–1, is displayed. It is recommend that the installer run from its current location (i.e. the CD).

Note

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

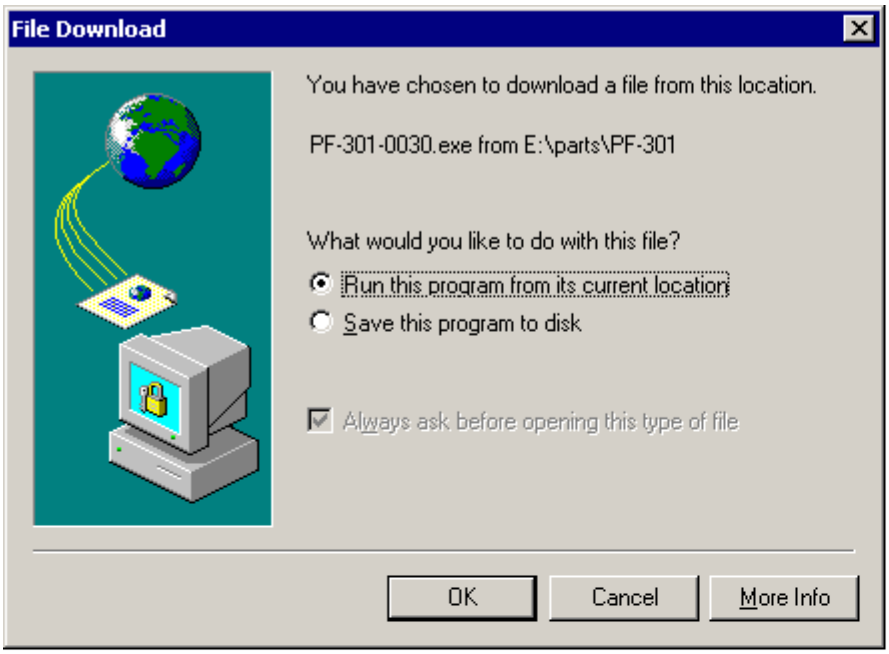

Figure 2–1: Startup Information for Embedded ICP2432

Step 4: Click "Yes" when the "Authenticode..." window is displayed and click "Next" when the "Welcome to..." window is displayed.

Step 5: The next window defines the 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 "`Change...`".

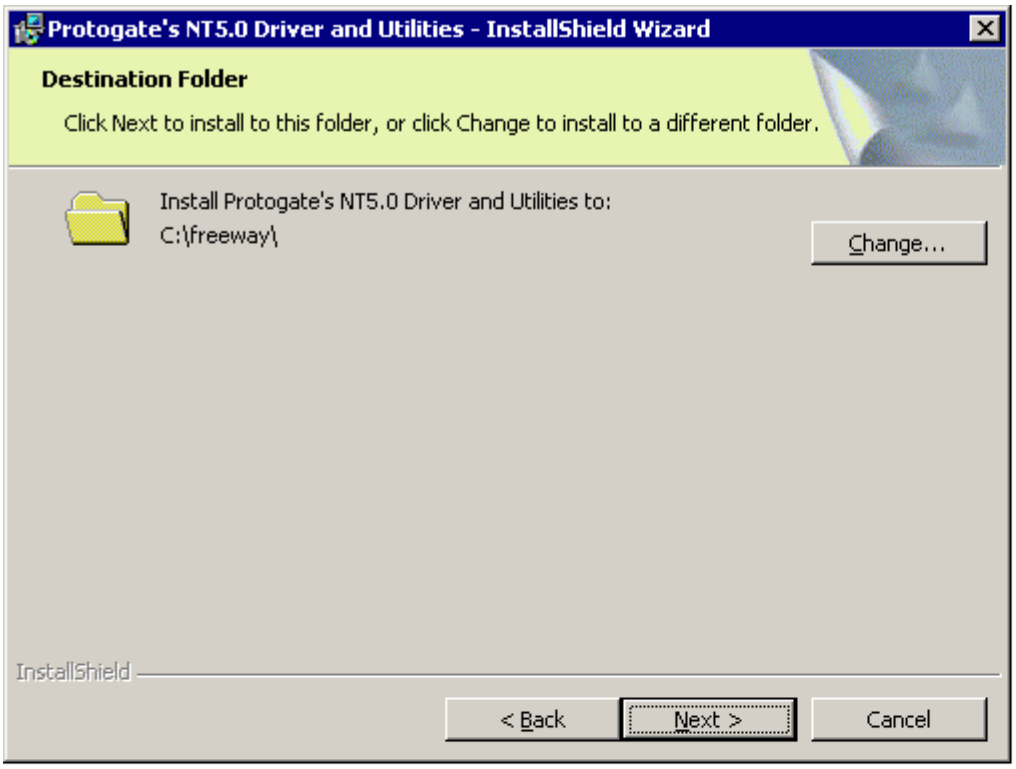

Figure 2–2: Installation Directory for Embedded ICP2432

Step 6: Select the desired installation directory and click "ok" (Figure 2–3).

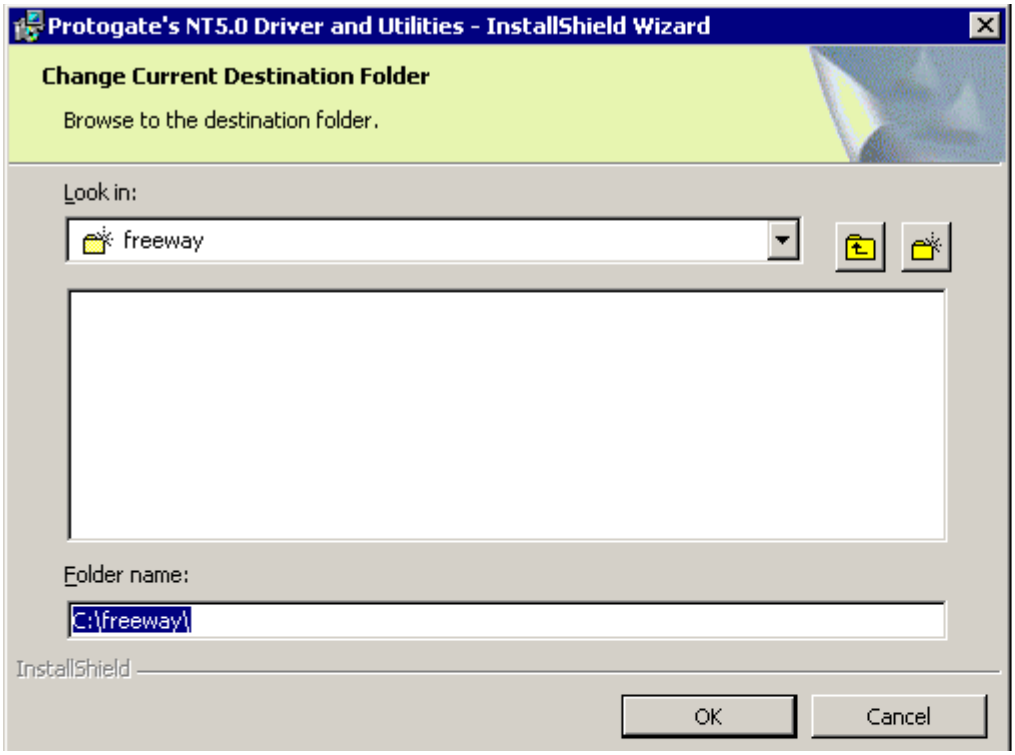

Figure 2–3: Program Folder

Step 7: After completion of Step 6, the installation script updates and inserts keys into the system registry.

Step 8: When the installation is complete, a "...Completed" window is displayed, click "Finish".

Step 9: The Restart Windows menu (Figure 2–4) provides two options, to restart your computer now or later. If you are running NT 4.0, click "Yes". If you are running NT 5.0 (Windows 2000), click "No" and "Shutdown" your system via the "Start -> Shutdown..." menu so that you can install the ICP2432s. After you have booted up NT 5.0

and all of the ICPs have been found, reboot the system so that the ICP number will be as expected. ICP numbering under NT 4.0 starts with Bus 0 and goes up, while NT 5.0 starts with the highest bus number and works down.

Note

Remove the installation CD before restarting your computer. Also note that the CD has the freeway directory on its root directory. In the `freeway\lib\emb\nt\tools` are the sources and build environment required for ICPTool and the loaders.

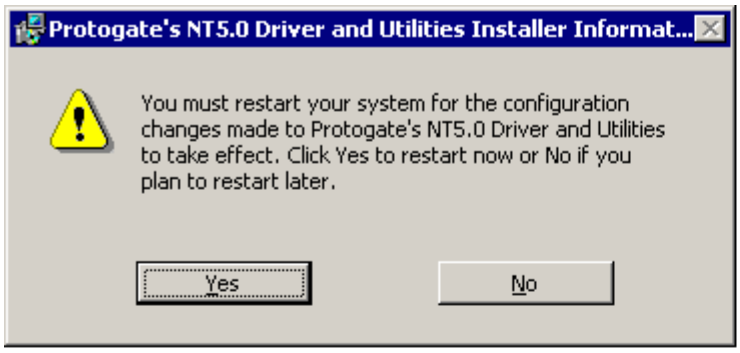

Figure 2–4: Restart Windows

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]
DDCMP	ddcmp[b]
FMP	fmp
Military/Government	mil[c]
Protocol Toolkit	sps
STD1200A	s12
X.25/HDLC	x25[d]

[a] Except for the readme, release notes, release history, and load configuration files where *ppp* is bsc. For example, bscload is used for BSC3270 and BSC2780/3780.

[b] Except for the readme, release notes, and release history configuration files where *ppp* is ddc.

[c] Some Military/Government files use the identifier "mg*n*" where *n* is a Protogate-supplied product designator.

[d] 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 is in the `freeway\boot` directory.

The executable object for the system-services module for protocol software other than protocol toolkit (`xio_2432.mem`) is in the `freeway\boot` 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\nt_dlite\`*ppp*[1] 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.

1. The Military/Government protocols use the `freeway\client\test\mil` directory.

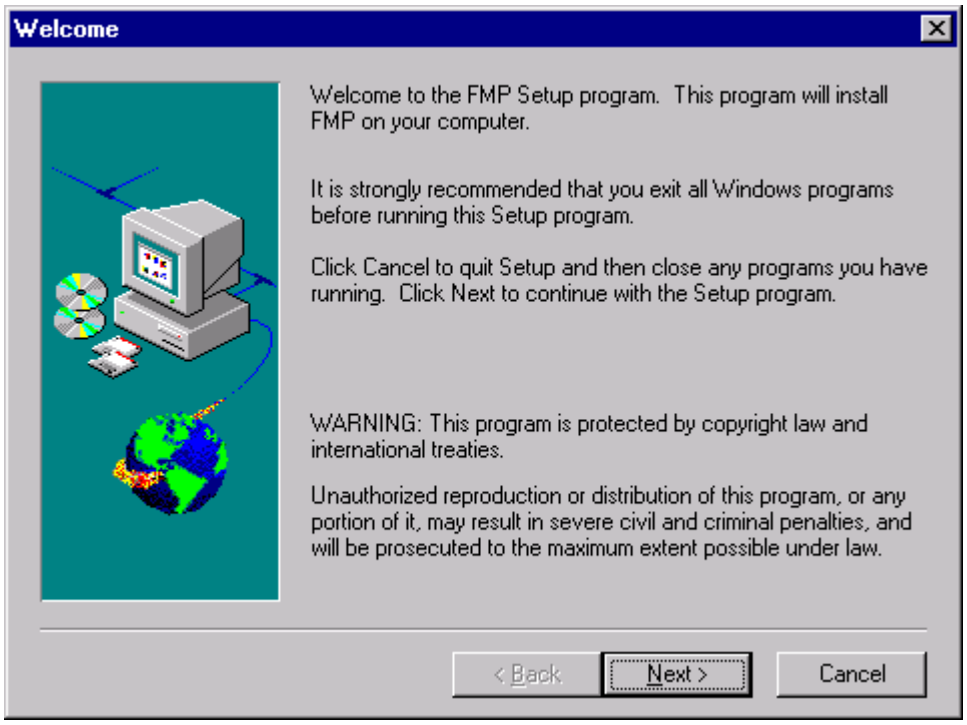

Figure 2–5: Startup Information for FMP

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`.

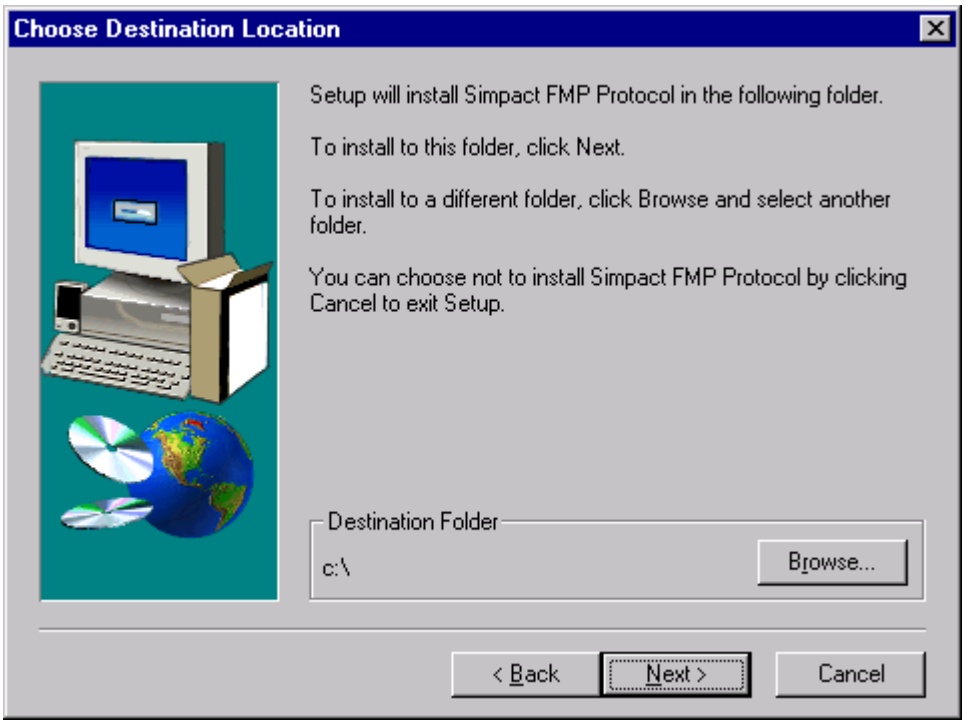

Figure 2–6: Installation Directory for FMP

Step 4: Using any text editor, edit the load file (`freeway\boot\`*`ppp`*`load`) for your protocol. Uncomment the lines associated with ICP2432. Do not change the memory locations (such as `40120000`) 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` or `axp_nt_emb]\bin` directory.

```
nmake -f makefile.ent
```

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` or `axp_nt_emb]\lib` to your path.

or

Copy the `.dll` files from `\freeway\client\[int_nt_emb` or `axp_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\nt_dlite\`*`ppp`*[2] directory, enter the following command:

```
nmake
```

2. The Military/Government protocols use the `freeway\client\test\mil` directory.

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

Step 6: Select "Start → Programs → Protogate ICP2432 → Protogate ICPTool" (or just double click on the Protogate 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: Protogate ICPTool Icon

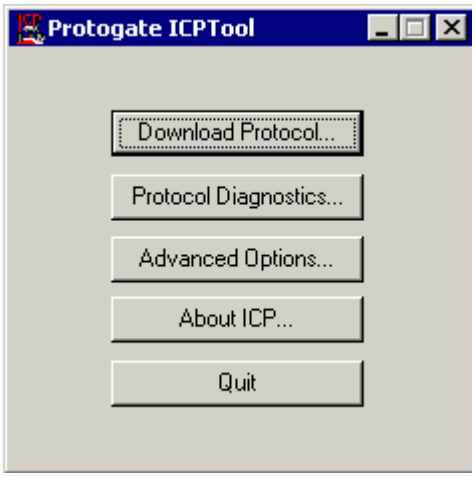

Figure 2–8: ICPTool Main Menu

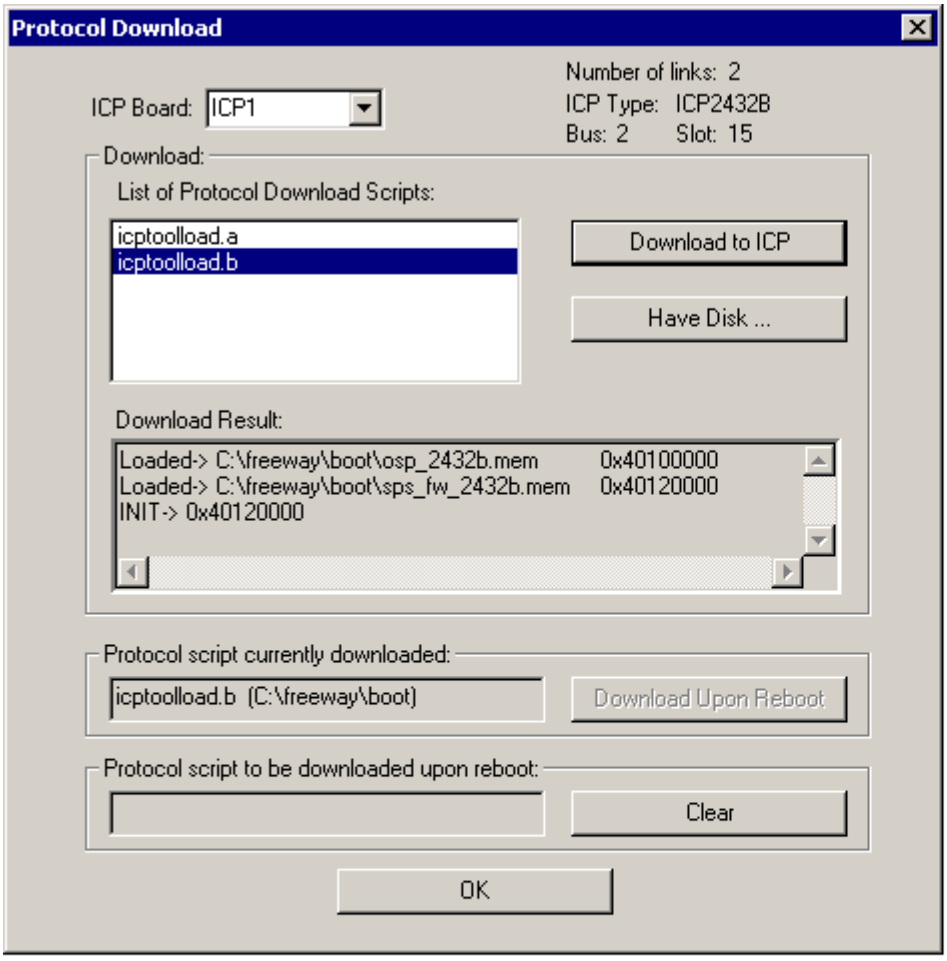

Figure 2–9: Protocol Download Menu

Step 7: Select the protocol you wish to download in the `List of Protocol Download Scripts`, then select `Download to ICP`. Note that the ICP type, ICP2432A or ICP2432B, the ICP's Bus number, and slot number are displayed in the upper left hand corner of the window.

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` or `axp_nt_emb]\bin` directory. Run the loopback test as described in Appendix D.

Chapter

3 Programming Using the DLITE Embedded Interface

3.1 Overview

This chapter primarily describes the differences between the data link interface (DLI) to Freeway (as described in the *Freeway Data Link Interface Reference Guide*) and the DLITE embedded interface in a Windows NT system, referred to as "DLITE." Changes to the scope and nature of Freeway DLI support are described.

This chapter should be read by application programmers who are doing one of the following:

- Porting an existing application (currently operational in the Freeway environment) to the embedded environment (for example, the embedded ICP2432 PCIbus board).
- Developing an initial DLITE application in the embedded environment. You should first read the *Freeway Data Link Interface Reference Guide* and have it available as your primary reference.

In addition to the *Freeway Data Link Interface Reference Guide*, the following Protogate reference documents are of interest to application programmers:

- *Freeway Client-Server Interface Control Document* (for writing to the socket level)
- The applicable protocol-specific programmer's guide for your application.

DLITE is a new, streamlined interface designed specifically for the embedded interface to the ICP2432 board. The interface provides new capabilities while retaining the

majority of the "Freeway DLI" (henceforth referred to as DLI) capabilities. By using DLITE, developers can concentrate on the communication requirements of the ICP2432 rather than the details required by the Win32 interface and the ICP2432 NT driver, thereby reducing programming complexity and development time. DLITE can be thought of as a communications pipe to the ICP2432. It is compatible with the existing Freeway DLI (with caveats described in Section 3.3.2 on page 39). DLITE provides a high-level open/close/read/write interface to the ICPs. It supports both blocking and non-blocking I/O. The DLITE interface is thread-safe and supports multiple threads requesting its services.

3.2 Embedded Interface Description

3.2.1 Comparison of Freeway Server and Embedded Interfaces

The traditional DLI and TSI interface supports client applications communicating with the Freeway server on a local-area network (LAN). This type of interface is shown in Figure 3–1. In an embedded environment, the application does not access a network in communicating with the ICP.

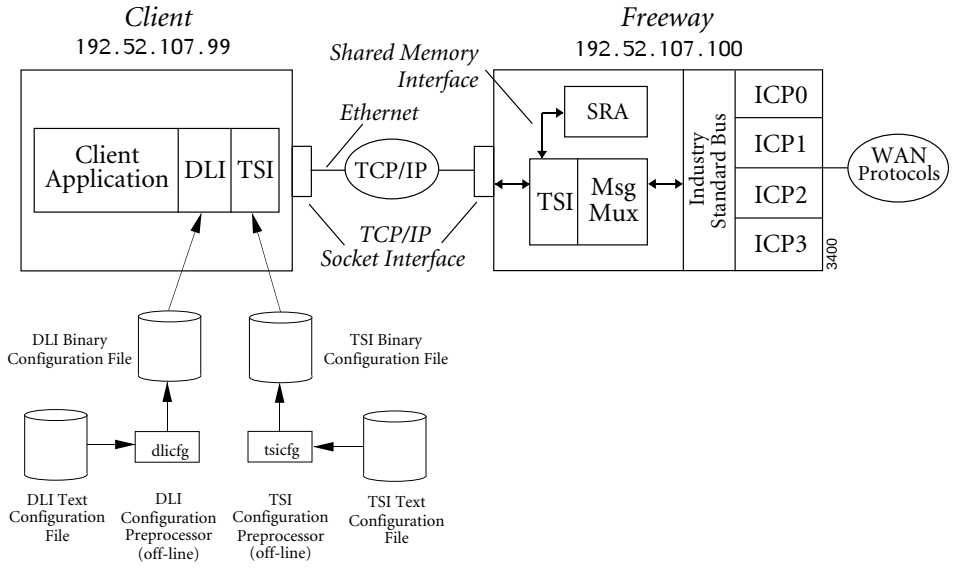

Figure 3–1: DLI/TSI Interface in the Freeway Server Environment

Instead, the embedded application using DLITE communicates directly with the Windows NT ICP2432 driver (through the Win32 interface), which accesses the locally attached ICP. This interface is shown in Figure 3–2. In this environment no Freeway-type communications take place; it is designed specifically for the embedded system.

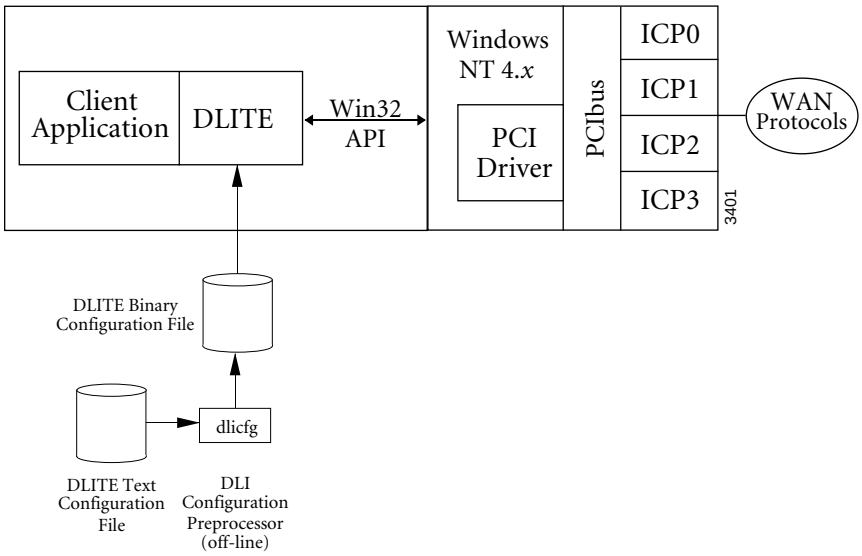

Figure 3–2: DLITE Interface in an Embedded ICP2432 Environment

3.2.2 Embedded Interface Objectives

The DLITE interface was designed as a streamlined interface to the ICP2432 supporting a multithreaded application. It supports only *Raw* operation protocols, which means that the application is responsible for all communications with the ICP.

DLITE was designed to maximize portability between existing applications. The objective was an interface that would require "no changes" when porting from a Freeway environment to an embedded environment. While this objective has been met (for *Raw* operation), there are differences between these environments, as well as differences in system behavior. These differences are addressed in the following sections.

3.3 DLITE Interface

The DLITE interface is described here in terms of enhanced capabilities, limitations and caveats, the API itself, configuration files, and logging/tracing (see Section 3.3.1 through Section 3.3.5). Within each context, necessary changes and any behavior differences are noted.

3.3.1 DLITE Enhancements

3.3.1.1 Multithread Support

DLITE supports a multithread application interface which is thread-safe for both blocking I/O and non-blocking I/O. Sample multithread programs are provided, as described in Appendix D.

Caution

Users are not protected from the misuse of threads.

Multithread support is accomplished by serializing access to shared processing and eliminating or otherwise guaranteeing integrity of global data.

Access is serialized to the following services so that only a single thread can be in the service at any one time:

- `dlInit`
- `dlOpen`
- `dlClose`
- `dlTerm`

The following functions allow application threads concurrent access to the degree specified:

- `dlRead` — read requests block if another read for the same session is currently being serviced
- `dlWrite` — write requests block if another write for the same session is currently being serviced
- `dlBufAlloc` — multiple thread concurrent access
- `dlBufFree` — multiple thread concurrent access
- `dlPoll` — request dependent
    - Read complete — blocks at session level
    - Write complete — blocks at session level
    - Read cancel — blocks at session level
    - Write cancel — blocks at session level
    - Session status — multiple thread concurrent access
    - System configuration — multiple thread concurrent access
    - Driver information — multiple thread concurrent access
    - Trace control — multiple thread concurrent access

3.3.2 DLITE Limitations and Caveats

3.3.2.1 *Raw* Operation Only

DLITE supports only *Raw* operation. As with DLI, *Raw* operation means that the API sends nothing to the ICPs except that which is provided by the application for transmission; therefore, the client application must handle all the following:

- Configuration of the ICP/Protocol
- ICP and protocol control data (using the DLI `OptArgs` structure accompanying each `dlRead` and `dlWrite` request)
- I/O details of the specific protocol

Raw operation especially impacts configuration of the ICP. Whereas *Normal* operation performs ICP configuration for the application using information from the DLI configuration file, the application using *Raw* operation is totally responsible for configuration. The DLI configuration file does not support "protocol" parameters (in fact, their presence results in errors during configuration file processing because they are not allowed in *Raw* operation).

3.3.2.2 No LocalAck Processing Support

Local acknowledgment (`LocalAck`) processing is not supported. When data is written to an ICP, the user receives an acknowledgment that the ICP did in fact receive that data (refer to your protocol-specific programmer's guide for details). The Freeway DLI does support a "LocalAck" capability that hides this from the application programmer (previous writes are not posted as complete until DLI receives this LocalAck, then the LocalAck is thrown away). However, the DLITE user is responsible for receiving each LocalAck and performing any necessary processing. The DLITE behavior is exactly the same as when the DLI `LocalAck` configuration parameter is set to "no". This generally implies the client application should post a `dlRead` after each `dlWrite` to receive the expected Local Ack.

3.3.2.3 AlwaysQIO Support

DLI optionally supported an "AlwaysQIO" feature (applicable only when using non-blocking I/O), which restricted notification of completed I/O to callback invocations only. If an I/O completed immediately in the I/O request, the completion would not be reported with the return of the `dlRead` or `dlWrite` request. Instead, notification would be through the user-supplied callback.

DLITE always behaves as if the `AlwaysQIO` configuration parameter is set to "yes" (non-blocking I/O only). Non-blocking I/O should always return with `EWOULDBLOCK` while the I/O completes (via Win32 Overlapped I/O).

3.3.2.4 Changes in Global Variable Support

DLI maintained three global variables; `dlerrno`, `ilCPStatus`, and `cfgerrno`. The global variables `ilCPStatus` and `cfgerrno` are not supported for DLITE. The `ilCPStatus` value simply returned the value contained in the ICP status field, which is now available to the DLITE application in the `ilCPStatus` field from the `OptArgs`. The information in `cfgerrno` is no longer available.

The `dlerrno` variable is still available, but has been redefined for DLITE as a function call returning an integer (`int _dlerrno()`). Reference to `dlerrno` becomes a function call which returns the last error for the thread making the call. Note that this definition precludes using `dlerrno` as an "L-value" in a "C" expression.

3.3.2.5 dlInit Function No Longer Implied

DLI allowed users to perform `dlOpen` before calling `dlInit` (`dlInit` would be invoked if required, not a recommended practice). This results in an error when using DLITE. Processing must be initialized using `dlInit` before any other service is requested.

3.3.2.6 Unsupported Functions

The following functions are not supported. Applications invoking these functions return with the `DLI_XX…XX_ERR_NEVER_INIT` error.

- `dlControl` (see note below and Figure 3–3)
- `dlListen`
- `dlPost`
- `dlSyncSelect`

DLITE does not support the dynamic building of the DLI configuration file if the .bin does not currently exist. This means that DLITE expects the binary configuration file to exist at run time in order to function properly.

Note

Any previous application which used `dlControl` to perform a programmatic download to the ICP must use an alternate method. The Figure 3–3 code fragment illustrates the `DownloadICP()` function. The application must link with the `icpdnld.dll` and `icpdnld.lib` libraries, which are found in the `bin` and `lib` directories of `c:\freeway\client\[int_nt_emb` or `axp_nt_emb]`, respectively.

3.3.3 The Application Program's Interface to DLITE

Except where described in the previous sections, the embedded DLITE interface does not change the application's interface to DLI. While the DLI interface has remained intact, changes have been made in both the methods supporting DLI and in the underlying functionality.

```
#include "c:\\freeway\\include\\icpdnld.h"

DWORD result;                                   /* return code 0=success */
char  buff[80];                                 /* ICP Error Msg Buffer */

result =  DownloadICP(
     "//./icp1",                                /* ICP Device Name */
     "c:\\freeway\\boot\\bscload.txt",   /* Full Path of Download Script File */
     buff,                                      /* Error Msg Buffer */
     80,                                        /* Error Msg Buffer Size */
     NULL,                                      /* Report Buffer */
     0);                                        /* Report Buffer Size */

if ( result )
     printf( "\nDownloadICP Error =%d %s", result, buff );
```

Figure 3–3: Code Fragment Example to Download ICP

3.3.3.1 Building a DLITE Application

The DLITE API library for NT (Intel) is `dliteint.lib` and the associated DLL is `dliteint.dll`. For the Alpha processor, the names are `dliteant.lib` and `dliteant.dll` respectively. The user must include the preprocessor definitions "WINNT" and "DLITE" (e.g., `/D "WINNT"` and `/D "DLITE"`) when building the application using the Protogate-supplied libraries and include header files.

3.3.3.2 Blocking and Non-blocking I/O

Implementation of non-blocking I/O has changed in some of the services. In summary, the following functions use blocking I/O, regardless of the session's definition of the `asyncIO` parameter in the DLI configuration file. These functions do not return to the application until all processing is completed for the service requested:

- `dlInit`
- `dlOpen`
- `dlClose`
- `dlTerm`

- `dlPoll`
- `dlBufAlloc`
- `dlBufFree`

The following functions use non-blocking I/O when requested by the application (that is, when the `asyncIO` configuration parameter is set to "yes"). They return to the application immediately after the operation is queued.

- `dlRead`
- `dlWrite`

Using non-blocking I/O, a successful operation returns `OK`, and `dlerrno` has the value of `EWOULDBLOCK`. The application is notified of I/O completion through the I/O completion handler (IOCH). The completed I/O operation is retrieved using a `dlPoll` request for read/write complete. See Section 3.3.3.5 on page 50 for more information on callbacks and I/O completion.

Using blocking I/O, the `dlRead` and `dlWrite` functions return `ERROR` if unsuccessful; otherwise, they return the number of bytes transferred (not including the ICP and Protocol Header inserted by DLITE).

3.3.3.3 Changes in DLI/TSI

The lack of a network connection has eliminated the need for some of the client/server communications between the current DLI and TSI. While the user buffer is not affected, some data previously in the DLI header (i.e. the Freeway header) and the TSI header is no longer built by the API. These changes are transparent to the user but may be noted when examining DLITE trace files.

3.3.3.4 Changes in DLI Functions

No changes are required in the user interface to DLI. Some DLI functions have changed in their implementation, which might affect the user's expected behavior of the function. Changes in the 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 presented to DLITE (using `dlBufAlloc`) invoke NT system memory services to allocate buffers (using `malloc` calls). Do not assume any type of buffer initialization. Also, the size requested in `dlBufAlloc` can be thought of as the size requested from the system (the actual size is somewhat larger, which includes some DLITE overhead requirements). If the application requests one byte for the data buffer size, it should assume only one byte is returned.

User requests are verified against the `MaxBufs` and `MaxBufSize` DLITE configuration parameters. Requests exceeding either of these return a buffer allocation error.

Buffers allocated using `dlBufAlloc` are allocated with room for the ICP and Protocol header, and a small DLITE work area prefacing the user's data area. This area is added to the user's request; users do not have to account for these requirements in their buffer request. DLITE also "tags" each buffer, and verifies the buffer was allocated using `dlBufAlloc` before it frees the buffer in `dlBufFree`. Users can not free a buffer they allocated directly from the system using `dlBufFree`. Buffer alignment requirements for communications with the NT driver are performed by `dlBufAlloc`. The buffer returned is correctly aligned.

Note

The user's buffer allocation request should be only for the user's data; the space required for the ICP and Protocol headers are "silently" added to the buffer request by `dlBufAlloc`. If the application is not using the DLITE buffer allocation service, it must account for the following:

- Sixteen (16) bytes for the protocol header immediately prefacing the data buffer
- Sixteen (16) bytes for the ICP header immediately prefacing the protocol header
- Alignment of the buffer address on the correct boundary

dlBufFree

This service 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 and attempts to free a buffer not allocated with `dlBufAlloc` return with a buffer deallocation error message.

dlClose

A close request (`dlClose`) for a specific session blocks until all other threads have exited that same session's close (`dlClose`), read (`dlRead`), and write (`dlWrite`) request. This might cause the close thread to block on a blocking I/O request (only for the same session) which is blocked and waiting on its timeout. Users can circumvent this problem by assuring all I/O is cancelled prior to the close request. Close processing waits for all the closed session's threads to complete before returning to the application.

dlInit

The user application must call `dlInit` before any other DLITE service. If `dlInit` does not find the DLI configuration file, it returns the `DLI_INIT_ERR_CFG_LOAD_FAILED` error. It does not try to find a DLI source configuration file and perform the configuration processing in-line. The logging and tracing capabilities can fail initialization (e.g. the log_server is not installed) without inhibiting DLITE from providing all its other services. However, Protogate strongly discourages the operation of DLITE without the log facility.

dlOpen

A session open (`dlOpen`) initiates communications with the NT driver. In both blocking and non-blocking I/O, `dlOpen` returns with the result of the operation: a session ID if successful, an error otherwise. A successful open of a non-blocking operation returns a `dlerrno` of `EWOULDBLOCK` and generates a callback. This callback could be delivered before the API returns from the open request and would contain the correct session ID. This callback can be ignored, since the application can use the completion of the open request to control the open operation.

dlPoll

A new poll request of `DLI_POLL_GET_DRV_INFO` returns NT driver information. The information shown in Figure 3–4 is returned through the `pStat` parameter provided by the application (the application provides a pointer to an allocated area of type `DLI_ICP_DRV_INFO`). The area used to return this information must have been allocated by the requesting application.

Note

The `DLI_POLL_TRACE_STORE` and `DLI_POLL_TRACE_WRITE` poll requests are not supported by DLITE.

```
typedef struct          _DLI_ICP_DRV_INFO
{
    unsigned long       Node;               /* Node assigned */
    unsigned long       DeviceNumber;       /* Device Number (ICP) */
    unsigned long       NumberOfPorts;      /* Number of ports on ICP */
    unsigned long       BufferAlignment;    /* Byte alignment requirement */
    unsigned long       NumberOfIcps;       /* Number of ICPs installed */
    unsigned char       Version[DLI_MAX_STRING + 1];
                                            /* Driver version string. */
}                       DLI_ICP_DRV_INFO;
typedef DLI_ICP_DRV_INFO    *PDLI_ICP_DRV_INFO;
#define DLI_ICP_DRV_INFO_SIZE   sizeof(DLI_ICP_DRV_INFO)
```

Figure 3–4: `DLI_ICP_DRV_INFO` "C" Structure

Cancel Processing using `dlPoll` (`DLI_POLL_READ_CANCEL` and `DLI_POLL_WRITE_CANCEL`) is performed differently. The change should be transparent to existing applications. New applications can optionally take advantage of this change.

- A request to cancel reads or writes (`dlPoll` request cancel read/write) cancels all outstanding reads or writes for the session at the time the request is received. In the Freeway DLI, these were cancelled individually, with the buffer pointer and `OptArgs` pointer returned for each request.

- Cancelled I/O is considered as completed. If a user has five read requests queued and performs a read cancel, a poll would show five reads completed.

- Cancelled I/O is returned as previously; each request is returned (with buffer pointer and `OptArgs` pointer) with each poll requesting the cancel, until all are returned. Returning the cancelled request reduces the number of I/O completions by one.

- Because cancelled I/O is considered completed, cancelled requests are also returned in response to requests for completed reads and writes (using `dlPoll`). These requests are returned with the `DLI_IO_ERR_IO_CANCELLED` error code.

- This implementation of cancel processing supports those applications designed for the Freeway DLI.
- The user application should ignore the buffer length and associated buffer data when a cancelled I/O request is returned.

dlRead

There is no change to the `dlRead` function. However, because DLITE supports *Raw* operation only, it does require an associated `OptArgs` with each I/O request. DLITE fills in the supplied `OptArgs` structure with the appropriate data from the ICP and Protocol headers associated with the read data received from the ICP. Read requests (`dlRead`) are returned to the application with the supplied `OptArgs` structure built from the ICP and Protocol header received with the data buffer. All the ICP and protocol information is available in the `OptArgs` structure when the read buffer is returned.

Non-blocking I/O should expect an `EWOULDBLOCK` error upon return. A callback is issued when the read is completed. A callback is invoked for each (both read and write) read completion.

If the read operation is returned with an error, the data in the `OptArgs` structure is not valid. The application must verify the read operation before referencing `OptArgs` data.

Note

As with the DLI interface, read requests with a NULL buffer pointer result in DLITE allocating and returning a read buffer. The address of the buffer allocated is returned in the supplied buffer pointer upon return from the call. This is true for both blocking and non-blocking I/O. The user that wants a DLITE allocated buffer should ensure the buffer pointer supplied with the `dlRead` call is NULL.

dlTerm

Termination processing (`dlTerm`) releases resources and terminates DLITE. Any active I/O active is cancelled when `dlTerm` is called. Data buffers associated with the cancelled I/O are deallocated if those buffers were allocated by DLITE (using `dlBufAlloc`). `OptArgs` buffers are not deallocated. The application should cancel all I/O before terminating.

The `dlTerm` function sleeps for 1–2 seconds (not including any time required in the cancelling of active I/O) to allow threads which might have been active previous to the termination request to exit.

dlWrite

As with `dlRead`, `dlWrite` requires an associated `OptArgs` structure with the write request. DLITE builds the ICP and Protocol headers, which preface every application buffer (see `dlBufAlloc`), from information supplied in this `OptArgs` structure. Specifically, DLITE does the following for *Raw* operation writes:

1. `ICP->usClientID = htons (OptArgs->usICPClientID);`
2. `ICP->usServerID = htons (OptArgs->usICPServerID);`
3. `ICP->usCommand = htons (OptArgs->usICPCommand);`
4. `ICP->usParms[0-2] = htons (OptArgs->usICPParms[0-2]);`
5. DLITE adds `ICP->iStatus = LittleEndian ? htons (0x4000) : htons (0);`
6. DLITE adds `ICP->usDataBytes = htons (BufLen + DLI_PROT_HDR_SIZE);`
7. If the ICP command is an Attach, or a Write Expedite, the node ID (previously retrieved from the NT driver) is stored in `ICP->usParam[0] ( ICP->usParms[0] = htons( Session->drvNodeID ) )`.

```
8. PROT->usCommand = OptArgs->usProtCommand;

9. PROT->iModifier = OptArgs->iProtModifier;

10. PROT->usLinkID = OptArgs->usProtLinkID;

11. PROT->usCircuitID = OptArgs->usProtCircuitID;

12. PROT->usSessionID = OptArgs->usProtSessionID;

13. PROT->usSequence = OptArgs->usProtSequence;

14. PROT->usXParms[0-1] = OptArgs-> usProtXParms [0-1]);
```

Non-blocking I/O should expect an `EWOULDBLOCK` error upon return. A callback is issued when the write is completed. A callback is invoked for each (both read and write) write completion.

3.3.3.5 Callbacks

Callbacks occur only in those sessions configured for non-blocking I/O. They represent the completion of an I/O activity; signaling the application to perform actions dependent on that I/O completion. In the DLITE interface, this operation might be a `dlPoll` to retrieve session status to ascertain the session's I/O state, or to request read/write completes (using `dlPoll`). Blocking I/O applications receive their I/O upon return from the `dlRead` or `dlWrite` function.

Callbacks are issued in the context of their own thread. Only one callback thread exists in each DLITE process. Callbacks are delivered sequentially; they are never reentered by another callback.

Caution

As the callback operates in the context of its own thread, the application must protect itself with data referenced by its callback processing and processing of other, concurrent, threads.

There is no difference between the "main" callback and the "session" callback. They are initiated sequentially by DLITE. For sake of efficiency, Protogate recommends the user make use of only one.

To maintain conformity with the existing DLI, callbacks are delivered upon completion of `dlOpen` processing. Although `dlOpen` processing does not generate a callback from the system (i.e., an I/O completion port thread is not "kicked-off") the API does, just prior to exiting the `dlOpen` processing, emulate the event by placing a "callback" request in an internal callback queue for delivery to the application.

In a similar manner, callbacks on `dlClose` requests are generated and delivered by the API.

The callback thread runs at a higher priority. This ensures that callbacks do not backup on the delivery queue. This backup would occur when the application processes more than one I/O completion event in the callback routine (e.g., processing more than one read/write compete in a single invocation of the application callback routine). At a higher priority, the application callback processing can process as many (or as few) as design dictates without regard to a queue backup.

3.3.3.6 DLITE Error Codes

The error codes listed in Table 3–1 have been added to DLITE.

Table 3–1: DLITE Error Codes

Value	DLITE Error Code	Description and Recommended Action
–10211	DLI_OPEN_ERR_ICP_INVALID_STATUS	Returned by dlOpen(). The ICP has not been downloaded with a protocol or is in a non-operational state.
–10231	DLI_OPEN_ERR_NO_DRV_INFO	An error occurred in the I/O interface while requesting NT driver information. Terminate the interface, verify NT driver installation.
–10232	DLI_OPEN_ERR_NO CMPLT_PORT	An error occurred while requesting an I/O completion port from the system. Terminate and try re-establishing the application.
–10518	DLI_READ_ERR_NO_OPTARG	The application failed to provide an OptArgs structure with the read request. Modify the application to build and supply an OptArgs structure with each read request.
–10721	DLI_POLL_ERR_INVALID_STATE	A request for driver information was made for a session not currently open. Open the session before requesting NT driver information.
–10902	DLI_BUFA_ERR_SIZE_EXCEEDED	An attempt was made to allocate more buffers, or a buffer of greater size, than that defined in the DLI configuration file. Modify the application to adhere to sizes defined in the DLI configuration file.
–11003	DLI_BUFF_ERR_NONE_ALLOC	An attempt was made to deallocate a buffer when none were allocated. Modify application to account for used buffers.
–11004	DLI_BUFF_ERR_ALREADY_FREE	Returned by dlBufFree(). The buffer specified has already been released.
–11918	DLI_WRIT_ERR_NO_OPTARG	The application failed to provide an OptArgs structure with the write request. Modify the application to build and supply an OptArgs structure with each write request.
–12003	DLI_IO_ERR_IO_CANCELLED	The read or write request was cancelled at the request of the user application.

All NT system errors are mapped into existing DLI error codes (dlerrno) so the application can recognize the error condition and react accordingly. All NT errors are returned from calls to GetLastError() when a NT service fails. NT errors are mapped to dlerrno as described in Table 3–2.

Table 3–2: NT Errors Mapped to dlerrno

NT Value	NT Error Code	Applicable dlerrno Codes
1 8 87 998 1117	ERROR_INVALID_FUNCTION ERROR_NOT_ENOUGH_MEMORY ERROR_INVALID_PARAMETER ERROR_NOACCESS ERROR_IO_DEVICE	DLI_POLL_ERR_IO_FATAL DLI_READ_ERR_INTERNAL_DLI_ERROR DLI_WRIT_ERR_INTERNAL_DLI_ERROR
5	ERROR_ACCESS_DENIED	DLI_READ_ERR_UNBIND DLI_WRIT_ERR_UNBIND
22	ERROR_BAD_COMMAND	DLI_READ_ERR_IO_FATAL DLI_WRIT_ERR_IO_FATAL DLI_POLL_ERR_IO_FATAL
170 121	ERROR_BUSY ERROR_SEM_TIMEOUT	DLI_READ_ERR_TIMEOUT DLI_WRIT_ERR_TIMEOUT DLI_POLL_ERR_READ_TIMEOUT DLI_POLL_ERR_WRITE_TIMEOUT
234	ERROR_MORE_DATA	DLI_READ_ERR_OVERFLOW DLI_POLL_ERR_OVERFLOW
995	ERROR_OPERATION_ABORTED	DLI_READ_ERR_INTERNAL_DLI_ERROR DLI_WRIT_ERR_INTERNAL_DLI_ERROR DLI_POLL_ERR_IO_FATAL
1784	ERROR_INVALID_USER_BUFFER	DLI_READ_ERR_INVALID_BUF DLI_WRIT_ERR_INVALID_BUF DLI_POLL_ERR_INVALID_REQ_TYPE

3.3.4 Configuration Files

DLITE uses only the DLI configuration files (TSI configuration files are not used and are not required). The DLI configuration file must specify "`protocol = raw`" in the session sections. With this specification, no parameters are allowed in the protocol section.

The DLI configuration file has been changed to include parameters previously specified in the TSI configuration file (which is no longer used). These parameters are required to maintain conformity with those applications porting from DLI to DLITE. This file has been changed as follows:

`MaxBuffers` — This parameter has been added to the "main" section. It replaces the `MaxBuffers` parameter previously defined in the TSI configuration file. This value is returned in the `usMaxBufs` field of the configuration parameters returned in response to a `dlPoll` for system configuration. Operationally, this value limits the number of buffers the user can have outstanding using the `dlBufAlloc` function. If not explicitly defined in the DLI configuration file, the `MaxBuffers` parameter defaults to 1024.

`MaxBufSize` — This parameter has been added to the "main" section. It replaces the `MaxBufSize` parameter previously defined in the TSI configuration file. This value is returned in the `iMaxBufSize` field of the configuration parameters returned in response to a `dlPoll` for system configuration. Operationally, this value represents the greatest size an application can request using `dlRead`, and defines the buffer size used when a `dlRead` request is made without specifying a buffer (the API allocates and returns this buffer to the application). If not explicitly defined in the DLI configuration file, the `MaxBufSize` parameter defaults to 1024.

`MaxBufSize` — This parameter has been defined in the "session" section of the DLI configuration file. It replaces the `MaxBufSize` parameter previously defined in the TSI configuration file ("connection" section). This value is returned in the `usMaxSessBufSize` field of the session parameters returned in response to a `dlPoll` for session status. Operationally, this value represents the greatest size an applica-

tion can request to be written using `dlWrite`. If not explicitly defined in the DLI configuration file, the `MaxBufSize` parameter defaults to 1024.

`TSICfgName` — The meaning of the TSI configuration file name (no longer required) defined in the DLI "main" section is now used to define the location of the log/trace service. A value of "." (single period within quotes) signifies the current client machine. Also see Section 3.3.5.1 on page 56. Both the log file name and trace file names are modified by appending the current process ID to the supplied name.

3.3.5 Logging and Tracing

The DLITE logging and tracing is very different from that supported in the Freeway environment. The Freeway created and formatted trace and log files internally, whereas DLITE uses a pipe to send packets to a Windows NT Logger System Service. The service is a utility, `LOG_SRV`, which logs events such as errors and trace records to disk files. The service communicates with client applications through a well-known named pipe. Named pipes allow applications to be distributed among several NT systems on the same LAN.

There is no longer any need to "decode" the DLI trace file. Both trace and log data are immediately available for viewing, even when the application is generating the data. Data is sent to this service in one direction, the API does not know the status of the operation. The service can be installed in any client available on the network to the application machine.

The logger service application, `LOG_SRV`, and its configuration file, `LS_CFG`, reside in the directory:

`freeway\client\[int_nt_emb` or `axp_nt_emb]\bin`

The user's application can also make use of the service to write custom log files. For more information, see Appendix C.

3.3.5.1 Logger Service Parameters in the DLI Configuration File

Since DLITE no longer requires a TSI Configuration File, the `TSICfgName` parameter has been redefined to define the "server name" where the Logger System Service is installed. The name is used to construct the pipe name to the service. In most cases, the name "." (with quotation marks) is used to specify "this" server. However, the dot in the pipe name may be replaced with the network name of the server containing the system service, e.g.

`TSICfgName` = "`MyServer`"

Using the name allows the application to be placed on a different NT system on the same LAN and still communicate with the service.

`TSICfgName` — Specifies the server name where the Logger System Service is installed. (The TSI Configuration File is not applicable with DLITE.) Also see Section 3.3.4 on page 54.

Caution

If the user neglects to define the `TSICfgName` with an appropriate server name, an NT System Error 53 (`ERROR_BAD_NETPATH`) occurs notifying the user the network path was not found.

`LogName` — A fully qualified path and file name of the file to store the DLI logging information.

`TraceName` — A fully qualified path and file name of the file to store the DLI trace information.

Note

If the user neglects to define a fully qualified path, files will be deposited into the "`c:\winnt\system32`" directory.

3.3.5.2 Common Logging Service Errors

An application can encounter several errors related to logging and tracing upon initialization with the `dlInit` function. See Table 3–3 and Table 3–4. These errors can result from the unavailability of the Windows NT Logger System Service, either because the service has not been installed or has not been started. In either case, the errors are non-fatal and the application proceeds normally; however, logging and tracing are not activated. The application can ignore these errors (since these services are not available).

Note

The Windows NT Logger System Service records severe errors to the Windows NT system event log. Messages can be viewed using the Windows Event Viewer program.

Table 3–3: DLI Error Codes

Error Code	Error Description	Recommended Action
–10006	`DLI_INIT_ERR_LOG_INIT_FAILED`	`dlLogInit()` failed to start logging. Non-fatal return from `dlInit`. Application can ignore this error (since this service is not available).
–11701	`DLI_LOGI_ERR_TRACE_OPEN_FAILED`	`dlTrcInit()` failed to start tracing. Non-fatal return from `dlInit`. Application can ignore this error (since this service is not available).

Table 3–4: Windows NT Error Codes

NT Error Code	Error Description	Recommended Action
2	ERROR_FILE_NOT_FOUND	Logging Service not installed or started. Install LOG_SRV application.
53	ERROR_BAD_NETPATH	DLI parameter TSICfgName specifies an invalid Logging Server Name. Specify either "." or the Server Name. Rebuild the DLI bin file using DLICFG.EXE.
109	ERROR_PIPE_BROKEN	The Logging Service has terminated abnormally. This can occur if the application attempts to open more pipes than specified in the LS_CFG file.
231	ERROR_PIPE_BUSY	The Logging Service is unable to process the request. Increase max_buffers in the LS_CFG file.
233	ERROR_PIPE_NOT_CONNECTED	The Logging Service is unable to establish a pipe connection to the specified trace/log file.

3.3.5.3 General Application Error File

DLITE creates an application error file "_DLITERR.TXT" which contains descriptive runtime errors. Regardless of log and trace levels defined in the DLITE configuration file, the error file is created in the directory where the application is started. It is a circular file containing a maximum of 1000 entries.

Chapter

4 Programming Using the Win32 Interface

Protogate'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. Protogate'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 DLITE 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. Protogate strongly encourages users to implement their applications using the DLITE interface; however, users who wish to bypass Protogate's API layers and use the Win32 system services directly may do so, although many services provided by the DLITE 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 Protogate'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. Due to Windows NT requirements, device numbers begin at one instead of zero.

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 Protogate 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 Protogate for backward compatibility with device drivers prior to version 1.1-0. Protogate 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; // also Bus #, Slot #, Mem-Map
   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) and Bus #, Slot # and the Memory Mapped flag for ICP2432As. ICP2432Bs are always Memory Mapped
`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 the driver has 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 Protogate protocols recognize expedited write requests, and will 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 Protogate on the distribution media should be used to initialize an ICP.

Note

Customers who have a genuine need to dynamically reinitialize an ICP from within their application should contact Protogate Customer Support as described on page 15 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

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.

Log–>System from the menu bar. Figure 4–3 shows a sample event log displayed in the Event Viewer.

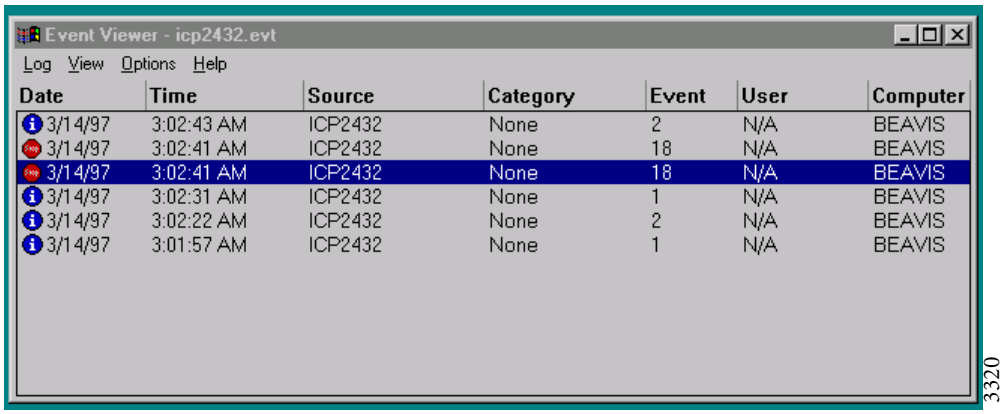

Figure 4–3: Sample Event Log Displayed in the Event Viewer

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 Protogate internal use only) is displayed in the "Data" field of the Event Detail (beginning at offset 0028).

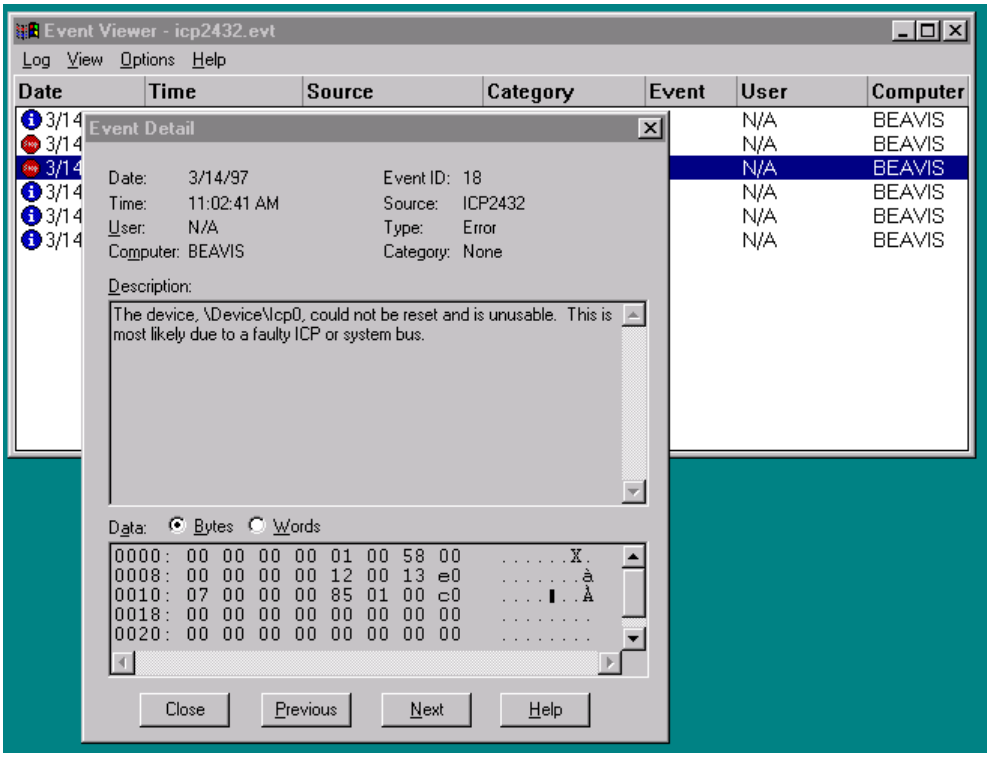

Figure 4–4: Log Message Event Detail

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 Protogate-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 Protogate-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 Protogate-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 Protogate-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 Protogate-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 Protogate-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 Protogate-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 Protogate-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 Protogate-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 the driver within the allotted time. This usually implies that the board has crashed.

Appendix

A ICPTool for Windows NT

This appendix describes the features of the Protogate 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 → Protogate ICP2432 → Protogate ICPTool" (or just double click on the Protogate ICPTool icon shown in Figure A–1).

Figure A–1: Protogate 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

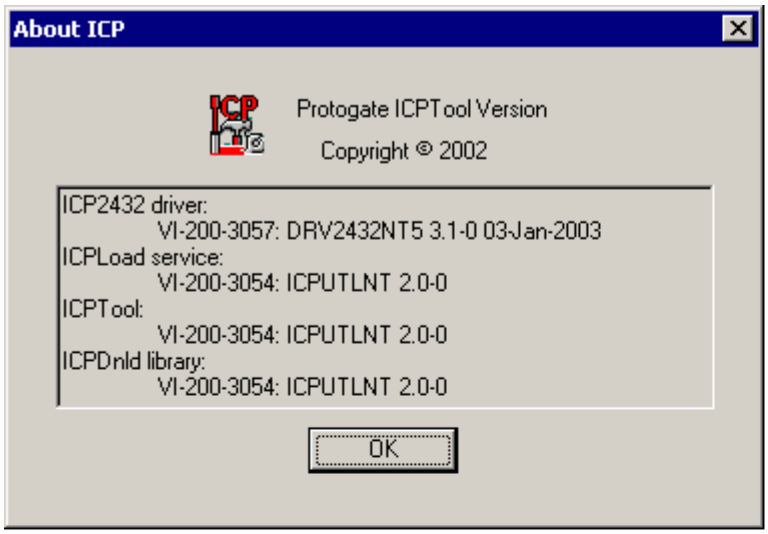

Figure A–3: ICP Information

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.

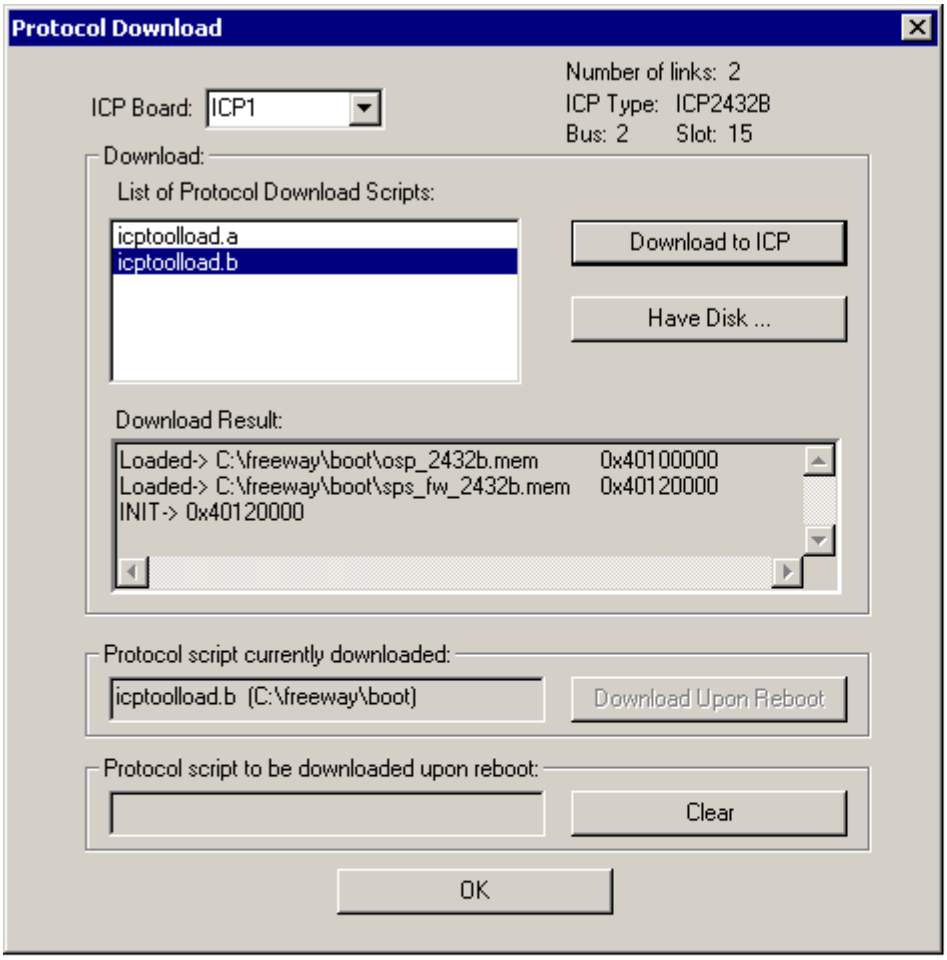

Figure A–4: Protocol Download Menu

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 88 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.

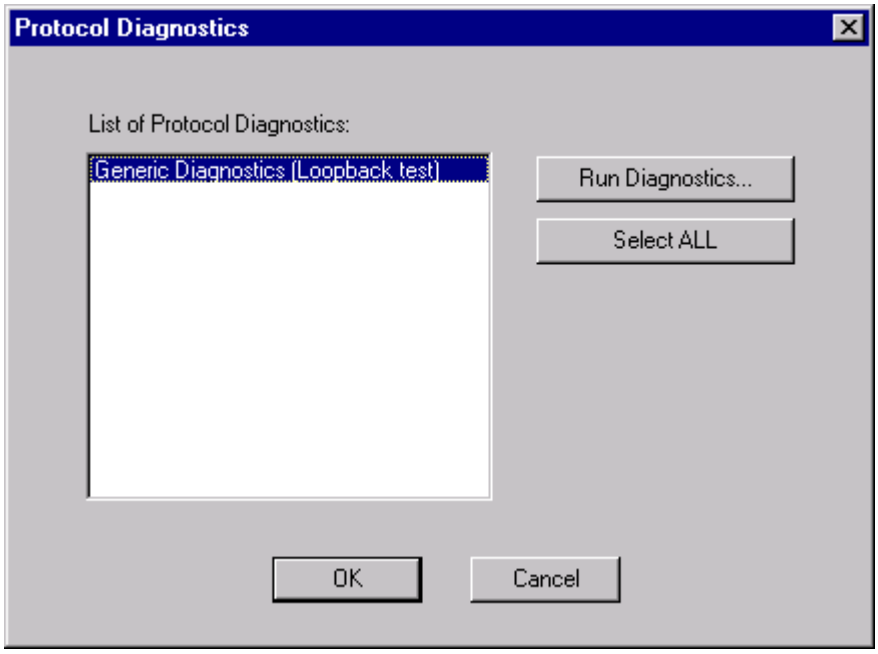

Figure A–6: Protocol Diagnostics Menu

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 87), 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

You can select `Run Default` to verify the ICP hardware and software installation.

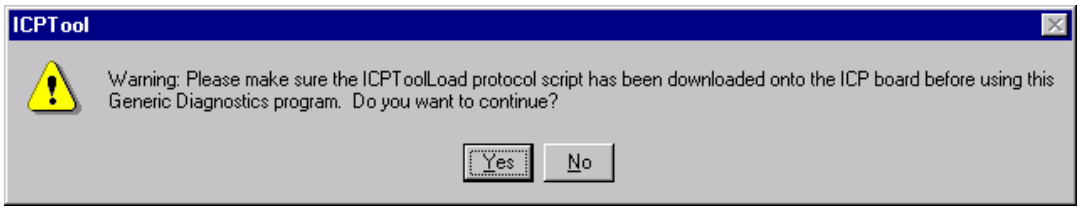

Figure A–7: Generic Diagnostic Warning

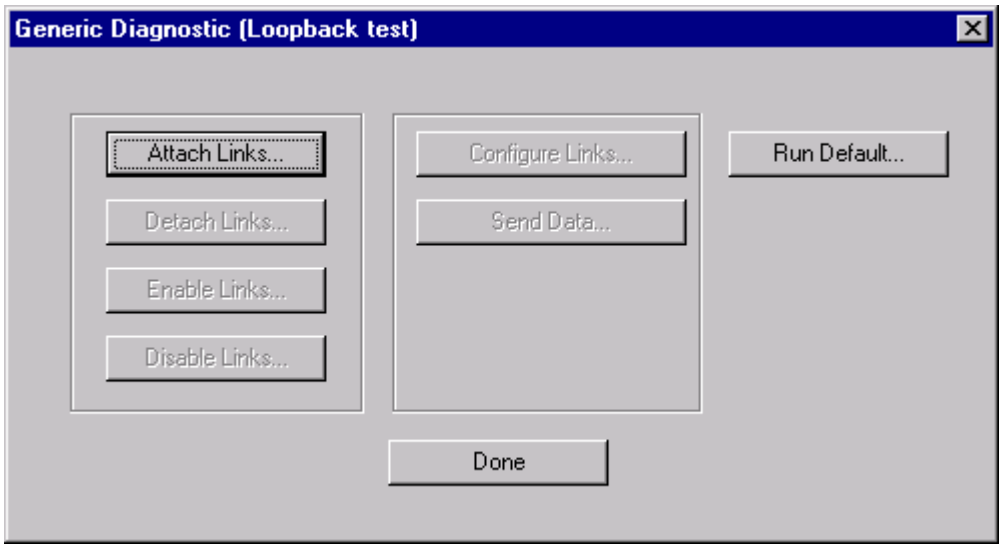

Figure A–8: Generic Diagnostic Main Menu

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

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 89).

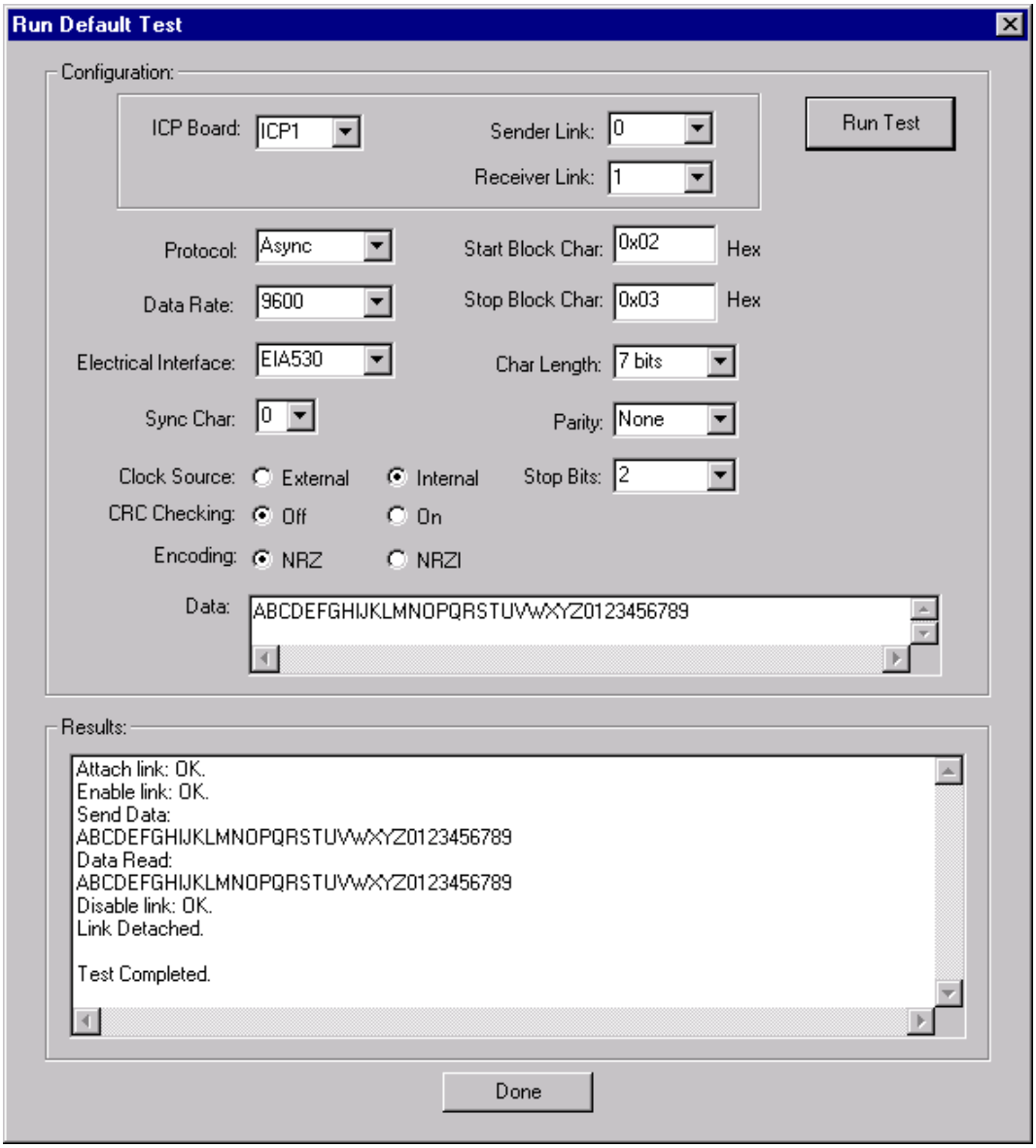

Figure A–9: Default Configuration Menu

A.1.2.4 Attach Link Menu

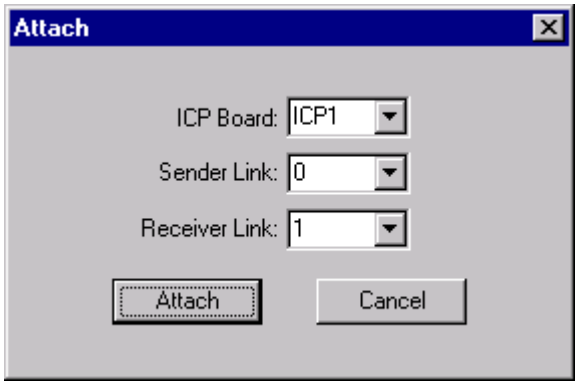

Figure A–10: Attach Link Menu

A.1.2.5 Configure Link Menu

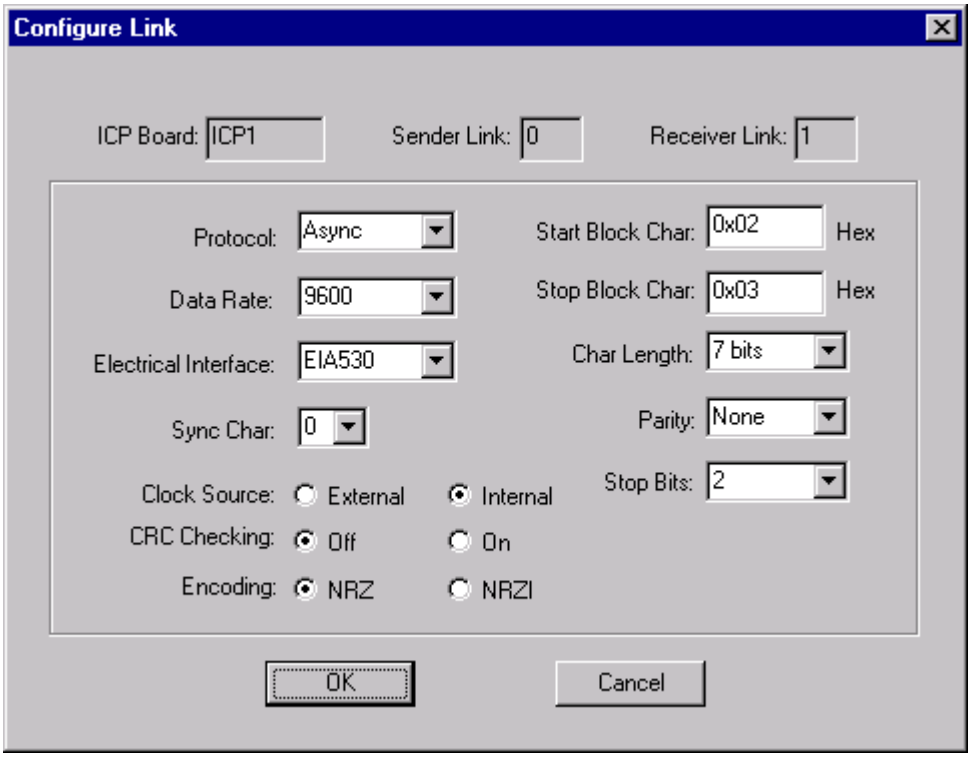

Figure A–11: Configure Link Menu

A.1.2.6 Enable Link Menu

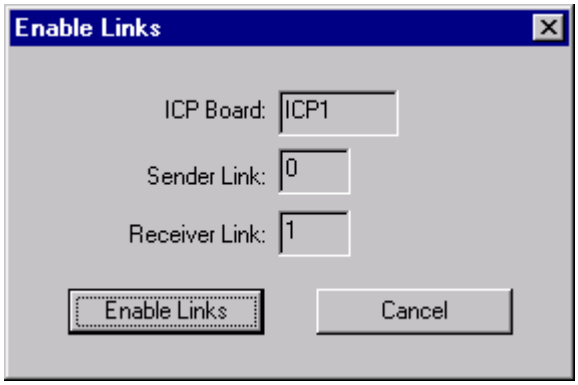

Figure A–12: Enable Link Menu

A.1.2.7 Send Data Menu

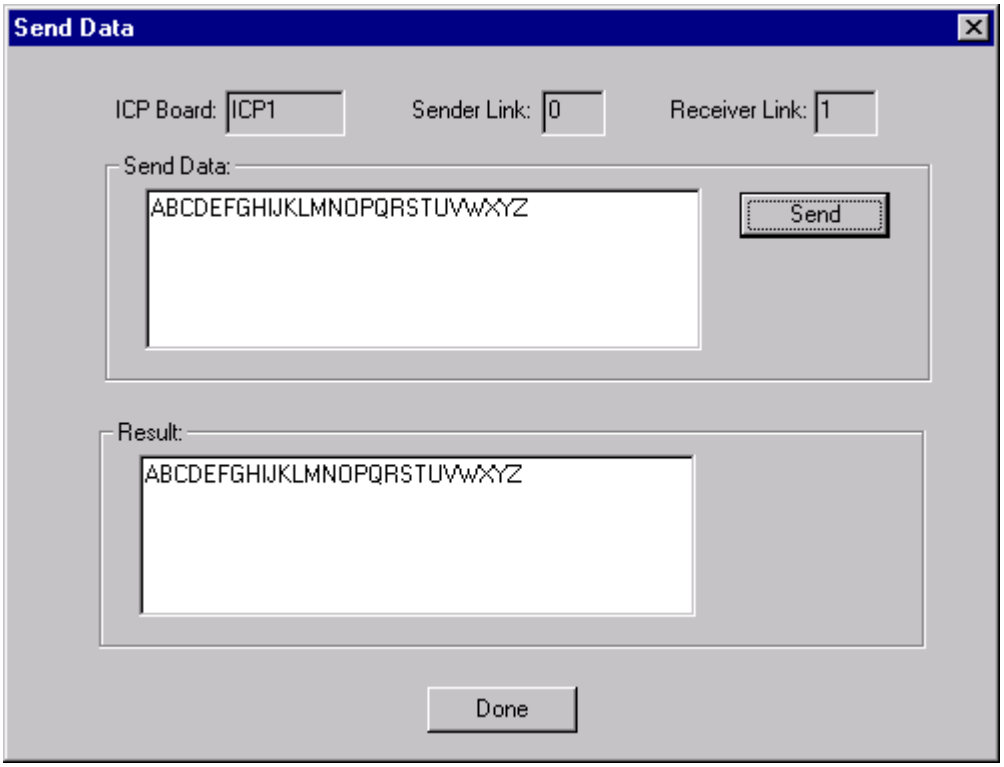

Figure A–13: Send Data Menu

A.1.2.8 Disable Link Menu

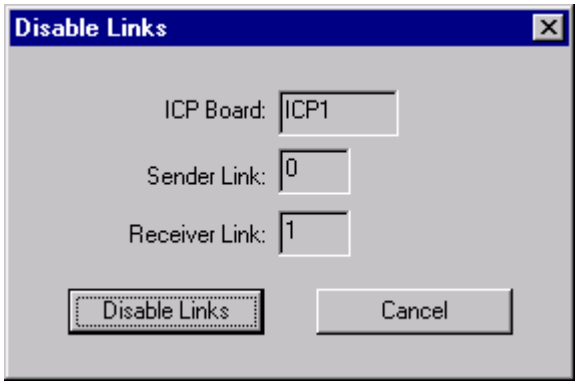

Figure A–14: Disable Link Menu

A.1.2.9 Detach Link Menu

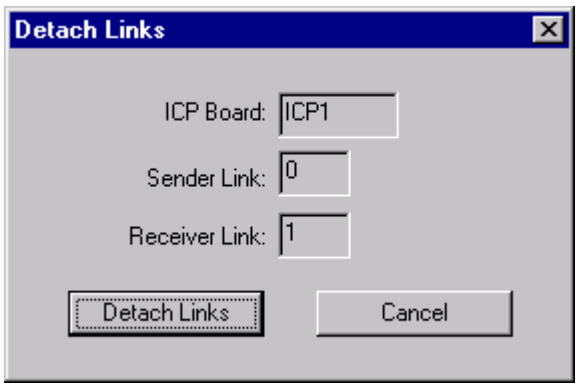

Figure A–15: Detach Link Menu

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

Appendix

B Debug Support for ICP-resident Software

Protogate'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, Protogate ships two versions of the driver. During product installation, a copy of each version is placed in the `C:\freeway\client\int_nt_emb\bin` directory (for Intel) or the `C:\freeway\client\axp_nt_emb\bin` 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\bin` or `axp_nt_emb\bin` directory into `%SystemRoot\System32\Drivers`.

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

5. Reboot the host machine.

ICP-resident software may now be debugged without worry. One thing needs to be noted, however. When the watchdog timers are disabled, 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)

Appendix C

DLITE Logger Windows NT System Service User's Guide

C.1 Introduction

The Windows NT Logger System Service is a software module, which logs events such as errors and DLITE formatted records to disk files. The service communicates with client applications through a well-known named pipe. Named pipes allow applications to be distributed among several NT systems on the same LAN.

There is a one-to-one relationship between a pipe and a particular logging file. A single pipe instance can not have more than one file open at any one time.

The service supports two types of logging files: circular and unlimited-length files. Circular files have a maximum number of records. When a circular file reaches its maximum number of records, it starts writing over records at the top of the file, which means that if all of the records are not of the same size, the current record might only partially overwrite an existing record. In the context of this paragraph, a record is one record from the client application, in other words, an entire DLITE formatted record is considered one record regardless of the number of lines actually written to the file.

Unlimited-length files support records of varying lengths, and the file size is limited only by the amount of available disk space. This type of file should be used with caution since it could fill a disk.

C.2 Starting the Service

The Logger System Service can be installed as an NT system service or run as a console application. To start the service type the following:

`log_srv install` — to install as an NT system service

`log_srv uninstall` — to uninstall the system service

`log_srv console` — to run as a console application

After the logger has been installed as an NT system service, it can be started automatically or manually. To start, stop or set the service to run automatically at system boot-up, go to the Services icon in the Control Panel and select the desired option for the `log_srv` service.

C.3 Configuring the Service

The logger service can be configured by the user supplying a configuration file "`ls_cfg`". The logger uses the WIN32 function `SearchPath()` to find the configuration file in the current directory or in the system path. Figure C–1 is an example configuration file. For information on DLITE logging, see Section 3.3.5 on page 55.

```
#
# Logger System Service configuration
#
#     default values:
#        max_pipe_connections =   25  --> # application client connections
#        max_data_size        = 1024  --> max SERVICE_BUF data size (see log_srv.h)
#        max_buffers          =  250  --> max number of communication buffers
#
#
max_pipe_connections     3
max_data_size          256
max_buffers             15
```

Figure C–1: Example Logger Configuration File

C.4 Connecting to the Service

Client applications connect to the service by calling the `CreateFile` NT library function, as shown in Figure C–2.

```
Code Example Segment
        ...

HANDLE PipeHandle;

        PipeHandle = CreatFile(
              "\\\\.\\pipe\\log_srv", // address of name of the pipe
              GENERIC_WRITE,          // access, write mode
              0,                      // share mode
              NULL,                   // address of security descriptor
              OPEN_EXISTING,          // how to create
              0,                      // file attributes
              NULL);
```

Figure C–2: `CreateFile` Code Example Segment

The dot in the pipe name can be replaced with the network name of the server running the logger system service, e.g.

"\\\\MyServer\\pipe\\log_srv"

Using the name allows the application to be placed on a different NT system on the same LAN and still communicate with the service.

Note

A portion of the pipe name, "`\\pipe\\log_srv`", is fixed and cannot be changed. Only the server name portion, "`MyServer`", can be changed to define the location of the logging service.

C.5 Packet Exchanges

The following is a list of all of the possible packet exchange commands from an application to the Logger System Service. There are no packet exchanges from the logger service to a client application.

`OPEN_FILE` — this packet is used to open a file for logging

`CLOSE_FILE` — this packet is used to close an opened file and its pipe

`WRITE_FILE` — this packet is used for a direct write to an opened file

C.6 Client Structures

The `service_buf` structure shown in Figure C–3 can be used by an application for passing packets to the logger service.

```
typedef struct service_buf
{
        int comand; /* command to perform (OPEN_FILE…FORMAT_FILE) */
        union
        {
              struct open_file open_file; /* structure for the OPEN_FILE command */
              char buffer[1];         /* FORMAT_FILE/WRITE_FILE command */
        } data;
}SERVICE_BUF;

struct open_file
{
        int file_size     /* maximum file size in records for circular files */
        char filename[1]; /* fully qualified file name MUST be null terminated/
};
```

Figure C–3: Structure `service_buf` "C" Definition

Note

1. The `file_size` field is set to zero (0) for an unlimited-length file. The number of records is the number of `WRITE_FILE` packets. A `WRITE_FILE` packet is equivalent to a line or record in the file.

2. The [1] size for char strings is a place holder only

C.7 Packet Examples

OPEN_FILE

This command is sent to the service to open a file associated with the pipe. The file name must be a fully qualified file name; this means it contains the drive and path.

An example code segment (error checking left out for simplicity) is shown in Figure C–4.

```
SERVICE_BUF *buf;
char *fname = "c:\freeway\logs\error.log";

buf = (SERVICE_BUF *)malloc(sizeof(SERVICE_BUF) + strlen(fname));
buf->command = OPEN_FILE;
buf->file_size = 20000;
strncpy(buf->data.open_file.filename, fname, strlen(fname)+1);
WriteFile(PipeHandle, (void )buf, sizeof(SERVICE_BUF) + strlen(fname),
      &bytes_written, &overlapped);
```

Figure C–4: `OPEN_FILE` Code Example Segment

Note

The value `sizeof(SERVICE_BUF)` contains the extra byte for the filename NULL.

CLOSE_FILE

This command is sent to the service to close the file associated with the pipe. It also closes the pipe. The application should still call CloseHandle.

An example code segment (error checking left out for simplicity) is shown in Figure C–5.

```
SERVICE_BUF *buf;
buf = (SERVICE_BUF *)malloc(sizeof(SERVICE_BUF));
buf->command = CLOSE_FILE;
WriteFile(PipeHandle, (void )buf, sizeof(SERVICE_BUF), &bytes_written,
      &overlapped);
CloseHandle(PipeHandle);
```

Figure C–5: CLOSE_FILE Code Example Segment

WRITE_FILE

This command is sent to the service to write a record to the file associated with the pipe.

An example code segment (error checking left out for simplicity) is shown in Figure C–6.

```
SERVICE_BUF *buf;
char *message = "Hello World";
int length;
length = sizeof(int) + strlen(message) + 1;
buf = (SERVICE_BUF *)malloc(length);
buf->command = WRITE_FILE;
strcpy(buf->data.buffer, message);

WriteFile(PipeHandle, (void *)buf, length, &bytes_written, &overlapped);
```

Figure C–6: WRITE_FILE Code Example Segment

Appendix D

Multithreaded Sample Programs

This appendix describes the multithreaded sample programs for Windows NT, including the following:

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

Table D–1 shows the sample program file names for each protocol.

Table D–1: Sample Program File Names

Protocol	Blocking Program	Non-blocking Program
ADCCP NRM	`nrmsync.exe`	`nrmasync.exe`
AWS	`awssync.exe`	`awsasync.exe`
BSC 3270	`327sync.exe`	`327async.exe`
BSC 2780/3780	`378sync.exe`	`378async.exe`
DDCMP	`ddcsync.exe`	`ddcasync.exe`
FMP	`fmpsync.exe`	`fmpasync.exe`
Military/Government	Refer to the *Military/Government Protocols Programmer's Guide*	
Protocol Toolkit	`spssync.exe`	`spsasync.exe`
STD1200A	`s12sync.exe`	`s12async.exe`

D.1 Overview of the Test Program

The multithreaded sample programs are placed in the `freeway\client\[int_nt_emb` or `axp_nt_emb]\bin` directory during the installation procedures.

Note

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

Two high-level test programs (shown in Table D–1) written in C are supplied with each protocol. The programs are interactive; they prompt you for all the information needed to run the test. The test communicates with the ICP through the embedded DLITE interface (described in Chapter 3).

The multithreaded sample programs perform the following functions:

- Configure the link-level control parameters such as baud rates, clocking, and protocol
- Enable and disable links
- Initiate the transmission and reception of data on the serial lines

You can use these programs to verify that the installed devices and cables are functioning correctly. You can also use them as templates for designing client applications that use the embedded DLITE interface.

D.2 Hardware Setup for the Test Programs

Select a pair of adjacent ports to test. Ports are looped back in the following pairs: (0,1), (2,3), (4,5), and so on. Install a two-headed loopback cable between each pair of ports to be tested. You can test up to eight ports by using more cables; however, you must start with ports 0 and 1. For example, in Step 2 below you are asked how many ports you want to test. If you answer "6", you must install cables between ports (0,1), (2,3), and (4,5).

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 sample program: `freeway\client\[int_nt_emb` or `axp_nt_emb]\bin`. Enter one of the sample test commands shown in Table D–1 (for example, `ddcsync` or `awsasync`) at the system prompt:

Step 2: The following prompts are displayed:

`How many ports do you want to run on? (2 - 8):`

Enter the number of ports on which to run the test.

`How many messages do you want to send?:`

Enter the number of messages to send.

`What window size do you want?:`

For the non-blocking (asynchronous) program only, enter the window size.

`Verbose print? (Y/N):`

If you want verbose print, which traces the program flow through debug messages, enter "`y`".

Step 3: After you answer the last prompt, the test starts. It displays a spinner to indicate that it is running or a series of debug messages which trace the program flow if you selected verbose print in Step 2. 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 DDCMP blocking program (`ddcsync`). Figure D–2 shows the screen display from a sample DDCMP non-blocking program (`ddcasync`). The screen display for other protocols is similar. 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.

```
C:ddcsync
How many ports do you want to run on? (2 - 8) : 8
How many messages do you want to send? : 200
Verbose print? (Y/N) : n

starting threads and opening DLI sessions
writer for  port0 started
reader for  port1 started
writer for  port2 started
reader for  port3 started
writer for  port4 started
reader for  port5 started
writer for  port6 started
reader for  port7 started
5 seconds elapsed
 port0 sent     200 packets
 port2 sent     200 packets
 port4 sent     200 packets
 port6 sent     200 packets
-----WRITER FOR port0 COMPLETED
-----WRITER FOR port2 COMPLETED
-----WRITER FOR port4 COMPLETED
-----WRITER FOR port6 COMPLETED
 port1 received 200 packets
-----READER FOR port1 COMPLETED
 port3 received 200 packets
-----READER FOR port3 COMPLETED
 port5 received 200 packets
-----READER FOR port5 COMPLETED
 port7 received 200 packets
-----READER FOR port7 COMPLETED
Program Completed.
```

Figure D–1: Sample Output from DDCMP Blocking Multithreaded Program

```
C:ddcasync
How many ports do you want to run on? (2 - 8) : 8
How many messages do you want to send? : 200
What window size do you want? : 2
Verbose print? (Y/N) : n

starting threads and opening DLI sessions
writer for  port0 started
reader for  port1 started
writer for  port2 started
reader for  port3 started
writer for  port4 started
reader for  port5 started
writer for  port6 started
reader for  port7 started
5 seconds elapsed
 port3 received 200 packets
 port5 received 200 packets
 port7 received 200 packets
 port1 received 200 packets
 port2 sent     200 packets
 port4 sent     200 packets
 port6 sent     200 packets
 port0 sent     200 packets
-----WRITER FOR port2 COMPLETED
-----WRITER FOR port4 COMPLETED
-----WRITER FOR port6 COMPLETED
-----WRITER FOR port0 COMPLETED
-----READER FOR port3 COMPLETED
-----READER FOR port5 COMPLETED
-----READER FOR port7 COMPLETED
-----READER FOR port1 COMPLETED
Program Completed.
```

Figure D–2: Sample Output from DDCMP Non-Blocking Multithreaded Program

Index

Customer Report Form

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

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

Your Name: _____

Company: _____

Address: _____

_____

_____

Phone Number: _____

Product: _____

Problem or
Suggestion: _____

_____

_____

_____

_____

_____

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

PROTOGATE

ICP2432 User's Guide for Windows NT 4.0 and NT 5.0 (Windows 2000) (DLITE Interface)

DC 900-1514E